Andrew Pitonyak

BASIC-Makros für OpenOffice und LibreOffice

Von den Grundlagen zu konkreten Praxisbeispielen

Ins Deutsche übertragen und bearbeitet

von Volker Lenhardt

Mit Beiträgen von Andreas Heier und Volker Lenhardt

Letzte Änderung

Montag, 13. November 2023

Die amerikanische Originalausgabe erscheint unter dem Titel

OpenOffice.org Macros Explained, 4. ed.

ODT-Version: https://www.pitonyak.org/OOME_4_0.odt

Durchgesehene, korrigierte und ergänzte Ausgabe.

Vormals erschienen unter dem Titel „OpenOffice.org-Makros Erklärt“.

Anmerkung des Übersetzers

Für die Übersetzung habe ich OpenOffice.org, bzw. Apache OpenOffice (aktuell AOO 4.1.6) und LibreOffice (aktuell LO 6.4.5.2) auf dem Linux-Betriebssystem openSUSE Leap 15.2 (64 Bit) und der Benutzeroberfläche KDE (zur Zeit 5.18.5) verwendet. Zum Einsatz kamen auch macOS Catalina bzw. Big Sur (zur Zeit 11.1) und Windows 10 Home.

Ich habe die vom Autor auf Fedora (Gebietsschema en-US) verwendeten Menübefehle und Bildschirmfotos zum überwiegenden Teil durch die von LO und AOO (de-DE) ersetzt.

Ich habe in diesem Buch alle Variablennamen in Makros im allgemeinen in ihrer englischen Originalform belassen und habe sie kommentiert, wenn deren Bedeutung aus dem Zusammenhang nicht direkt erschlossen werden konnte. Wenn es mir aber notwendig schien, habe ich sie übersetzt.

Es gibt eine Reihe von Gründen, weshalb ich es für sinnvoll halte, auch im muttersprachlichen Umfeld für Variablen englische Bezeichner zu wählen: Verben ohne Flexionen, einsilbige Wörter, harmonischer Einklang mit den englischsprachigen Anweisungen, Objektmethoden und -eigenschaften, sowie – last but not least – internationale Hilfemöglichkeiten im Falle, dass man im Internet nachfragen muss, weil etwas nicht so funktioniert, wie man geglaubt hat.

Hier und da können Zahlen mit Tausenderpunkt und Dezimalzahlen für Verwirrung sorgen. Alle Zahlen in Basic-Anweisungen sind ohne Tausendertrenner und mit einem Dezimalpunkt zu schreiben. Die Ausgabe mit Print oder MsgBox verwendet jedoch die lokalisierte Form mit Dezimalkomma. Im laufenden Text schreibe ich Zahlen auf deutsche Weise mit Dezimalkomma und falls angebracht mit Tausenderpunkt, beim Zitieren von Basic-Anweisungen jedoch mit Dezimalpunkt.

Der Text ist in den Schriftarten Arial, Times New Roman und Courier New formatiert, da ich davon ausgehe, dass der überwiegende Teil der Leser auf MS Windows arbeitet, wo diese Schriften Standard sind. Auf Linux und macOS sind diese Schriften üblicherweise nicht installiert. Daher habe ich sie nicht nur in die PDF-Datei eingebettet, sondern auch in die ODT-Datei. Leider wird diese Option nur von LO, nicht aber von AOO unterstützt. In AOO kann es also sein, dass die konkret verwendete Ersatzschriftart abweichende Maße aufweist und dass daher abweichende Zeilen- und Seitenumbrüche resultieren. Als Linux-AOO-Nutzer sollten Sie also die PDF-Datei als Grundlage für Quellenangaben für Zitate wählen oder die oben genannten Schriften installieren.

Beim Zitieren sollten Sie immer auch das Datum der aktuellen Bearbeitung angeben.

Ich habe auf genderspezifische Formen verzichtet. Ich schließe in allen Fällen des generischen Maskulinums sowohl Frauen als auch Männer und Personen anderer geschlechtlicher Identifizierung mit ein.

Volker Lenhardt

Inhaltsverzeichnis

1. Einführung und Organisation

Am Anfang stand die erste Auflage von OpenOffice.org Macros Explained (OOME). Ein paar Jahre später stellte ich die zweite Auflage fertig, aktualisiert als Anpassung an die OpenOffice.org (OOo)-Version 2.x. Diese Aktualisierung wurde nie veröffentlicht, jedoch die nächste Überarbeitung, dann als dritte Auflage. Nun denke ich, wird es Zeit für die vierte Auflage.

Seit der letzten Veröffentlichung hat sich die Anzahl der von OOo unterstützten Services und Interfaces vervielfacht, und die Funktionalität ist erheblich erweitert worden. Leider ist der Leistungsumfang größer, als ich Zeit oder Raum habe zu dokumentieren. So umfangreich dieses Buch auch ist, es fehlt leider noch viel. Sie sollten dieses Buch daher als Nachschlagewerk mit einer Vielzahl an Beispielen nutzen, aber immer daran denken, dass OOo in einem kontinuierlichen Wandel steckt und immer wieder neue Funktionalitäten unterstützt.

Das Dokument enthält Schaltflächen zum Starten der Makros, die im Text vorgestellt werden. Das ist allerdings nur in der odt-Datei möglich. Da in einer pdf-Datei keine Makros ausgeführt werden können, werden die Schaltflächen dort auch nicht angezeigt.

1.1. In eigener Sache

Ich bin der Hauptautor dieses Dokuments, ich bestreite meinen Lebensunterhalt nicht mit der Arbeit mit OOo, und nichts in diesem Buch hat mit meinem Hauptberuf zu tun. Mit anderen Worten, ich bin einfach irgendein Mitglied der OOo-Gemeinschaft, der dies hier weitgehend ohne Entlohnung tut.

Ich erhalte zahllose Bitten um Hilfe, weil ich in der OOo-Gemeinschaft prominent bin. Unglücklicherweise ist meine Zeit schon über Gebühr beansprucht, und es ist schwierig, allen persönlich zu Hilfe zu kommen. Ich helfe gerne in meiner nicht vorhandenen Freizeit, aber bitte, nutzen Sie nach Möglichkeit schon vorhandenes Material, Mailinglisten und Foren. Gelegentlich biete ich Lösungen auf Vergütungsbasis an, aber für größere Projekte fehlt mir einfach die Zeit.

Ich begrüße Kommentare und Bug-Reports. Wenn Sie glauben, etwas Interessantes sollte mit aufgenommen werden, lassen Sie es mich wissen. Wenn Sie einen ganzen Abschnitt oder ein Kapitel schreiben wollen, tun Sie es. Wahrscheinlich werde ich Ihr Werk heftig bearbeiten. Ich bitte um Ihr Feedback und Anregungen.

1.2. Arbeitsumgebung und Kommentare

Die Hauptarbeit an diesem Buch wurde mit der offiziellen 64-Bit-Linuxversion unter Fedora Linux geleistet. Begonnen wurde das Werk mit OpenOffice.org (OOo), das nun in den beiden Linien LibreOffice (LO) und Apache OpenOffice (AOO) weitergeführt wird. Ich verwende in diesem Buch den ursprünglichen Namen OpenOffice oder OOo als Oberbegriff für jedes der drei Produkte.

AOO und LO sind unabhängig voneinander in ständigem Wandel, daher ähneln sich mit der Zeit die APIs, Funktionalitäten und Benutzerschnittstellen immer weniger. Es kann dazu führen, dass ein Makro, das in LO funktioniert, in AOO scheitert. Auch Bildschirmfotos und Menüpfade können von dem abweichen, was Sie mit Ihrer verwendeten Anwendungsversion vorfinden. Die Weiterentwicklung geht in so schnellem Tempo vor sich, dass meine begrenzte Zeit nicht ausreicht, mit der Dokumentation Schritt zu halten. Ich erhalte keine Vergütung für meine Arbeit an diesem Dokument.

Am besten probieren Sie es aus, was in Ihrer Arbeitsumgebung funktioniert. Inspizieren Sie auch die Objektinstanzen, um die von ihnen unterstützten Eigenschaften, Methoden, Konstanten, Enumerationen, Structs, Services und Interfaces zu erfahren.

2. Die Grundlagen

In OpenOffice.org (OOo) werden Makros und Dialoge in Dokumenten und Bibliotheken gespeichert. Die integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) dient zum Erstellen und zum Debuggen von Makros und Dialogen. Dieses Kapitel führt in die Grundkonzepte des Starts der IDE und der Makroerstellung ein. Es zeigt schrittweise den Aufbau eines einfachen Makros, das den Text „Hallo Welt“ auf dem Bildschirm anzeigt.

Ein Makro ist eine für den späteren Gebrauch gespeicherte Folge von Anweisungen oder Tastenkombinationen. Ein Beispiel für ein einfaches Makro wäre eines, das Ihre Adresse ausgibt. Makros unterstützen Anweisungen für eine Vielzahl weiterreichender Funktionen: Entscheidungen zu treffen (wenn zum Beispiel die Differenz kleiner ist als null, zeige sie in Rot, falls nicht, zeige sie in Schwarz), Schleifen zu durchlaufen (solange die Differenz größer ist als null, subtrahiere 10), und sogar mit einer Person zu kommunizieren (den Nutzer zum Beispiel nach einer Zahl zu fragen). Einige dieser Anweisungen basieren auf der Programmiersprache BASIC (Akronym für Beginner’s All-purpose Symbolic Instruction Code). Üblicherweise bindet man ein Makro an eine Tastenkombination oder ein Werkzeugleisten-Symbol, um es schnell starten zu können.

Ein Dialog – oder Dialogfenster – ist ein Fenstertyp für den „Dialog“ mit einem Nutzer. Der Dialog mag dem Nutzer Informationen bieten oder eine Eingabe vom Benutzer erwarten. Sie können Ihre eigenen Dialoge erschaffen und sie in einem Modul mit Ihren Makros speichern.

Die OOo-Makrosprache ist sehr flexibel. Sie erlaubt die Automatisierung sowohl einfacher als auch komplexer Aufgaben. Obwohl es viel Spaß machen kann, Makros zu schreiben und die internen Abläufe von OOo kennenzulernen, ist es nicht immer der beste Ansatz. Makros sind vor allem nützlich, wenn dieselbe Aufgabe immer wieder durchgeführt werden muss, oder wenn man einen einzigen Tastendruck wünscht für etwas, das normalerweise mehrere Schritte benötigt. Hier und da wird man auch ein Makro schreiben, um etwas zu tun, dass man ansonsten in OOo nicht tun kann, aber in einem solchen Fall sollte man vorher gründlich recherchieren, um sich zu vergewissern, dass OOo es auch wirklich nicht kann. Zum Beispiel findet man in einigen OOo-Mailinglisten regelmäßig die Bitte um ein Makro zum Entfernen leerer Absätze. Diese Funktionalität steht über die AutoKorrektur zur Verfügung (**Extras > AutoKorrektur > AutoKorrektur-Optionen > Optionen**: *Leere Absätze entfernen*). Man kann auch reguläre Ausdrücke verwenden, um Leerzeichen zu suchen und zu ersetzen. Es gibt eine Zeit und einen Anlass für Makros und eine Zeit für andere Lösungen. Dieses Kapitel wird Sie auf die Zeiten vorbereiten, wenn ein Makro das Mittel der Wahl ist.

2.1. Makrospeicherung

In OOo werden logisch zusammengehörende Prozeduren in einem Modul gespeichert. Ein Modul könnte zum Beispiel Prozeduren enthalten zum Auffinden typischer Fehler, die weitere Bearbeitung erfordern. Logisch zusammenhängende Module werden in einer Bibliothek gespeichert, und Bibliotheken wiederum werden in Bibliothekscontainern gespeichert. Die OOo-Anwendung kann, wie auch jedes OOo-Dokument, als Bibliothekscontainer fungieren. Zusammenfassend gesagt, die OOo-Anwendung und jedes OOo-Dokument können Bibliotheken, Module und Makros enthalten.

Container	Ein Bibliothekscontainer enthält keine, eine oder mehrere Bibliotheken.
Bibliothek	Eine Bibliothek enthält keine, eine oder mehrere Module und Dialoge.
Modul	Ein Modul enthält keine, eine oder mehrere Subroutinen oder Funktionen.

2.1.1. Bibliothekscontainer

Ein OOo-Dokument ist ein Bibliothekscontainer, genauso wie die Anwendung als ganze. Wenn ein bestimmtes Dokument ein Makro benötigt, speichert man das Makro praktischerweise in dem Dokument. Das hat den Vorteil, dass das Makro beim Dokument bleibt. So kann man auch ganz einfach Makros versenden.

Wenn jedoch verschiedene Dokumente dasselbe Makro benötigen und jedes Dokument eine Kopie davon hätte und Sie dann Änderungen an dem Makro vornehmen wollten, müssten Sie es in jedem Dokument ändern, das dieses Makro enthält. In einem Dokument gelagerte Makros sind nur für dieses Dokument sichtbar. Es ist daher nicht einfach, ein Makro in einem Dokument von außerhalb des Dokuments aufzurufen.

Tipp Speichern Sie keine Makros in einem Dokument, die von außerhalb des Dokuments aufgerufen werden (von seltenen Ausnahmen abgesehen), denn Makros in einem Dokument sind nur für dieses Dokument sichtbar.

Der Bibliothekscontainer der Anwendung besteht aus zwei Hauptkomponenten: mit OOo installierte Makros und von Ihnen selbst erstellte Makros. Der Basic-Makros-Dialog zeigt Ihre Makros in einem Container mit dem Namen „Meine Makros“ und die mitgelieferten als „LibreOffice Makros“ (s. Bild 1). LibreOffice-Makros werden in einem Verzeichnis der Anwendungsinstallation gespeichert, „Meine Makros“ hingegen in Ihrem Nutzerverzeichnis.

Über **Extras > Makros > Makros verwalten > Basic...** öffnen Sie den Dialog für Basic-Makros (s. Bild 1). Die Bibliothekscontainer sind die Objekte der obersten Stufe in der Spalte „Makro aus“.

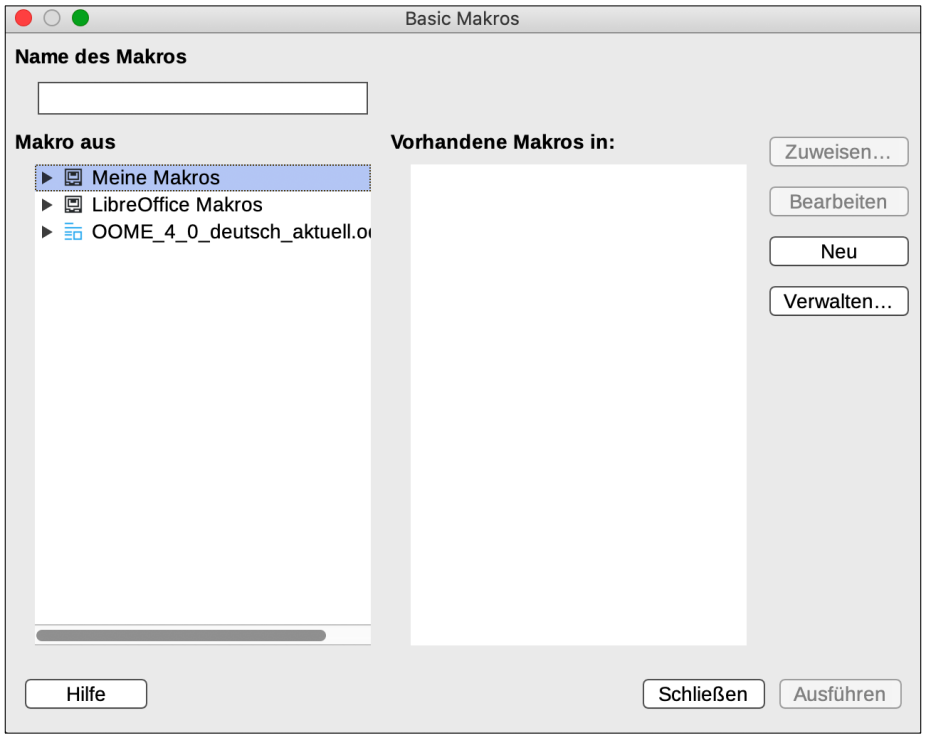


Bild 1. *Im Dialog „Basic Makros“ legen Sie neue Makros an und organisieren die Bibliotheken.*

2.1.2. Bibliotheken

Ein *Bibliothekscontainer* enthält eine oder mehrere Bibliotheken, und eine *Bibliothek* enthält ein oder mehrere Module und Dialoge. Doppelklicken Sie auf einen Bibliothekscontainer im Bild 1, um die enthaltenen Bibliotheken zu sehen. Doppelklicken Sie auf eine Bibliothek, um die Bibliothek zu laden und die enthaltenen Module und Dialoge zu sehen.

Der Dialog „Basic Makros“ zeigt geladene Bibliotheken mit einem anderen Symbol an als ungeladene, vorausgesetzt, der verwendete Symbolstil unterstützt es. Im Bild 2 sind Standard, XMLProcs, XrayTool und OOME_40 geladen, die anderen Bibliotheken nicht. Für dieses Bild musste ich den Symbolstil Galaxy einstellen, da der von mir bevorzugte Stil Breeze keinerlei Unterschiede zeigt.

Tipp Die Symbole und Farben auf Ihrem Rechner können sich von denen der Bildschirmfotos unterscheiden. Unterschiedliche OOo-Versionen können unterschiedliche Symbole und Farben nutzen, und es wird mehr als nur ein Symbol-Set unterstützt. Über **Extras > Optionen > LibreOffice > Ansicht** können Sie Größe und Stil der Symbole einstellen.

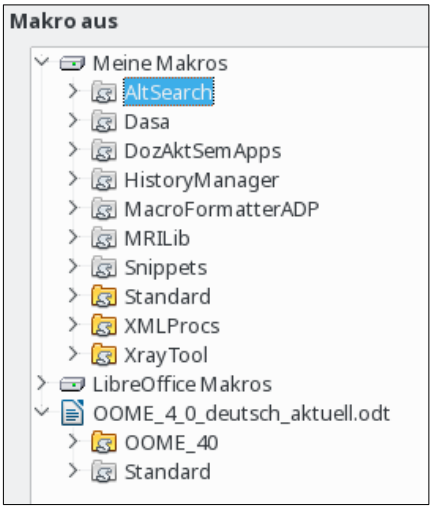


Bild 2. *Geladene Bibliotheken werden anders angezeigt.*

2.1.3. Module und Dialoge

In einem Modul werden typischerweise ähnliche Funktionalitäten auf einer niedrigeren Stufe als einer Bibliothek gruppiert. Die Makros werden in den Modulen gespeichert. Um ein neues Modul anzulegen, markieren Sie eine Bibliothek und klicken auf Neu.

2.1.4. Kernpunkte

Zu beachten:

- Sie können Bibliotheken von einem Bibliothekscontainer in einen anderen importieren.
- Sie importieren ein Modul dadurch, dass Sie die Bibliothek importieren, die das Modul enthält. Es ist über das GUI nicht möglich, einfach ein einzelnes Modul zu importieren.
- Geben Sie den Bibliotheken, Modulen und Makros anschauliche Namen. Anschauliche Namen reduzieren die Wahrscheinlichkeit von Namenskollisionen, die den Bibliotheksimport behindern.
- Die Standardbibliothek spielt eine Sonderrolle: sie wird automatisch geladen, so dass die enthaltenen Makros immer verfügbar sind.
- Die Standardbibliothek wird automatisch angelegt und kann nicht importiert werden.
- Makros in einer Bibliothek sind erst verfügbar, nachdem die Bibliothek geladen ist.
- Über den „Basic Makros“-Verwaltungsdialog kann man neue Module anlegen, aber keine neuen Bibliotheken.

Diese Kernpunkte habe gewisse Konsequenzen. Zum Beispiel speichere ich Makros selten in der Standardbibliothek, weil ich die Bibliothek nicht woanders hin importieren kann. Ich verwende die Standardbibliothek normalerweise für Makros, die über Schaltflächen in einem Dokument gestartet werden. Die Makros in der Standardbibliothek laden dann die eigentlichen Arbeitsmakros in anderen Bibliotheken und starten sie.

2.2. Neue Module und Bibliotheken anlegen

Im Dialog „Basic Makros“ erstellt die Schaltfläche „Neu“ das Gerüst einer neuen Subroutine in der ausgewählten Bibliothek (s. Bild 1 und Bild 2). Wenn die Bibliothek noch kein Modul enthält, wird nach einer Namensabfrage ein neues angelegt.

Über die Schaltfläche „Verwalten...“ öffnen Sie die Makroverwaltung „Basic Makro-Organizer“ (s. Bild 3). Die Registerkarten Module und Dialoge sind nahezu identisch. Aktivieren Sie sie, um Module oder Dialoge anzulegen, zu löschen oder umzubenennen.

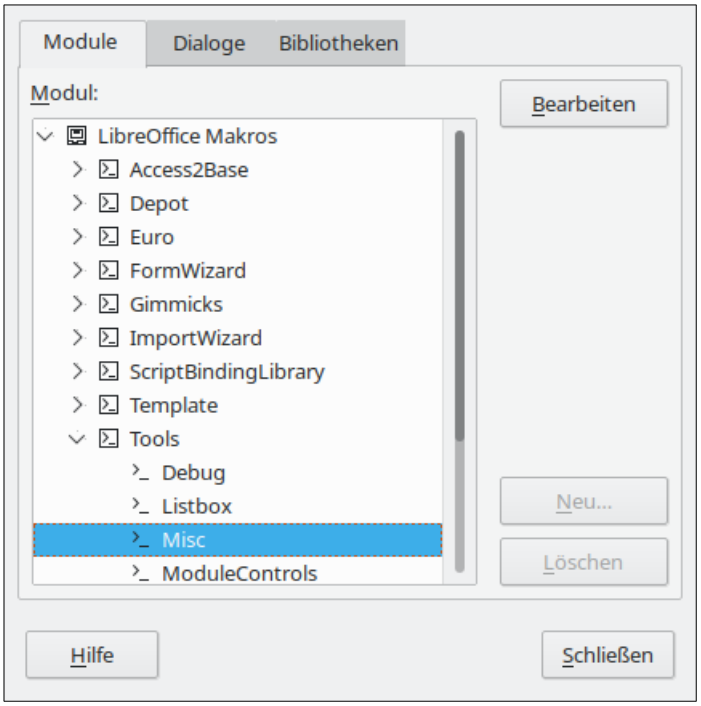


Bild 3. *Die Registerkarte Module der Basic-Makroverwaltung.*

Verwenden Sie die Registerkarte Bibliotheken (s. Bild 4), um Bibliotheken anzulegen, zu löschen, umzubenennen, zu importieren oder zu exportieren.

Achtung Das Umbenennen einer Bibliothek kann Folgen haben. Als ich die Bibliothek „OOME_30“ in diesem Dokument in „OOME_40“ umbenannte, verloren alle Schaltflächen, die Makros aufrufen, ihr Verknüpfungsziel. Dasselbe gilt für alle Makros, die zum Aufruf eines anderen Makros explizit den Bibliotheksnamen verwenden. Ich fand keinen besseren Weg, als die Datei mit unzip auszupacken, alle Referenzen in einem Texteditor zu korrigieren, und die Datei danach wieder zu zippen.

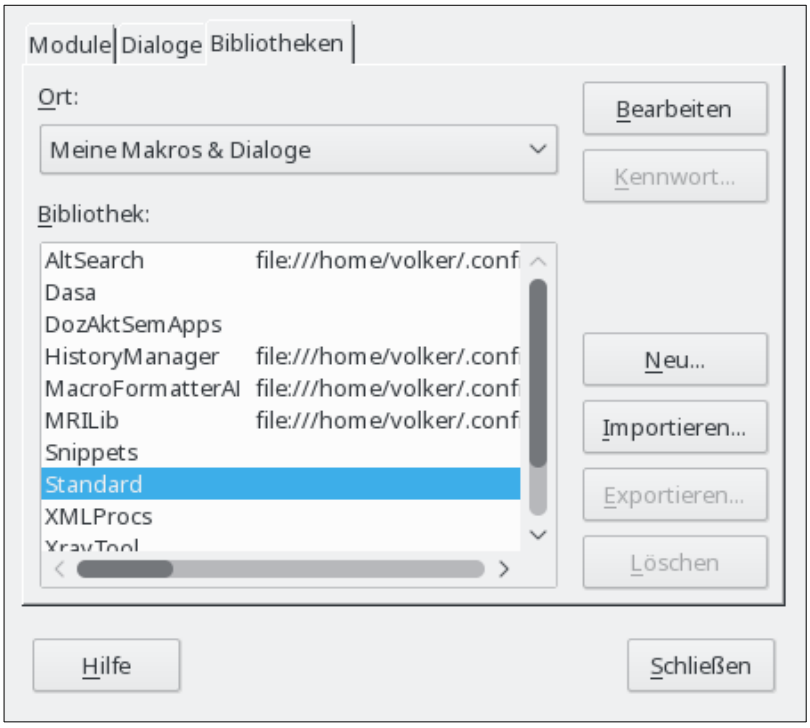


Bild 4. *Die Registerkarte Bibliotheken der OOo-Makroverwaltung.*

Aus der Aufklappliste „Ort“ wählen Sie den gewünschten Bibliothekscontainer. Um eine Bibliothek umzubenennen, doppelklicken Sie auf die Bibliothek und ändern den Namen direkt.

Achtung Das Umbenennen von Modulen und Bibliotheken in der Makroverwaltung kann frustrierend sein. Um eine Bibliothek umzubenennen, klicken Sie doppelt oder dreifach auf den Namen und warten dann ein paar Sekunden. Versuchen Sie es wieder. Und noch einmal. Wechseln Sie die Bibliothek. Klicken Sie wieder doppelt oder dreifach auf den Namen. Sie merken schon, worum es geht.

Den einfachsten Weg, einen Modulnamen zu ändern, finden Sie in der IDE. Rechtsklicken Sie auf den Modulnamen am unteren Rand und wählen Umbenennen (s. Bild 11).

2.3. Makrosprache

Die OOo-Makrosprache basiert auf der Programmiersprache BASIC. Die Standardmakrosprache heißt offiziell StarBasic, wird aber auch als OOo Basic oder nur Basic bezeichnet. Viele andere Programmiersprachen können zur Automatisierung von OOo genutzt werden. Es gibt praktische Unterstützung für Makros, die in Basic, JavaScript, Python und BeanShell geschrieben sind. In diesem Dokument liegt mein Hauptinteresse auf Basic.

2.4. Ein Modul in einem Dokument anlegen

Jedes OOo-Dokument ist ein Bibliothekscontainer, der Makros und Dialoge enthalten kann. Wenn ein Dokument die Makros enthält, die es benutzt, bedeutet folglich der Besitz des Dokuments auch den Besitz der Makros. Das ist eine praktische Methode für die Weitergabe und Speicherung. Senden Sie das Dokument an jemand anderen oder an einen anderen Speicherort, sind die Makros immer erreichbar und nutzbar.

1. Um einem Dokument ein Makro hinzuzufügen, müssen Sie das Dokument zur Bearbeitung öffnen. Öffnen Sie als erstes ein neues Textdokument, das den Namen „Unbenannt 1“ erhält – vorausgesetzt, dass kein anderes Dokument ohne Namen geöffnet wurde.
2. Über **Extras > Makros > Makros verwalten > LibreOffice Basic** öffnen Sie den Dialog für Basic-Makros (s. Bild 1).

3. Klicken Sie auf „Verwalten...“, um die Makro-Verwaltung zu öffnen, und klicken dann auf die Registerkarte Bibliotheken (s. Bild 4).
4. Wählen Sie „Unbenannt 1“ aus der Ort-Aufklappliste.

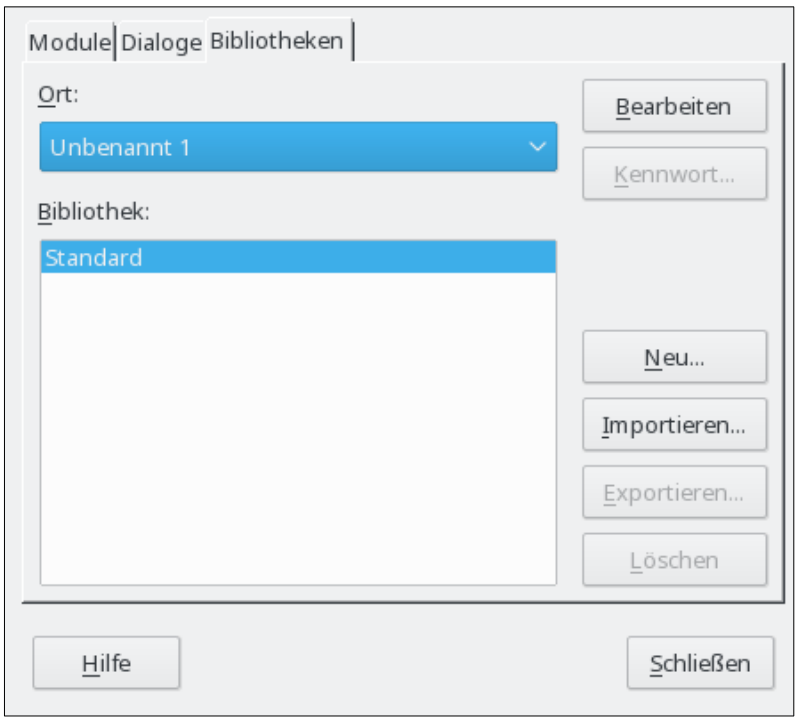


Bild 5. *Die Registerkarte Bibliotheken der Makroverwaltung.*

5. Klicken Sie auf Neu, um den Dialog für eine neue Bibliothek zu öffnen.

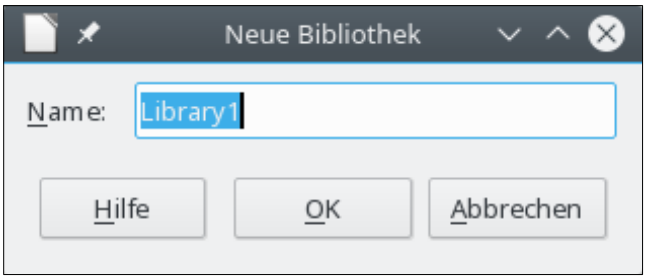


Bild 6. *Der Dialog Neue Bibliothek.*

6. Der Standardname ist Library1, nicht gerade aussagekräftig. Verwenden Sie einen aussagekräftigen Namen und klicken Sie OK. Die neue Bibliothek steht nun in der Liste. Für dieses Beispiel habe ich die Bibliothek „HalloWelt“ genannt.

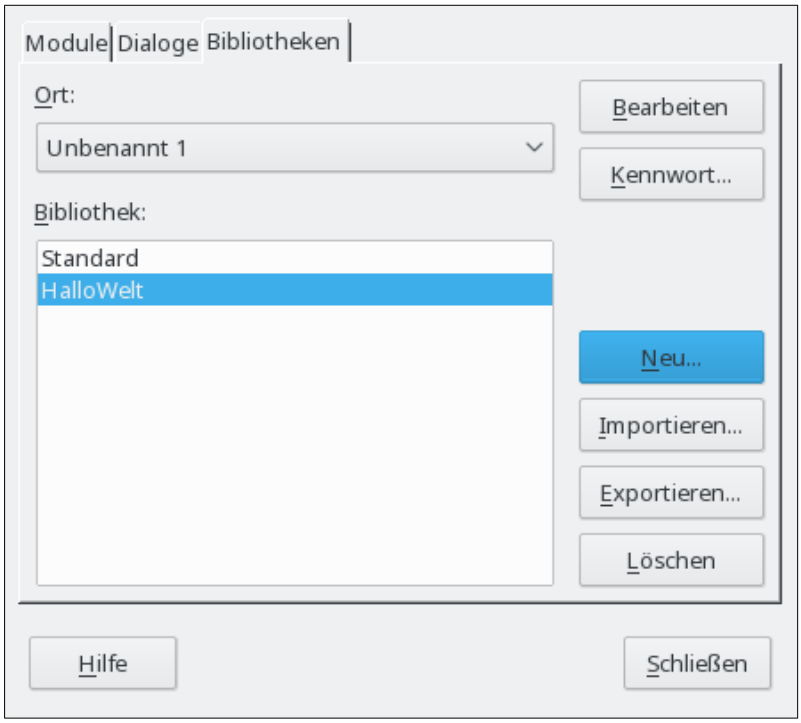


Bild 7. *Die neue Bibliothek steht in der Liste.*

7. In der Registerkarte Module wählen Sie die Bibliothek HalloWelt. OOo hat das Modul namens „Module1“ beim Anlegen der Bibliothek erstellt.

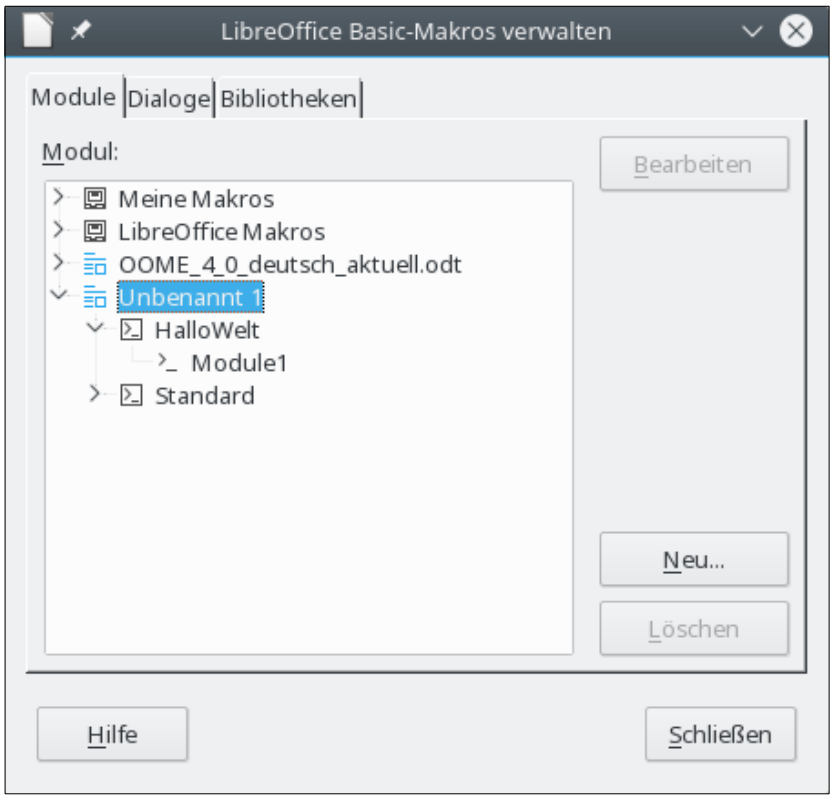


Bild 8. *Die neue Bibliothek enthält automatisch das Modul „Module1“.*

8. Mit einem Klick auf Neu öffnen Sie den Dialog Neues Modul. Der Standardname ist Module2, weil Module1 schon existiert.

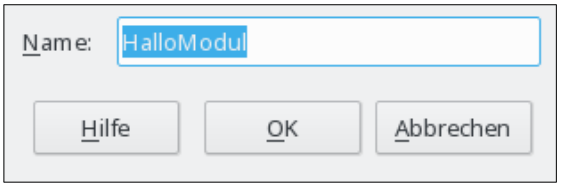


Bild 9. *Der Dialog Neues Modul.*

9. Geben Sie einen aussagekräftigen Namen ein klicken auf OK.

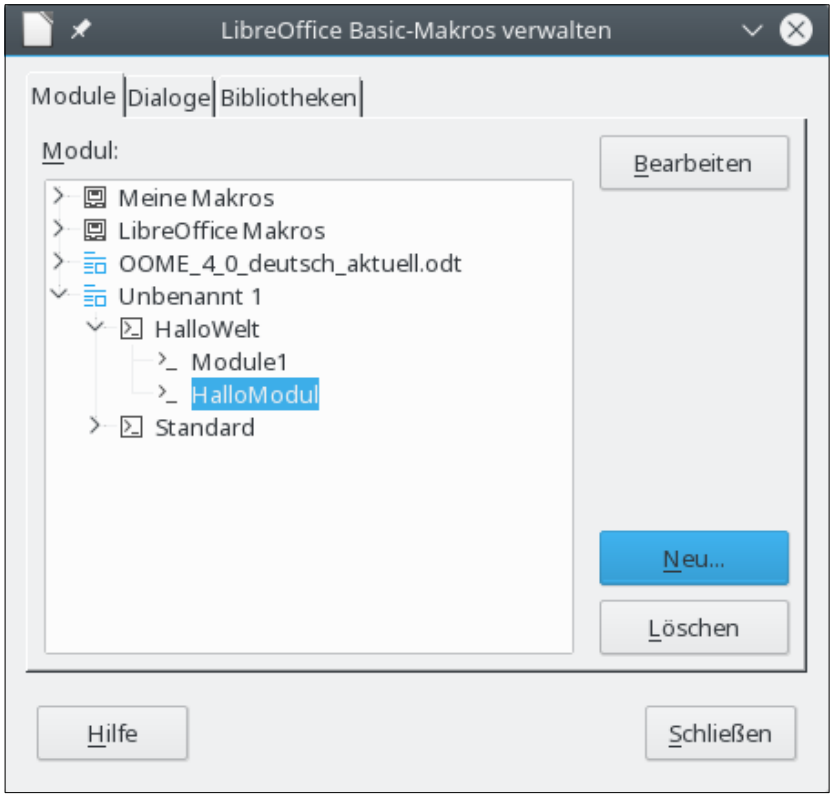


Bild 10. *Das neue Modul steht in der Liste.*

10. Markieren Sie HalloModul und klicken auf Bearbeiten.
11. An diesem Punkt habe ich das Dokument unter dem Namen „WegMitMir“ gespeichert, weil ich es löschen werde, wenn ich mit dem Beispiel fertig bin. Nennen Sie es nach Ihren Wünschen. Wenn Sie dann den Dialog im Bild 10 neu öffnen, wird statt „Unbenannt 1“ der aktuelle Name des Dokuments angezeigt.

Jetzt wird die Integrierte Entwicklungsumgebung (Integrated Debugging Environment, IDE) zum Editieren des Makros geöffnet.

2.5. Integrierte Entwicklungsumgebung (Integrated Debugging Environment)

In der Integrierten Entwicklungsumgebung (IDE) für Basic erstellen Sie Makros und führen sie aus (s. Bild 11). Die IDE bietet wesentliche Funktionalitäten auf kleinem Raum. Die Symbole der Werkzeugleiste sind in der Tabelle 1 erläutert. Links über dem Bearbeitungsfenster finden Sie eine Aufklappliste [WegMitMir.odt].HalloWelt mit der Anzeige der aktuellen Bibliothek. Der Bibliothekscontainer steht in eckigen Klammern, gefolgt von der Bibliothek. So kann man schnell eine Bibliothek auswählen. Die linke Seite des Fensters bietet Ihnen die Möglichkeit, über den Objektkatalog schnell zu einem Bibliothekscontainer, einer Bibliothek oder einem Modul zu wechseln.

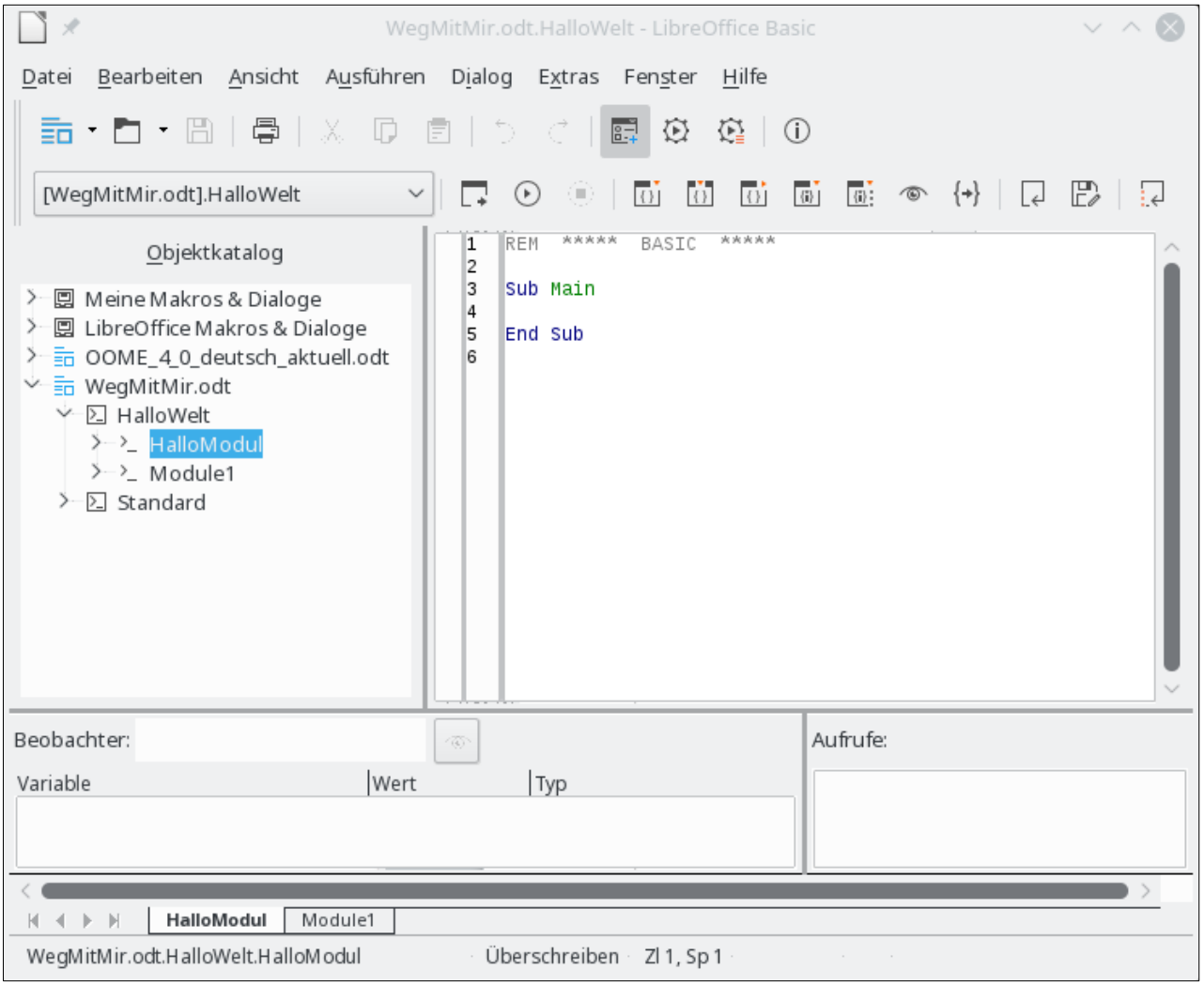


Bild 11. *Die Integrierte Entwicklungsumgebung für Basic.*

Lassen Sie den Mauscursor ein paar Sekunden auf einem Symbol der Werkzeugleiste ruhen, um den beschreibenden Text zu lesen, der einen Hinweis auf die dahinter liegende Funktion gibt.

Tabelle 1. *Symbole der Werkzeugleiste in der Basic-IDE.*

Symbol	Taste	Beschreibung
	Ctrl+N	Neues Dokument anlegen.
	Ctrl+O	Vorhandenes Dokument öffnen.
	Ctrl+S	Aktuelle Bibliothek speichern. Wenn die Bibliothek zu einem Dokument gehört, dann wird das ganze Dokument gespeichert.
	Ctrl+P	Makro zum Drucker senden.
	Ctrl+X	Ausschneiden und in die Zwischenablage einfügen.
	Ctrl-C	In die Zwischenablage kopieren.
	Ctrl+V	Aus der Zwischenablage einfügen.
	Ctrl+Z	Letzte Aktivität rückgängig machen.

Symbol	Taste	Beschreibung
	Ctrl+Y	Letzte rückgängig gemachte Aktivität wiederherstellen.
		Objektkatalog öffnen (s. Bild 12). Doppelklicken Sie auf das ausgewählte Makro.
		Makrodialog öffnen (s. Bild 2). Wählen Sie ein Makro aus und klicken Sie auf Bearbeiten oder Ausführen. Shortcut für **Extras > Makros > Makros verwalten > LibreOffice Basic**.
		Modul auswählen. Öffnet die Makroverwaltung und zeigt die Registerkarte Module an (s. Bild 3). Wählen Sie ein Modul und klicken auf Bearbeiten.
		OOo-Hilfe öffnen. Sie enthält eine Vielzahl nützlicher Beispiele für Basic.
		Kompilierungssymbol zum Prüfen von Syntaxfehlern. Eine Meldung erscheint nur, wenn ein Fehler gefunden wurde. Es wird nur das aktuelle Modul kompiliert.
	F5	Ausführen des ersten Makros im aktuellen Modul (AOO), in LO ist es das Makro, in dem sich der Cursor befindet. Ist es angehalten (von einem Haltepunkt oder bei Einzelschritten), wird es fortgesetzt. Um ein anderes Makro auszuführen, nutzen Sie  . Damit öffnen Sie den Makrodialog. Wählen Sie das gewünschte Makro und klicken auf Ausführen.
	Shift+F5	Stoppen Sie das gerade laufende Makro.
	Shift+F8	Prozedurschritt zur nächsten Anweisung. Wenn ein Makro an einem Haltepunkt steht, wird die aktuelle Anweisung ausgeführt. Dient auch zum Start eines Makros im Einzelschritt-modus.
	F8	Einzelschritt. Wie Prozedurschritt, außer dass der Schritt, wenn die aktuelle Anweisung ein anderes Makro aufruft, in dieses Makro hineinführt, so dass die Beobachtung dort weiter-geht.
	Ctrl+ Shift+F8	Rücksprung. Führt das Makro bis zum Ende der aktuellen Subroutine oder Funktion aus.
	Shift+F9	Haltepunkt ein/aus an der aktuellen Cursorposition in der IDE. Links neben der Zeile wird ein Symbol () platziert, das zeigt, dass für diese Zeile ein Haltepunkt gesetzt ist. Sie können auch in diesen Haltepunktbereich doppelklicken, um einen Haltepunkt ein- oder auszuschalten.
		Öffnet den Dialog zur Verwaltung der Haltepunkte (s. Bild 17), mit dem Sie wahlweise Haltepunkte ein- oder ausschalten können. Zusätzlich können Sie einstellen, nach wie vie-len Durchläufen ein Haltepunkt erstmalig ausgelöst wird.
	F7	Beobachter-Symbol. Fügt die Variable unter dem aktuellen Cursor in der IDE in das Beob-achterfenster ein. Sie können alternativ den Variablennamen in die Beobachter-Einfügezei-le eintragen und Enter drücken.
		Klammernsuche.
		Basic-Quellcode einfügen. Öffnet einen Dialog zur Suche einer Datei mit Basic-Code, der dann eingefügt wird.
		Das aktuelle Modul als Textdatei speichern. OOo speichert Module auf der Festplatte in ei-nem speziellen Format. Auf diesem Weg erstellte Dateien sind Standardtextdateien. Diese Form eignet sich ausgezeichnet für Makro-Backups oder zum Versenden an andere Perso-nen. Im Gegensatz dazu wird über das Speichern-Symbol die gesamte Bibliothek oder das Dokument mit dem Modul gespeichert.
		Import eines Dialogs aus einem anderen Modul.

Die Modulnamen sind entlang dem unteren Ende der IDE aufgeführt. Links von den Namen sind Schaltflächen zur Modulnavigation  . Die Schaltflächen führen zum ersten, vorherigen, folgenden und letzten Modul der aktuellen Bibliothek. Rechtsklicken Sie auf einen Modulnamen, um:

- Ein neues Modul oder einen neuen Dialog einzufügen.
- Ein Modul oder einen Dialog zu löschen.
- Ein Modul umzubenennen. Ich kenne keinen einfacheren Weg, ein Modul oder einen Dialog umzubenennen.
- Ein Modul oder einen Dialog auszublenden.
- Die Basic-Makroverwaltung zu öffnen.

Mit  öffnen Sie den Objektkatalog (s. Bild 12). Doppelklicken Sie auf das Makro, das Sie bearbeiten wollen.

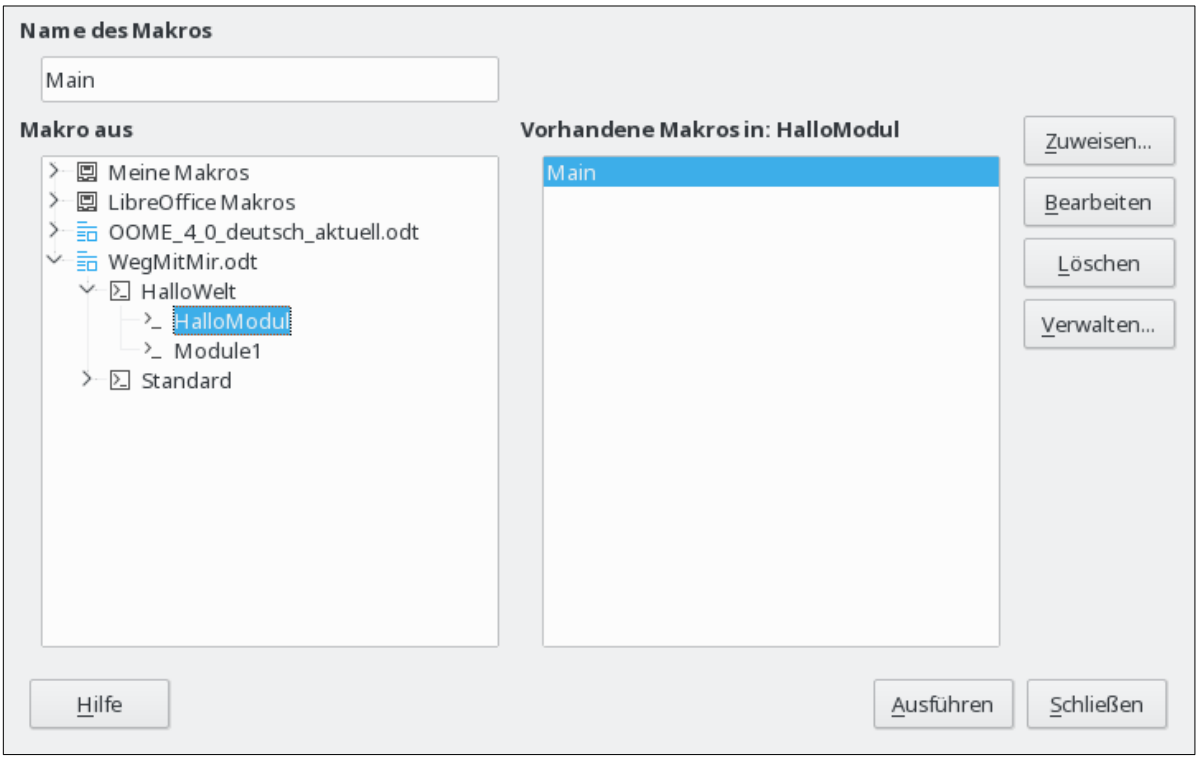


Bild 12. *Der Objektkatalog.*

2.6. Das Makro eingeben

Überschreiben Sie den Text in der IDE gemäß Listing 1. Klicken Sie auf das Ausführen-Symbol.

Listing 1. *Das Makro Hallo Welt*

```
REM  *****  BASIC  *****
Option Explicit

Sub Main
  Print "Hallo Welt"
End Sub
```

Tabelle 2. *Erläuterung der Zeilen in Listing 1.*

Zeile	Erläuterung
`REM ***** BASIC *****`	Basic-Kommentar, diese Zeile wird ignoriert. Ein Kommentar kann auch mit einem einfachen Anführungszeichen (= Apostroph) beginnen.
`Option Explicit`	Teilt dem Basic-Übersetzer mit, dass es ein Fehler ist, eine Variable zu verwenden, die nicht explizit deklariert ist. Schreibfehler in Variablennamen können sonst leicht zu einem Laufzeitfehler führen.

Zeile	Erläuterung
`Sub Main`	Beginn der Definition der Subroutine mit dem Namen Main.
`Print "Hallo Welt"`	Print-Anweisung.
`End Sub`	Ende der Subroutine Main.

Das Makro im Listing 1 ist ein Text, den ein Mensch lesen kann. Die Anwendung muss den Makrotext in eine Form bringen, die der Rechner ausführen kann. Diesen Vorgang nennt man Kompilieren. Ein Makro wird also zeilenweise als Text geschrieben und dann kompiliert, so dass ein ausführbares Programm entsteht.

2.7. Ein Makro ausführen

Das Ausführen-Symbol startet immer das erste Makro des aktuellen Moduls (AOO) beziehungsweise das Makro, in dem sich der Cursor befindet (LO). Daraus folgt, dass ein anderer Weg gebraucht wird, wenn das Modul mehr als ein Makro enthält. Folgende Optionen stehen zur Auswahl:

- Schieben Sie das Makro an den Anfang des Moduls (AOO) beziehungsweise platzieren Sie den Cursor in das auszuführende Makro (LO) und klicken dann auf das Ausführen-Symbol.
- Rufen Sie mit dem ersten Makro das gewünschte Makro auf. Ich nutze diese Methode gerne während der Entwicklung. Ich habe immer ein Main-Makro als erstes, das nichts tut. Aber während der Entwicklung ändere ich es dahin, dass es das gerade relevante Makro aufruft. Ganz allgemein lasse ich das Main-Makro ganz oben das am meisten ausgeführte Makro aufrufen.
- Gehen Sie über den Makrodialog (s. Bild 2), um eine beliebige Routine im Modul auszuführen.
- Fügen Sie Ihrem Dokument oder einer Werkzeugleiste eine Schaltfläche zu, über die Sie das Makro ausführen.
- Binden Sie das Makro an eine Tastenkombination. Öffnen Sie den Anpassen-Dialog über **Extras > Anpassen**. In der Registerkarte Tastatur finden Sie die Makrobibliotheken ganz unten in der Liste Kategorien. Ein anderer Weg geht über **Extras > Makros > Makros verwalten > Basic**. Wählen Sie das Makro aus und klicken auf „Zuweisen...“. Damit öffnet sich der Anpassen-Dialog, in dem Sie über entsprechende Registerkarten das Makro als Menüpunkt, von einer Werkzeugleiste oder als Systemereignis ausführen lassen können.

Über den Makrodialog eine beliebige Subroutine in einem Modul auszuführen, geht so:

1. Über **Extras > Makros > Makros verwalten > Basic** öffnen Sie den Makrodialog (s. Bild 2).
2. Suchen Sie das Dokument mit dem gewünschten Modul in der Liste „Makro aus“.
3. Doppelklicken Sie auf eine Bibliothek, um deren enthaltene Module anzuzeigen.
4. Wählen Sie das Modul aus. In der Liste „Vorhandene Makros in: <selektierter Modulname>“ finden Sie die enthaltenen Subroutinen und Funktionen.
5. Wählen Sie die zum Ausführen gewünschte Subroutine oder Funktion aus – zum Beispiel HelloWorld.
6. Klicken Sie zum Ausführen der Subroutine oder der Funktion auf die Schaltfläche Ausführen.

2.8. Makrosicherheit

Je nachdem, wie OOo konfiguriert ist, kann es möglich sein, dass Sie keine Makros in einem Dokument ausführen dürfen. Wenn ich ein Dokument öffne, das ein Makro enthält, erscheint die Warnung Bild 13. Wenn Sie kein Makro erwarten oder der Quelle nicht trauen, deaktivieren Sie die Makros. Bedenken Sie, schließlich bin ich in der Lage, ein Makro zu schreiben, das Ihren Rechner zerstört.

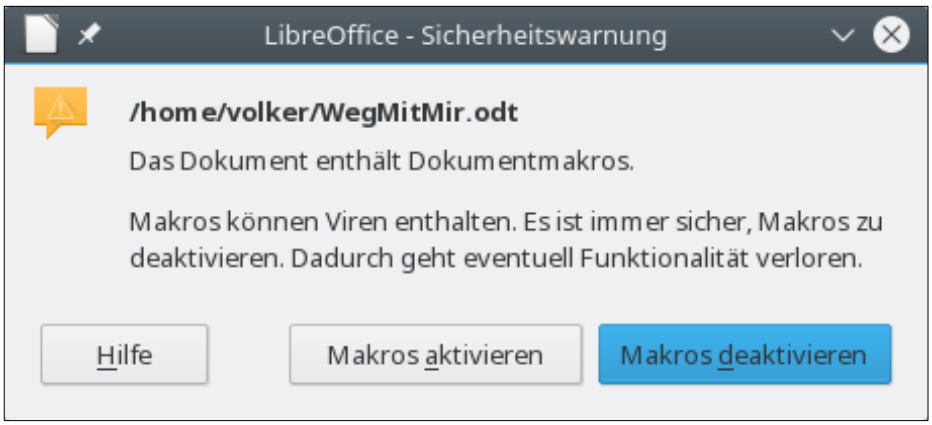


Bild 13. *Das geöffnete Dokument enthält ein Makro.*

Über **Extras > Optionen > LibreOffice > Sicherheit** öffnen Sie das Sicherheitsblatt des Optionen-Dialogs.

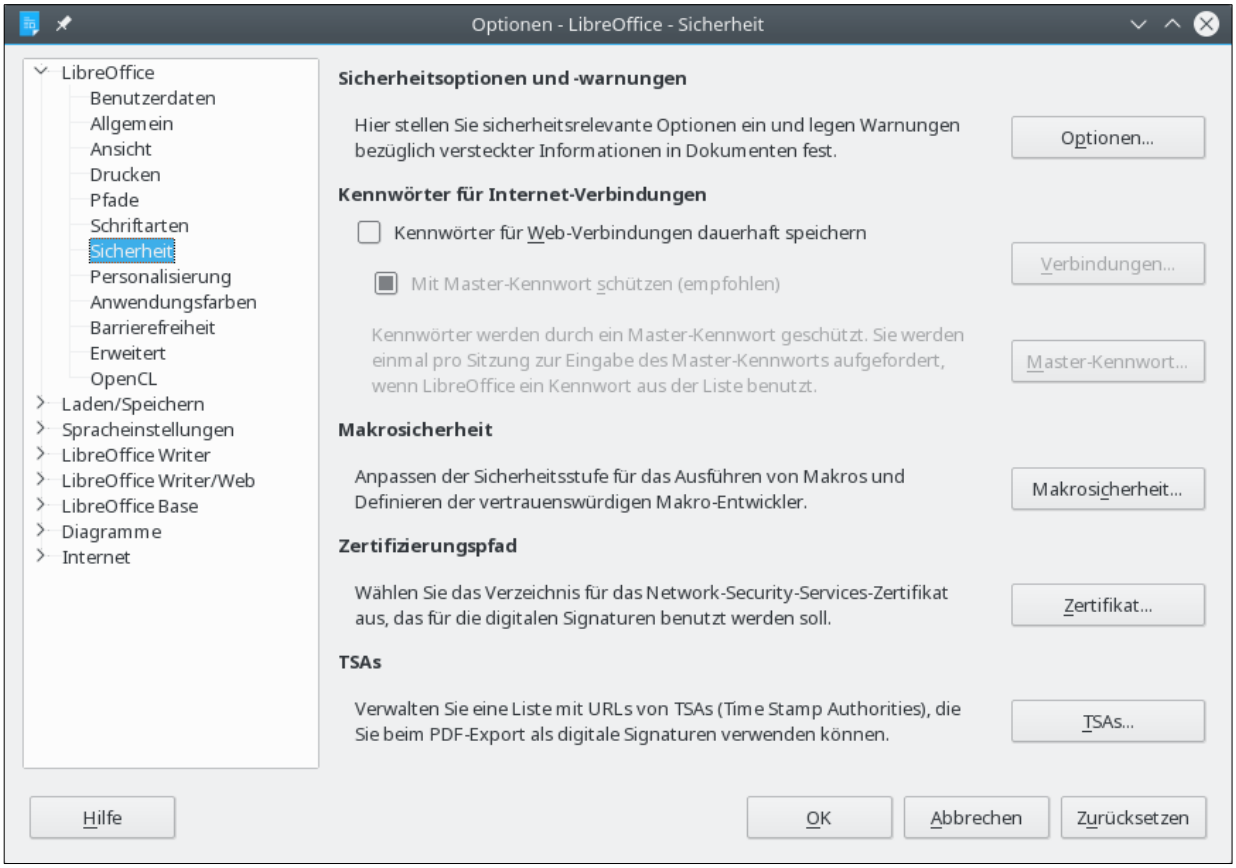


Bild 14. *Der Optionen-Dialog, Sicherheitseinstellungen.*

Klicken Sie auf die Schaltfläche Makrosicherheit, um den Makrosicherheitsdialog zu öffnen. Wählen Sie eine Sicherheitsstufe, bei der Ihnen wohl ist. Die mittlere Sicherheitsstufe verwendet die im Bild 13 gezeigte Rückfrage. Das ist unaufdringlich und relativ schnell erledigt.

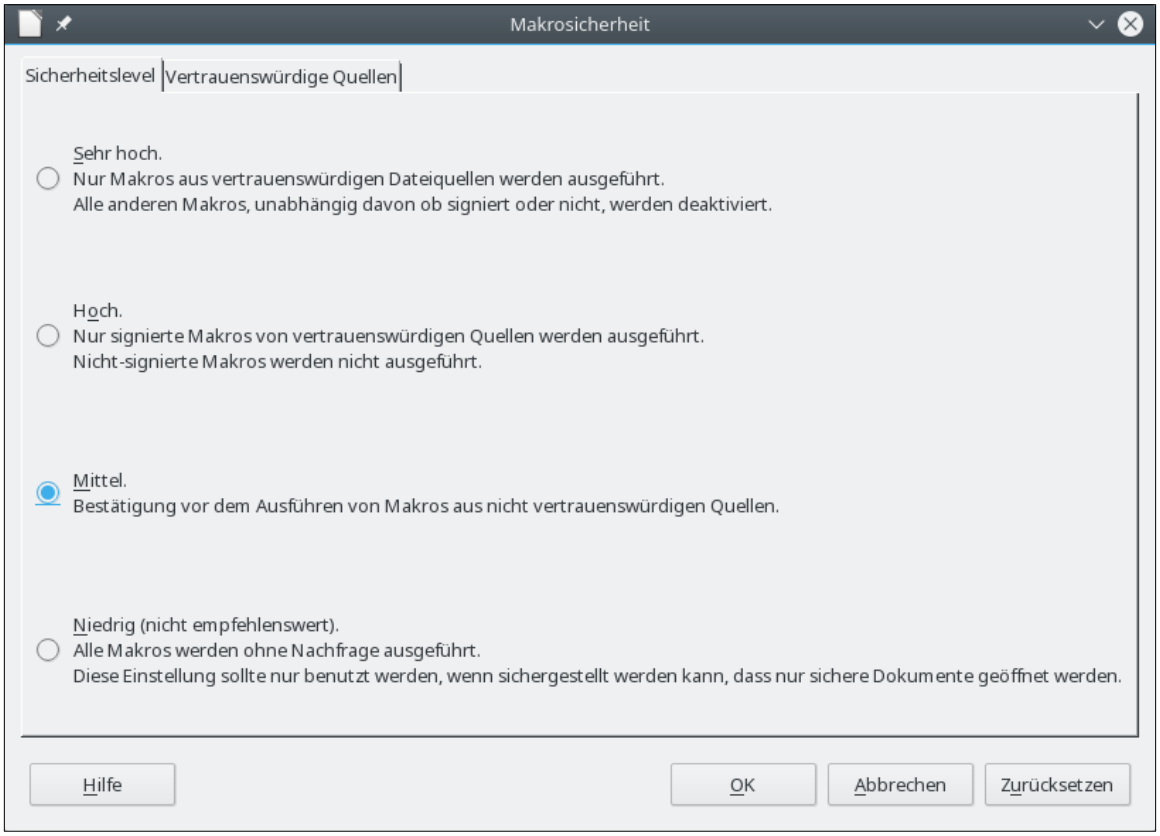


Bild 15. *Der Dialog Makrosicherheit, Registerkarte Sicherheitsstufe.*

Sie können vertrauenswürdige Quellen und Zertifikate eintragen, um Dokumente ohne Sicherheitsabfrage zu laden, entweder je nach Speicherort oder nach dem für das Dokument ausgestellten Zertifikat.

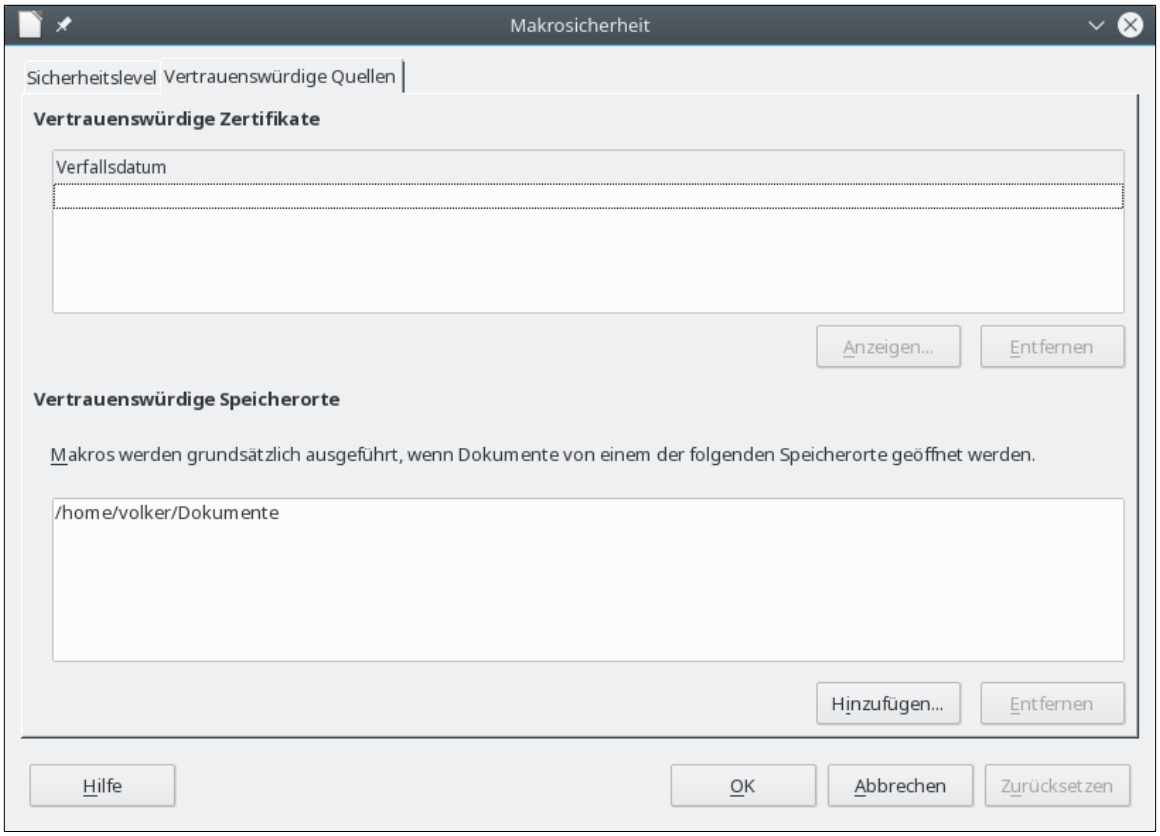


Bild 16. *Der Dialog Makrosicherheit, Registerkarte Vertrauenswürdige Quellen.*

2.9. Haltepunkte einsetzen

Wenn Sie im Quelltext einen Haltepunkt setzen, bleibt das Makro bei der Ausführung an dieser Stelle stehen. Sie können dann Variablenwerte überprüfen, die Ausführung des Makros komplett oder in Einzelschritten fortsetzen. Wenn ein Makro fehlerhaft arbeitet und Sie nicht wissen warum, erlaubt Ihnen der Einzelschrittmodus (jeweils 1 Anweisungsschritt), das Makro bei der Arbeit zu beobachten. Sie wissen dann also, wie es zu dem Fehler kam. Falls aber vor der problematischen Stelle eine Menge Anweisungszeilen stehen, kann es sehr beschwerlich sein, in Einzelschritten dorthin zu gelangen. In solchen Fällen setzen Sie an oder in der Nähe der Stelle einen Haltepunkt, an der der Fehler auftritt. Das Programm hält an der Stelle an, so dass Sie nun in Einzelschritten das weitere Verhalten des Makros beobachten können.

Das Symbol Haltepunkt-Ein/Aus setzt einen Haltepunkt an die Anweisung, auf dem der Cursor steht. Ein rotes Stoppschild kennzeichnet die Zeile in der Haltepunktspalte. Doppelklicken Sie in die Haltepunktspalte, um einen Haltepunkt an der Anweisungszeile zu setzen oder zu entfernen. Ein Rechtsklick auf einen Haltepunkt in der Haltepunktspalte aktiviert oder deaktiviert ihn.

Mit dem Symbol zur Haltepunktverwaltung  gelangen Sie zu dem Dialog, in dem alle aktiven Haltepunkte im aktuellen Modul mit der Zeilennummer aufgelistet sind. Um einen weiteren Haltepunkt zu setzen, geben Sie eine Zeilennummer in das Eingabefeld ein und klicken auf Neu. Um einen Haltepunkt zu löschen, markieren Sie ihn in der Liste und klicken auf Löschen. Um den markierten Haltepunkt zu deaktivieren, ohne ihn zu löschen, nehmen Sie den Haken aus dem Ankreuzfeld Aktiv. Das Eingabefeld Durchlauf zeigt an, wie oft ein Haltepunkt erreicht werden muss, bis er aktiviert wird. Bei der Durchlaufzählung 4 wird die Anweisung mit dem Haltepunkt nicht ausgeführt, wenn die Zeile zum vierten Mal erreicht wird: das Makro wird angehalten. Das ist außerordentlich wichtig, wenn ein Makrobereich erst dann einen Fehler produziert, wenn er mehrfach aufgerufen wurde.

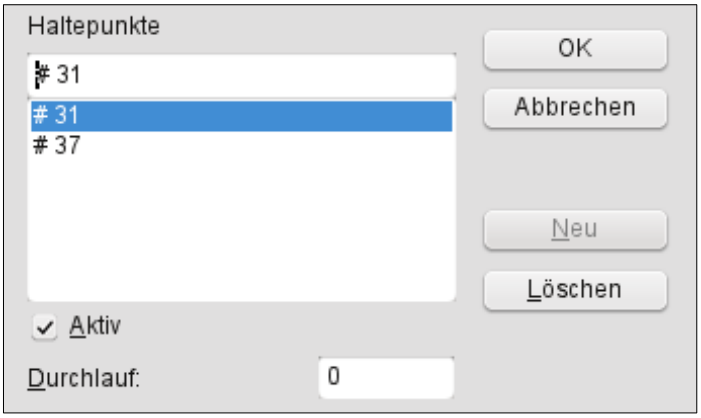


Bild 17. *Der Dialog zur Verwaltung der Haltepunkte.*

Ein Haltepunkt kann aus zwei Gründen ignoriert werden: eine Durchlaufzählung größer als null und wenn der Haltepunkt in der Haltepunkt-Verwaltung als „nicht aktiv“ gekennzeichnet ist. Zu jedem Haltepunkt gehört eine Zählung, die beim Durchlauf in Richtung null heruntergezählt wird. Wenn der Wert null erreicht ist, wird der Haltepunkt aktiv und bleibt es, weil die Durchlaufzählung danach auf null bleibt. Die Durchlaufzählung wird beim Beenden oder Neustart des Makros nicht auf den Originalwert zurückgesetzt.

2.10. Wie Bibliotheken gespeichert werden

Makrobibliotheken werden als XML-Dateien gespeichert, die man leicht mit einem einfachen Texteditor bearbeiten kann. Mit anderen Worten, es ist ganz einfach, darin herumzustöbern und die Dateien unbrauchbar zu machen. Dieses Kapitel ist eigentlich für Fortgeschrittene, so dass Sie es vielleicht überschlagen wollen. Wenn Sie nichts von XML verstehen und nicht wissen, weshalb in der Datei > anstatt > steht, sollten Sie die Datei vielleicht nicht bearbeiten. Obwohl es im allgemeinen als töricht gilt, die externen Bibliotheken manuell zu bearbeiten, hatte ich wenigstens einmal den Fall, wo es doch notwendig war, weil eine OOo-Version ein Modul wegen eines darin enthaltenen Syntaxfehlers nicht laden konnte.

Jede Bibliothek wird in einem einzelnen Verzeichnis gespeichert, und jedes Modul wie auch jeder Dialog in einer einzelnen Datei. Die mit OOo installierten globalen Bibliotheken werden in einem von allen Nutzern verwendeten Verzeichnis basic unterhalb des Verzeichnisses gespeichert, in dem OOo installiert ist. Beispiele:

```
C:\Programme\OpenOffice3.2\share\basic       'Eine Windows-Installation
/opt/openoffice.org/basis3.2/share/basic     'Eine Linux-Installation
```

OOo speichert nutzerspezifische Daten in einem Verzeichnis unterhalb des Home-Verzeichnisses des Nutzers. Der genaue Ort hängt vom Betriebssystem ab. Über **Extras > Optionen > LibreOffice > Pfade** können Sie nachlesen, wo andere Konfigurationsdaten gespeichert sind. Hier sind einige Beispiele, wo die Basic-Makros des Übersetzers liegen:

```
C:\Users\volker\AppData\Roaming\LibreOffice\4\user\basic                  'Windows 10
/Users/volker/Library/Application Support/LibreOffice/4/user/basic       'macOS
/home/volker/.config/libreoffice/4/user/basic                            'Linux
```

Nutzermakros werden in LibreOffice/4/user/basic gespeichert. Jede Bibliothek befindet sich in einem eigenen Verzeichnis unterhalb des Verzeichnisses basic. Das Verzeichnis user enthält zwei Dateien und je ein Verzeichnis für jede Bibliothek (s. Tabelle 3).

Tabelle 3. *Dateien und ein paar Verzeichnisse in meinem Verzeichnis user/basic.*

Eintrag	Beschreibung
dialog.xlc	XML-Datei, die auf jede Dialogdatei zeigt, die diesem Nutzer bekannt ist.

Eintrag	Beschreibung
script.xlc	XML-Datei, die auf jede Bibliotheksdatei zeigt, die diesem Nutzer bekannt ist.
Standard	Verzeichnis der Standardbibliothek.
Pitonyak	Verzeichnis der Bibliothek mit dem Namen Pitonyak.
PitonyakDialogs	Verzeichnis der Bibliothek mit dem Namen PitonyakDialogs.

Die Dateien dialog.xlc und script.xlc enthalten je einen Zeiger auf alle Dialoge und Bibliotheken, die OOo kennt. Wenn diese Dateien überschrieben werden, weiß OOo nichts mehr von Ihren persönlichen Bibliotheken, auch wenn sie noch existieren. Sie können jedoch die Dateien manuell bearbeiten, oder besser noch, mit dem OOo-Makroverwalter die Bibliotheken importieren (weil Sie eine Bibliothek importieren können, die in einem Verzeichnis liegt).

Das Verzeichnis einer Bibliothek enthält je eine Datei für jedes Modul und für jeden Dialog der Bibliothek. Das Verzeichnis enthält außerdem die Dateien dialog.xlb und script.xlb, die auf die Dialoge bzw. Module zeigen.

2.11. Wie Dokumente gespeichert werden

Die Standard-OOo-Dateiformate speichern alle Komponenten als Standard-ZIP-Dateien. Mit jedem Programm, das ZIP-Dateien lesen und auspacken kann, ist die Untersuchung von OOo-Dokumenten möglich – für manche Programme müssen Sie jedoch die Dateinamenergänzung in ZIP ändern.

Nach dem Auspacken eines OOo-Dokuments erhalten Sie Dateien, die den Hauptinhalt, die Formatvorlagen und die Einstellungen enthalten, und außerdem noch drei Verzeichnisse. Das Verzeichnis META-INF zeigt auf alle anderen Dateien, eingebetteten Grafiken, Bibliotheken mit Makroquelltext und Dialoge. Das Verzeichnis Dialogs enthält alle eingebetteten Dialoge, und das Verzeichnis Basic enthält alle eingebetteten Bibliotheken.

Der entscheidende Punkt daran ist, dass Sie im Notfall die XML-Daten manuell einsehen und möglicherweise Probleme beheben können. Sie können die Datei aber auch so vermasseln, dass sie nicht mehr geöffnet werden kann. Verwenden Sie also besser eine Kopie und nicht das Original.

2.12. Fazit

Makros und Dialoge werden in Modulen gespeichert, Module in Bibliotheken, und Bibliotheken in Bibliothekscontainern. Die Anwendung ist ein Bibliothekscontainer, ebenso jedes Dokument. Mit Hilfe der IDE werden Makros und Dialoge entwickelt und getestet.

Zum Schreiben von Makros für OOo haben Sie nun einen der schwierigsten Schritte getan: Sie haben Ihr erstes Makro geschrieben! Jetzt sind sie so weit, andere Makrobeispiele zu untersuchen und auszuprobieren, sowie ein paar eigene zu entwickeln.

3. Sprachstrukturen

Die OOo-Makrosprache ist der von Microsoft Office ähnlich, weil beide auf BASIC beruhen. Beide Makrosprachen greifen auf die Strukturen der zugrunde liegenden Anwendung zu, die sich ganz wesentlich unterscheiden und daher nicht kompatibel sind. Dieses Kapitel greift die Teile der Sprache heraus, die nicht auf die zugrunde liegende Anwendung zugreifen.

Dieses Kapitel zeigt, wie verschiedene Komponenten zusammenkommen, um ein Makro zu produzieren, das übersetzt (kompiliert) und ausgeführt werden kann. In einem Wort: Syntax. Korrekte Syntax bedeutet nicht unbedingt, dass das Makro tut, was Sie wollen. Es bedeutet nur, dass die Einzelteile in korrekter Art und Weise zusammengefügt sind. Die Fragen „Kann ich fahren?“ und „Darf ich fahren?“ sind beide syntaktisch korrekt. Die erste Frage bezieht sich auf die Fähigkeit, die zweite Frage auf die Erlaubnis. In der Umgangssprache können beide Fragen dieselbe Bedeutung haben. Der Rechner andererseits tut nur genau, was man ihm sagt, und nicht, was man meint.

Die syntaktischen Grundkomponenten eines OOo-Makros sind Anweisungen, Variablen, Subroutinen, Funktionen und Programmflusskontrollen. Eine Anweisung ist ein kleiner, ausführbarer Teil des Codes, der im allgemeinen eine einzelne Textzeile umfasst. Variablen sind Behälter und enthalten Werte, die sich während der Makroausführung ändern können. Subroutinen und Funktionen unterteilen ein Makro in benannte Gruppen von funktional zusammengehörenden Anweisungen und ermöglichen so eine bessere organisatorische Struktur. Die Programmflusskontrollen steuern, welche Anweisungen ausgeführt werden und wie oft.

OOo führt Makros zeilenweise aus. Jede Makrozeile wird genau so begrenzt, wie es klingt: durch eine Zeilenumbruch-Kennung (s. Listing 2).

Listing 2. *Makro aus zwei Zeilen*

```
Print "Dies ist Zeile eins"
Print "Dies ist Zeile zwei"
```

Überlange Anweisungen können über mehr als eine Textzeile gehen, indem ein Unterstrich (_) an das Ende der Textzeile gefügt wird (s. Listing 3). Damit der Unterstrich zur Zeilenverbindung wird, muss er das letzte Zeichen der Textzeile sein. Der Unterstrich hat keine Sonderbedeutung, wenn er nicht am Zeilenende steht, er kann innerhalb von Zeichenfolgen und Variablennamen genutzt werden. Wenn er als Zeilenverbinder dient, können ihm Leerzeichen vorausgehen, in manchen Fällen sind Leerzeichen sogar notwendig, um den Zeileninhalt vom Unterstrich zu trennen. Wenn Sie zum Beispiel die Zeile „a+b+c“ hinter dem b teilen, benötigen Sie ein Leerzeichen zwischen dem b und dem Unterstrich. Ansonsten würde der Unterstrich als Teil des Variablennamens interpretiert. Achtung bei Leerzeichen, die versehentlich einem Zeilenverbinder folgen, sie können der Grund für einen Fehler bei der Kompilierung sein. Unglücklicherweise sagt der Fehler nichts darüber aus, dass etwas dem Unterstrich folgt, sondern nur, dass die nächste Zeile ungültig ist.

Listing 3. *Ein Unterstrich am Zeilenende setzt die Zeile logisch mit der nächsten Zeile fort.*

```
Print "Zeichenfolgen werden verkettet, indem sie " & _
      "mit dem Kaufmanns-Und verbunden werden"
```

Tipp Wenn auf ein Zeichen zur Zeilenverbindung irgendwelche Zeichen folgen, wird die nächste Zeile nicht als Fortsetzung erkannt. Wenn ich Quelltexte von Websites kopiere und sie in die IDE einfüge, wird manchmal ein Leerzeichen am Zeilenende eingefügt, das die Zeilenverbindung auflöst.

Sie können in einer Textzeile mehrere Anweisungen unterbringen, indem Sie sie mit Doppelpunkten abtrennen. Man macht das gewöhnlich zur besseren Lesbarkeit. Jede dieser kombinierten Anweisungen wirkt beim Testen des Makros in der Integrierten Entwicklungsumgebung (IDE) als eine eigene Codezeile. Die Zeile in Listing 4 verhält sich wie drei getrennte Anweisungen, wenn man in der IDE das Makro im Einzelschrittmodus testet.

Listing 4. *Die Variablen x, y und z werden auf null gesetzt.*

```
x = 0 : y = 0 : z = 0
```

Sie sollten in allen Makros, die Sie schreiben, nicht an Kommentaren sparen. Denken Sie beim Schreiben daran, dass das, was heute klar ist, morgen nicht mehr so klar sein könnte, denn die Zeit vergeht, neue Projekte entstehen, und das Gedächtnis schwindet nur allzu schnell. Sie leiten einen Kommentar entweder durch ein einfaches Anführungszeichen (Apostroph, Hochkomma) oder das Schlüsselwort REM (remark = Kommentar) ein. Der gesamte Text, der in dieser Zeile folgt, wird ignoriert. Kommentare gelten nicht als ausführbare Anweisungen. Sie werden auch im Einzelschrittmodus übergangen.

Listing 5. *Fügen Sie allen Makros, die Sie schreiben, Kommentare zu.*

```
REM Kommentare können mit dem Schlüsselwort REM beginnen.
ReM Groß- und Kleinschreibung spielen keine Rolle. Dies ist also auch ein Kommentar.
' Alles was dem Kommentarstart folgt, wird ignoriert.
X = 0 ' Ein Kommentar kann auch mit
      ' einem Apostroph beginnen.
z = 0 REM Alles was dem Kommentarstart folgt, wird ignoriert.
```

Tipp Groß- und Kleinschreibung der Schlüsselwörter, Variablen- und Routinennamen sind in Basic nicht von Belang.

Daher gelten sowohl REM, Rem oder rEm alle als Start eines Kommentars.

Einem Zeichen zur Zeilenverbindung (_) darf kein weiteres Zeichen folgen, auch kein Kommentar. Alles was einem Kommentarstart folgt, wird ignoriert – auch ein Zeilenverbinder. Die logische Folge dieser beiden Regeln ist, dass ein Zeilenverbinder niemals auf derselben Zeile mit einem Kommentar vorkommen kann.

3.1. Kompatibilität mit Visual Basic

Bezogen auf die Syntax und die BASIC-Funktionalität sind sich StarBasic und Visual Basic sehr nahe. Die beiden Basic-Dialekte haben ganz und gar nichts miteinander gemein, wenn es darum geht, Dokumente zu manipulieren, aber der Grundbestand der Befehle ist sehr ähnlich. Die allgemeine Kompatibilität zwischen den beiden Dialekten ist stufenweise verbessert worden.

Die Kompilierungsoption „Option Compatible“ steuert einige Besonderheiten zur Kompatibilität mit Visual Basic. Diese Option wirkt sich nur auf das Modul aus, in dem sie steht. Weil ein Makro während der Ausführung auch andere Module aufruft, können sowohl das alte als auch das neue Verhalten resultieren, abhängig davon, ob in den jeweils aufgerufenen Modulen „Option Compatible“ enthalten ist. Die Option in einem Modul zu setzen, hat also keinen Effekt in einem anderen aufgerufenen Modul.

Die Laufzeitfunktion CompatibilityMode(True/False) bietet die Möglichkeit, Laufzeitfunktionen während der Makroausführung zu modifizieren. Somit erhält man die Flexibilität, das neue Laufzeitverhalten einzuschalten, einige Operationen vorzunehmen und dann das neue Laufzeitverhalten wieder auszuschalten. CompatibilityMode(False) hebt Option Compatible für das neue Laufzeitverhalten auf. Es ist zu hoffen, dass eine Methode zur Prüfung des aktuellen Laufzeitverhaltens geschaffen wird.

Visual Basic erlaubt alle im Zeichensatz Latin-1 (ISO 8859-1) enthaltenen Zeichen in Variablennamen, StarBasic nicht. Wenn Sie „Option Compatible“ setzen, wird „ä“ zu einem gültigen Variablennamen. Dies ist nur eine der Änderungen, die durch „Option Compatible“ ermöglicht werden. Die Funktion CompatibilityMode() kann die neuen weiter gefassten Bezeichner weder aktivieren noch deaktivieren, weil CompatibilityMode() erst zur Laufzeit aufgerufen wird und Variablennamen schon beim Kompilieren erkannt werden.

Sowohl Visual Basic als auch StarBasic kennen die Anweisung RmDir(), um ein Verzeichnis zu löschen. VBA kann nur leere Verzeichnisse löschen, wohingegen StarBasic einen gesamten Verzeichnisbaum rekursiv löschen kann. Wenn die Funktion CompatibilityMode(True) vor der Anweisung RmDir() aufgerufen wird, wird StarBasic wie VBA agieren und eine Fehlermeldung ausgeben, wenn das spezifizierte Verzeichnis nicht leer ist. Dies ist nur eine der vielen Änderungen, die durch CompatibilityMode() ermöglicht werden.

StarBasic lässt viel mehr durchgehen als VBA. Es ist daher leichter, einfache Makros von VBA nach StarBasic zu konvertieren. Dazu zwei Beispiele:

In StarBasic ist der Zuweisungsbefehl „set“ optional. Daher ist „`set x = 5`“ sowohl in VBA als auch in StarBasic gültig, wogegen „`x = 5`“ in VBA ein Fehler ist, in StarBasic aber funktioniert.

Das zweite Beispiel betrifft die Arraymethoden. Sie sind weit stabiler und toleranter in OOo als in VBA. So arbeiten die Funktionen zur Ermittlung der Arraygrenzen (LBound und UBound) problemlos mit leeren Arrays, VBA hingegen stürzt dabei ab.

In LO können Sie mit der modulweiten Anweisung „Option VBASupport 1“ die VBA-Kompatibilität weiter erhöhen. „Option VBASupport 0“ schaltet sie für das Modul ab. Option Compatible und Compatibility(True) werden automatisch mit eingeschlossen. Beachten Sie dabei, dass dann auch Prozeduraufrufe der VBA-Syntax unterliegen. Das heißt, die Argumentliste einer Sub-Prozedur darf nicht in runden Klammern stehen, es sei denn, sie wird mit dem Befehl „Call“ aufgerufen. Ebenso werden die Klammern um die Argumentliste einer Function nur bei der Zuweisung zu einer Variablen gesetzt. Sie können sich aber auch daran gewöhnen, diese Syntax immer zu verwenden, denn StarBasic akzeptiert sie ohne Murren.

```
Option VBASupport 1
MySub arg1, arg2           'Richtig
Call MySub(arg1, arg2)     'Richtig
MySub(arg1, arg2)          'Falsch. Fehler beim Kompilieren
MyFunction arg1, arg2      'Richtig
x = MyFunction(arg1, arg2) 'Richtig
MyFunction(arg1, arg2)     'Falsch. Fehler beim Kompilieren
```

3.2. Kompilieroptionen und -direktiven

Der Compiler übersetzt das Makro in eine Form, die der Rechner zur Ausführung braucht. Das Verhalten des Compilers kann gesteuert werden durch solche Angaben wie „Option Explicit“ am Beginn des Moduls vor allen Variablen, Subroutinen und Funktionen. Eine Kompilieroption steuert die Kompilierung nur für das Modul, in dem sie steht.

Tabelle 4. *Kompilieroptionen und -direktiven*

Option	Beschreibung
Def	Gibt nicht-deklarierten Variablen einen Typ, der auf dem Namen der Variablen basiert.
Option Base	Setzt den Index des ersten Array-Elements auf 0 oder 1, falls er nicht spezifiziert ist.
Option Compatible	Lässt StarBasic eher wie VBA agieren.
Option VBASupport 1|0	Weitgehende Unterstützung für VBA-Makros, schließt Option Compatible und Compatibility(True) mit ein. Schalter 1 aktiviert die Option, Schalter 0 schaltet sie ab. Nur LO.
Option Explicit	Erzwingt, dass alle Variablen definiert werden. Wenn das Makro während des Laufs auf eine nicht definierte Variable stößt, wird ein Laufzeitfehler erzeugt.

3.3. Variablen

Variablen sind Behälter für Werte. OOo unterstützt verschiedene Typen von Variablen für unterschiedliche Wertetypen. Dieses Kapitel zeigt, wie man Variablen erstellt, benennt und benutzt. Ob-

wohl Sie von StarBasic nicht gezwungen werden, Variablen zu deklarieren, sollten Sie dennoch jede Variable deklarieren, die Sie benutzen. Die Gründe dafür werden in diesem Kapitel verdeutlicht.

3.3.1. Namen für Variablen, Routinen, Konstanten und Sprungmarken

Geben Sie Ihren Variablen immer aussagekräftige Namen. Auch wenn Ihnen der Variablenname „var1“ viel Nachdenken erspart, ist doch beispielsweise „Vorname“ viel sprechender. Es gibt aber auch Variablennamen, die zwar nicht wirklich sprechend, doch bei den Programmierern weit verbreitet sind. Zum Beispiel wird „i“ als Kürzel für „index“ verwendet, und zwar für eine Variable zur Durchlaufzählung einer Schleife. Folgende Einschränkungen gelten in Basic für Variablennamen:

- Offiziell darf ein Variablenname nicht mehr als 255 Zeichen enthalten. Ich habe aber Namen mit mehr als 300 Zeichen ohne Probleme getestet, doch das soll keine Empfehlung sein!
- Das erste Zeichen eines Variablennamens muss ein Buchstabe sein: A-Z oder a-z. Mit der Einstellung `Option Compatible` werden alle im Zeichensatz Latin-1 (ISO 8859-1) als Buchstaben definierten Zeichen als Teil eines Bezeichners akzeptiert.
- Die Ziffern 0-9 und der Unterstrich (_) dürfen in Variablennamen vorkommen, aber nicht als erstes Zeichen. Ein Unterstrich am Ende eines Variablennamens wird nicht als Zeilenverbinder missverstanden.
- Bei Variablennamen spielt die Groß- und Kleinschreibung keine Rolle. Sowohl „VorName“ als auch „vorNAME“ verweisen auf dieselbe Variable.
- Wenn Variablennamen in eckige Klammern eingeschlossen werden, dürfen sie auch Leerzeichen oder andere Latin-1-Zeichen enthalten, sogar als erstes Zeichen. Zum Beispiel sind erlaubt: [=5?], [6mal löschen], [suchen], [" zu ']. Eigentlich gilt so etwas als schlechte Programmierpraxis, doch manchmal sind solche Möglichkeiten nur zu verführerisch. Probieren Sie einfach einmal aus, was alles erlaubt ist (ein Apostroph als erstes Zeichen ist beispielsweise nicht möglich).

Tipp Diese Einschränkungen gelten auch bei Namen für Konstanten, Subroutinen, Funktionen und Sprungmarken.

3.3.2. Variablen deklarieren

Einige Programmiersprachen verlangen, dass Sie alle Variablen vor ihrer erstmaligen Verwendung ausdrücklich auflisten. Diesen Vorgang nennt man „Variablen deklarieren“. StarBasic verlangt das nicht. Es steht Ihnen frei, Variablen zu verwenden, ohne sie zu deklarieren.

Obwohl es ganz praktisch aussieht, Variablen ohne Deklarierung zu verwenden, so ist es doch fehleranfällig. Wenn Sie sich bei einem Variablennamen verschreiben, entsteht daraus eine neue Variable statt einer Fehlermeldung. Wenn Sie daher wollen, dass Basic nicht deklarierte Variablen als Laufzeitfehler behandelt, dann stellen Sie die Schlüsselwörter „Option Explicit“ ganz an den Anfang, vor den ausführbaren Code. Vor Option Explicit dürfen allenfalls noch Kommentare stehen, weil sie nicht ausführbar sind. Es wäre sicher besser, wenn Basic solche Fehler zur Kompilierungszeit fände, tatsächlich aber werden alle Variablen und Routinen erst zur Laufzeit aufgelöst.

Listing 6. *Option Explicit vor der ersten ausführbaren Codezeile eines Makros.*

```
REM  *****  BASIC  *****
Option Explicit
```

Tipp Verwenden Sie „Option Explicit“ ganz am Anfang eines jeden Moduls, das Sie schreiben. Sie werden damit viel Zeit bei der Fehlersuche in Ihrem Code sparen. Wenn ich gebeten werde, ein Makro zu debuggen, füge ich zuallererst „Option Explicit“ an den Anfang jedes Moduls.

Der Rahmen für „Option“-Anweisungen ist das Modul, in dem sie stehen. Anders gesagt, wenn Sie eine Option in einem Modul setzen und dann ein anderes Modul aufrufen, werden nicht deklarierte Variablen im aufgerufenen Modul keinen Laufzeitfehler produzieren, es sei denn, auch in diesem Modul stünde „Option Explicit“.

Sie können eine Variable mit oder ohne Typ deklarieren. Eine Variable ohne explizite Typangabe wird als Typ Variant geführt, der Daten eines jeden Typs enthalten kann. Das heißt, dass Sie Variant verwenden können, um zum Beispiel einen numerischen Wert aufzunehmen und ihn dann in der nächsten Codezeile zum Beispiel mit einer Zeichenfolge zu überschreiben. Tabelle 5 zeigt die von Basic unterstützten Variablentypen, die Werte, die sie unmittelbar nach der Deklarierung enthalten („Anfangswert“), und die jeweils verwendete Anzahl an Bytes. In Basic kann der Typ einer Variablen auch dadurch festgelegt werden, dass ein spezielles Zeichen bei der Deklarierung an das Namensende gesetzt wird. Die Spalte Annex in der Tabelle 5 enthält die vorgesehenen Zeichen, die an einen Variablennamen bei der Deklarierung angehängt werden können.

Tabelle 5. *Verfügbare Variablentypen und ihre Anfangswerte.*

Typ	Annex	Anfangswert	Bytes	Konvertierung	Beschreibung
Boolean		False	1	CBool	True oder False
Currency	@	0.0000	8	CCur	Währung mit 4 Dezimalstellen
Date		00:00:00	8	CDate	Datum und Uhrzeit
Double	#	0.0	8	CDbl	Dezimalzahl im Bereich von +/-1,79769313486232 x 10E308
Integer	%	0	2	CInt	Ganzzahl von -32.768 bis 32.767
Long	&	0	4	CLng	Ganzzahl von -2.147.483.648 bis 2.147.483.647
Object		Null	wechselt		Objekt
Single	!	0.0	4	CSng	Dezimalzahl im Bereich von +/-3,402823 x 10E38
String	$	""	wechselt	CStr	Text mit bis zu fast 2 GB Zeichen
Variant		Empty	wechselt	CVar	Kann Daten jedes Typs enthalten

Tipp Intern wird für die Länge eines Strings eine 32-Bit-Ganzzahl verwendet. Daraus entnehme ich, dass ein String 2,147,483,647 (2GB - 1) Zeichen enthalten kann.

Obwohl Basic den Variablentyp Byte unterstützt, können Sie ihn nicht direkt deklarieren und verwenden. Wie später erläutert, gibt die Funktion CByte einen Byte-Wert zurück, der einer Variablen vom Typ Variant zugewiesen werden kann. Seit OOo 2.0 können Sie eine Variable mit Typ Byte deklarieren. Diese Variable wird aber als ein extern definiertes Objekt vom Typ Byte behandelt und nicht als eine intern definierte Byte-Variable.

Mit dem Schlüsselwort DIM deklarieren Sie eine Variable vor dem ersten Gebrauch (s. Tabelle 6). Sie können mehrere Variablen in einer einzigen Zeile deklarieren, und Sie können jeder Variablen beim Deklarieren einen Typ zuweisen. Variablen ohne deklarierten Typ sind automatisch vom Typ Variant.

Tabelle 6. *Deklarierung einfacher Variablen.*

Deklarierung	Beschreibung
Dim Name	Name hat den Typ Variant, weil kein Typ festgelegt ist.
Dim Name As String	Name hat den Typ String, weil der Typ explizit festgelegt ist.
Dim Name$	Name$ hat den Typ String, weil Name$ mit einem $ endet.

Deklarierung	Beschreibung
Dim Name As String, Weight As Single	Name hat den Typ String, und Weight hat den Typ Single.
Dim Width, Length	Width und Length haben den Typ Variant.
Dim Weight, Height As Single	Weight hat den Typ Variant, und Height hat den Typ Single.

Tipp Wenn mehrere Variablen in einer Zeile deklariert werden, muss für jede Variable der Typ gesondert angegeben werden. In der letzten Zeile der Tabelle 6 ist Weight vom Typ Variant, auch wenn es so aussehen könnte, als ob es vom Typ Single wäre.

In einem Großteil der Literatur über OOo-Makroprogrammierung wird ein Namensschema für Variablen verwendet, das auf der Ungarischen Notation beruht. Dabei können Sie vom Namen auf den Typ einer Variablen schließen. Praktisch macht das jeder so oder anders und hält sich mehr oder weniger fest daran. Das ist eine Frage des Stils. Manche lieben es, und manche hassen es.

Basic hat eine Reihe von Anweisungen zur Förderung der Ungarischen Notation: Def<Typ>. Die Def-Anweisungen sind lokal auf das jeweilige Modul begrenzt, in dem sie stehen. Sie weisen nicht-deklarierten Variablen abhängig von ihren Namen einen bestimmten Typ zu. Normalerweise sind alle nicht-deklarierten Variablen vom Typ Variant.

Der Def-Anweisung folgt eine durch Kommas getrennte Liste von Zeichenbereichen für den ersten Buchstaben der Namen (s. Listing 7).

Listing 7. *Deklarierung von typlosen Variablen, die mit i, j, k oder n beginnen, als Typ Integer.*

```
DefInt i-k,n
```

Tabelle 7 enthält je ein Beispiel für die verfügbaren Def-Anweisungen. Wie Option-Anweisungen werden auch Def-Anweisungen vor alle anderen ausführbaren Zeilen oder Variablendeklarierungen eines Moduls platziert. Die Def-Anweisung erzwingt nicht für jede Variable, die mit einem bestimmten Buchstaben beginnt, einen spezifischen Typ, sondern stellt einen Ersatztyp statt Variant für Variablen zur Verfügung, die verwendet, aber nicht deklariert werden. Ich habe bisher die Def-Anweisungen noch nie in Gebrauch gesehen und kann sie auch nicht empfehlen.

Tipp Wenn Sie „Option Explicit“ nutzen, und das sollten Sie auch, müssen Sie alle Variablen deklarieren. Das macht die Def<Typ>-Anweisungen nutzlos, weil sie nur auf nicht deklarierte Variablen wirken.

Tabelle 7. *Beispiele für die verfügbaren Def-Anweisungen in OOo.*

Def-Anweisung	Typ
DefBool b	Boolean
DefDate t	Date
DebDbl d	Double
DefInt i	Integer
DefLng l	Long
DefObj o	Object
DefVar v	Variant

3.3.3. Variablen einen Wert zuweisen

Der Zweck einer Variablen ist, einen Wert aufzunehmen. Um einer Variablen einen Wert zuzuweisen, schreiben Sie den Namen der Variablen, wahlweise Leerzeichen, ein Gleichheitszeichen, wahlweise Leerzeichen und den Wert, den die Variable erhalten soll. Etwa so:

```
X = 3.141592654
y = 6
```

Das optionale Schlüsselwort Let darf dem Variablennamen vorangehen, dient aber ausschließlich der Lesbarkeit. Das ähnliche Schlüsselwort Set, für Objektvariablen, dient auch nur der Lesbarkeit. Der Gebrauch dieser Schlüsselwörter ist sehr selten.

3.3.4. Boolesche Variablen sind entweder True oder False

Boolesche Variablen zeigen nur zwei Zustände: wahr oder falsch, als Wertrepräsentation True oder False. Intern sind es die Werte -1 beziehungsweise 0. Jeder einer booleschen Variablen zugewiesene numerische Wert, der nicht genau 0 ergibt, wird zu True konvertiert. Das Makro in Listing 8 führt einige neue Konzepte ein. Eine String-Variable, s, akkumuliert die Berechnungsergebnisse, die dann in einem Dialog angezeigt werden (s. Bild 18). Um einen Zeilenumbruch im Dialog zu erzeugen, wird Chr$(10) hinzugefügt. Die Rechenergebnisse in einer Zeichenfolge zusammenzufügen, braucht leider ein etwas kompliziertes Makro als einfache Anweisungen wie „`Print CBool(5=3)`“, aber die Resultate sind leichter zu verstehen (s. Bild 18). In der odt-Version dieses Dokuments finden Sie häufig eine Schaltfläche, über die Sie das Makro direkt ausführen können.

Listing 8. *Demonstration der Konvertierung zum Typ Boolean.*

```
Sub ExampleBooleanType
  Dim b As Boolean
  Dim s As String
  b = True

  b = (5 = 3) REM Ergibt False
  s = "(5 = 3) => " & b
  b = (5 < 7) REM Ergibt True
  s = s & Chr$(10) & "(5 < 7) => " & b
  b = 7       REM Ergibt True, weil 7 nicht 0 ist.
  s = s & Chr$(10) & "(7) => " & b
  MsgBox s
End Sub
```

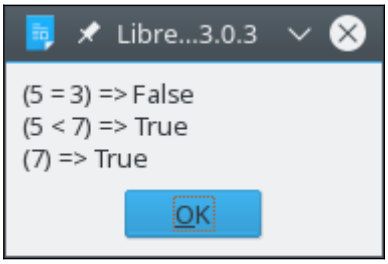


Bild 18. *Der von Listing 8 angezeigte Dialog.*

Intern wird True als -1 binär so dargestellt, dass alle Bits auf 1 gesetzt sind. Bei der binären Darstellung von False als 0 sind alle Bits auf 0.

3.3.5. Numerische Variablen

Numerische Variablen enthalten Zahlen. Basic unterstützt Ganzzahlen, Fließkommazahlen und Währungszahlen. Ganzzahlen können hexadezimal (Basis 16), oktal (Basis 8) oder in der Standardform dezimal (Basis 10) repräsentiert werden.

Eine Diskussion der anderen Zahlensysteme ist deswegen wichtig, weil Rechner ihre Daten intern binär speichern. Es ist einfach, zwischen binär, hexadezimal und oktal zu konvertieren, und für menschliche Wesen mag es einfacher sein, Binärzahlen visuell in anderen Zahlensystemen zu erkennen.

Dezimalzahlen bestehen aus den 10 Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8 und 9. Addieren Sie dezimal 1 zu 9 und Sie erhalten 10 (Basis 10).

Im Computerbereich werden häufig auch Zahlen in binärer (Basis 2), oktaler (Basis 8) und hexadezimaler (Basis 16) Darstellung verwendet.

Oktalzahlen bestehen aus den 8 Ziffern 0, 1, 2, 3, 4, 5, 6 und 7. Addieren Sie oktal 1 zu 7 und Sie erhalten 10 (Basis 8).

Hexadezimalzahlen bestehen aus den 16 Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E und F. Addieren Sie hexadezimal 1 zu F und Sie erhalten 10 (Basis 16).

Binärzahlen bestehen aus den beiden Ziffern 0 und 1. Addieren Sie binär 1 zu 1 und Sie erhalten 10 (Basis 2).

Tabelle 8 enthält die Zahlen von 0 bis 18 in dezimaler Form, mit ihren binären, oktalen und hexadezimalen Entsprechungen.

Tabelle 8. *Zahlen in verschiedenen Darstellungssystemen.*

Dezimal	Binär	Oktal	Hexadezimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12

Es wird vorausgesetzt, dass Ganzzahlen in dezimaler Form erscheinen. Kommas sind nicht erlaubt. Hexadezimalzahlen haben das Präfix „&H“ und Oktalzahlen das Präfix „&O“ (Buchstabe O, keine Null). Leider existiert kein einfacher Weg, Binärzahlen einzugeben. Tabelle 9 bietet einige einfache Richtlinien zur Zahleneingabe.

Tabelle 9. *Einige einfache Richtlinien zur Zahleneingabe in Basic.*

Beispiel	Beschreibung
Schreiben Sie 1000, nicht 1.000	Schreiben Sie Zahlen ohne Tausenderkomma, verwenden Sie überhaupt keine Kommas.
+ 1000	Zwischen einem Plus- oder Minuszeichen und der Zahl ist ein Leerzeichen erlaubt.

Beispiel	Beschreibung
&HFE ist dasselbe wie 254	Beginnen Sie eine Hexadezimalzahl mit &H.
&O11 ist dasselbe wie 9	Beginnen Sie eine Oktalzahl mit &O.
Schreiben Sie 3.1415, nicht 3,1415	Verwenden Sie einen Punkt als Dezimaltrenner, kein Komma.
6.022E23	In wissenschaftlicher Darstellung kann das „e“ groß oder klein geschrieben sein.
Schreiben Sie 6.6e-34, nicht 6.6e -34	Leerzeichen innerhalb von Zahlen sind nicht erlaubt. Mit dem Leerzeichen wird 6.6 - 34 = -27.4 errechnet.
6.022e+23	Vor dem Exponenten kann ein Plus- oder Minuszeichen stehen.
1.1e2.2 ergibt 1.1e2	Der Exponent muss eine Ganzzahl sein. Ein Bruchteil wird ignoriert.

Wenn man einer numerischen Variablen eine Zeichenfolge zuweist, wird die Variable im Allgemeinen auf null gesetzt. Es wird kein Fehler generiert. Wenn die ersten Zeichen des Strings aber eine Zahl ergeben, wird der String zu dieser Zahl konvertiert, und der nicht numerische Rest des Strings wird ignoriert – mit der Möglichkeit eines numerischen Überlaufs.

Typ Integer

Eine Integer-Zahl ist eine Ganzzahl, die positiv, negativ oder gleich null sein kann. Integer-Variablen sind eine gute Wahl für Zahlen ohne Bruchanteile, wie Lebensalter oder Kinderzahl. In Basic sind Integer-Variablen 16-Bit-Zahlen im Bereich von -32.768 bis 32.767. Bei der Zuweisung einer Fließkommazahl zu einer Integer-Variablen wird zur nächsten Ganzzahl gerundet. Einem Variablennamen ein „%“ anzuhängen ist die Kurzform der Deklarierung als Typ Integer.

Listing 9. *Demonstration von Integer-Variablen.*

```
Sub ExampleIntegerType
  Dim i1 As Integer, i2%  REM i1 und i2 sind beide Integer-Variablen
  Dim f2 As Double
  Dim s$     REM String-Variable
  f2= 3.5
  i1= f2     REM i1 wird gerundet zu 4

  s = "3.50 => " & i1
  f2= 3.49
  i2= f2     REM i2 wird gerundet zu 3
  s = s & Chr$(10) & "3.49 => " & i2
  MsgBox s
End Sub
```

Bild 19. *Demonstration von Integer-Variablen im Listing 9.*

Typ Long Integer

„Long“ ist eine Ganzzahl mit einem größeren Bereich als der Typ Integer. Long-Variablen sind 32-Bit-Zahlen im Bereich von -2.147.483.648 bis 2.147.483.647. Long-Variablen benötigen daher zweimal so viel Speicherplatz wie Integer-Variablen, aber sie können Zahlen aufnehmen, die um ein *Viel-*

faches größer sind. Bei der Zuweisung einer Fließkommazahl zu einer Long-Variablen wird zum nächsten Long-Ganzzahlwert gerundet. Einem Variablennamen ein „&“ anzuhängen ist die Kurzform der Deklarierung als Typ Long. Die Ausgabe von Listing 10 ist dieselbe wie von Listing 9 (s. Bild 19).

Listing 10. *Demonstration von Long-Variablen.*

```
Sub ExampleLongType
  Dim NumberOfDogs&, NumberOfCats As Long
                    'Anzahl der Hunde und Katzen. Beide Variablen sind Long.
  Dim f2 As Double
  Dim s$    REM String-Variable
  f2= 3.5
  NumberOfDogs = f2   REM Gerundet zu 4
  s = "3.50 => " & NumberOfDogs
  f2= 3.49
  NumberOfCats = f2   REM Gerundet zu 3
  s = s & Chr$(10) & "3.49 => " & NumberOfCats
  MsgBox s
End Sub
```

Typ Currency

Variablen vom Typ Currency sind, wie der Name schon sagt, für Finanzwerte gedacht. Der Typ Currency wurde ehemals eingeführt, um Rundungsfehler der Fließkommatypen Single und Double zu vermeiden. Visual Basic .NET ersetzte den Typ Currency durch den Typ Decimal.

Currency-Variablen sind 64-Bit-Festkommazahlen, die mit vier Dezimalstellen und 15 Stellen vor dem Komma genau berechnet werden. Der Bereich erstreckt sich von -922.337.203.658.477,5808 bis +922.337.203.658.477,5807. Einem Variablennamen ein „@“ anzuhängen ist die Kurzform der Deklarierung als Typ Currency.

Listing 11. *Demonstration von Currency-Variablen.*

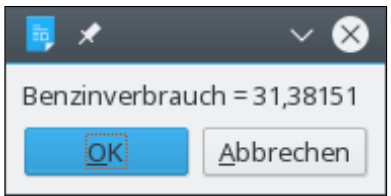
```
Sub ExampleCurrencyType
  Dim Income@, CostPerDog As Currency  'Einkommen und Hundehaltungskosten
  Income@ = 22134.37
  CostPerDog = 100.0 / 3.0
  REM Gibt 22134.3700 aus
  Print "Einkommen = " & Income@
  REM Gibt 33.3333 aus
  Print "Kosten pro Hund = " & CostPerDog
End Sub
```

Typ Single

Variablen vom Typ Single können im Gegensatz zu denen vom Typ Integer einen Bruchanteil haben. Sie werden „Fließkommazahlen“ genannt, weil im Gegensatz zum Typ Currency die erlaubte Dezimalstellenanzahl nicht festgelegt ist. Single-Variablen sind 32-Bit-Zahlen mit einer Genauigkeit von etwa sieben angezeigten Ziffern, was für mathematische Operationen mittlerer Genauigkeit ausreicht. Sie umfassen positive und negative Werte von 3,402823 x 10E38 bis 1,401298 x 10E-45. Jede Zahl, die kleiner ist als 1,401298 x 10E-45, wird zu null. Einem Variablennamen ein „!“ anzuhängen ist die Kurzform der Deklarierung als Typ Single.

Listing 12. *Demonstration von Single-Variablen.*

```
Sub ExampleSingleType
  Dim GallonsUsed As Single, Miles As Single, mpg!
  REM Verbrauch an Gallonen, gefahrene Meilen
  REM Amerikanische Methode für den durchschnittlichen
  REM Benzinverbrauch: Meilen pro Gallone
```



```
  GallonsUsed = 17.3
  Miles = 542.9
  mpg! = Miles / GallonsUsed
  Print "Benzinverbrauch = " & mpg!
End Sub
```

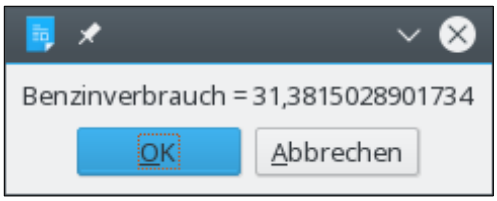
Typ Double

Variablen vom Typ Double ähneln denen vom Typ Single, außer dass sie 64 Bit lang sind und auf etwa 15 Ziffern genau sind. Sie eignen sich für mathematische Operationen hoher Genauigkeit. Sie umfassen positive und negative Werte von 1,79769313486232 x 10E308 bis 4,94065645841247 x 10E-324. Jede Zahl, die kleiner ist als 4,94065645841247 x 10E-324, wird zu null. Einem Variablennamen ein „#“ anzuhängen ist die Kurzform der Deklarierung als Typ Double.

Listing 13. *Demonstration von Double-Variablen.*

```
Sub ExampleDoubleType
  Dim GallonsUsed As Double, Miles As Double, mpg#
  GallonsUsed = 17.3
  Miles = 542.9

  mpg# = Miles / GallonsUsed
  Print " Benzinverbrauch = " & mpg#
End Sub
```



3.3.6. String-Variablen enthalten Text

Variablen vom Typ String sind dazu gedacht, Text zu enthalten. In OOo wird Text im Standard von Unicode 2.0 gespeichert, wodurch eine Vielzahl von Sprachen unterstützt wird. Jede String-Variable kann bis zu 65.535 Zeichen aufnehmen. Einem Variablennamen ein „$“ anzuhängen ist die Kurzform der Deklarierung als Typ String.

Listing 14. *Demonstration von String-Variablen.*

```
Sub ExampleStringType
  Dim FirstName As String, LastName$  'Vorname, Nachname
  FirstName = "Andrew"
  LastName$ = "Pitonyak"
  Print "Hallo " & FirstName & " " & LastName$
End Sub
```

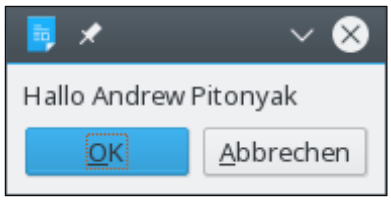


Bild 20. *Demonstration von String-Variablen in Listing 14.*

In Visual Basic .NET können String-Variablen annähernd 2 Milliarden Unicode-Zeichen enthalten. Früher waren OOo-String-Variablen auf 65.535 Zeichen begrenzt. Diese Grenze ist jedoch aufgehoben. Ich kenne die aktuelle Höchstgrenze nicht wirklich, habe aber Tests mit bis zu 10 Millionen Zeichen erfolgreich durchgeführt.

Um ein Anführungszeichen in einen String einzufügen, setzen Sie es zweimal hintereinander.

```
S = "Sie sagte ""Hallo"""  REM Sie sagte "Hallo"
```

Mit „Option Compatible“ machen Sie Stringkonstanten von Visual Basic verfügbar (s. Tabelle 10). Sie müssen modulweit Option Compatible verwenden statt CompatibilityMode(True), denn Stringkonstanten werden zur Kompilierungszeit aufgelöst und nicht zur Laufzeit.

Tabelle 10. *Zu Visual Basic kompatible Stringkonstanten.*

Konstante	Wert	Beschreibung
vbCr	Chr$(13)	Zeilenrücklauf
vbCrLf	Chr$(13) & Chr$(10)	Kombination Zeilenrücklauf/Zeilenvorschub
vbFormFeed	Chr$(12)	Seitenvorschub
vbLf	Chr$(10)	Zeilenvorschub
vbNewLine	Chr$(13) & Chr$(10) oder Chr$(10)	Neue Zeile, betriebssystemabhängig – was gerade angebracht ist
vbNullChar	Chr$(0)	Zeichen mit dem ASCII-Wert 0
vbNullString	""	Leerer String. Ein String mit dem ASCII-Wert 0 am Ende.
vbTab	Chr$(9)	Waagerechter Tabulatorschritt
vbVerticalTab	Chr$(11)	Senkrechter Tabulatorschritt

Die Stringkonstanten in Tabelle 10 erlauben Ihnen, Strings mit Sonderzeichen zu definieren. Früher mussten Sie dafür die Funktion Chr$() aufrufen. Außerdem erlaubt Ihnen Option Compatible, Konstanten als Typ zu definieren.

```
Option Compatible
Const sGreeting As String = "Hallo" & vbCr & "Johnny"  ' Inklusive Zeilenrücklauf (CR).
```

3.3.7. Date-Variablen

Variablen vom Typ Date enthalten Datums- und Uhrzeitwerte. Basic speichert Date intern als Double. Date-Variablen werden wie alle numerischen Typen zu null initialisiert. Das entspricht dem 30. Dezember 1899, 00:00:00 Uhr. Zu einem Date 1 zu addieren oder 1 von ihm zu subtrahieren entsprechen der Addition oder Subtraktion eines Tages. Eine Stunde, eine Minute und eine Sekunde entsprechen den Zahlen 1/24, 1/(24 * 60) und 1/(24 * 60 * 60). Die Basic-Datumsfunktionen werden in Tabelle 11 vorgestellt und an späterer Stelle im einzelnen erörtert.

Listing 15. *Demonstration von Date-Variablen.*

```
Sub ExampleDateType
  Dim tNow As Date, tToday As Date  'Jetzt, Heute
  Dim tBirthDay As Date             'Geburtstag
  tNow = Now()
  tToday = Date()
  tBirthDay = DateSerial(1776, 7, 4)
  Print "Heute = " & tToday
  Print "Jetzt = " & tNow
  Print "Insgesamt sind " & (tToday - tBirthDay) &_
    " Tage vergangen seit " & tBirthDay
End Sub
```

Negative Zahlen sind erlaubt und beziehen sich auf Datumswerte vor dem 30. Dezember 1899. Der 1. Januar 0001 wird also als Fließkommazahl -693.595 dargestellt. Weiter zurückgehend kommt man zu Datumswerten vor Christi Geburt (englische Abkürzung B.C = Before Christ) im Gegensatz zur heutigen Zeitzählung A.D. (Anno Domini). An späterer Stelle wird eingehend auf die Datumsbehandlung eingegangen.

Tabelle 11. *Funktionen und Subroutinen mit Bezug auf Datum und Uhrzeit.*

Funktion	Typ	Beschreibung
CDate(Ausdruck)	Date	Konvertiert Zeichenfolgen oder numerische Ausdrücke in Datumswerte.
CDateFromIso(String)	Date	Gibt aus einer Zeichenkette mit einem Datum im ISO-Format eine Datumszahl im internen Format zurück.
CDateToIso(Date)	String	Gibt aus einer Datumszahl das Datum im ISO-Format zurück.
Date()	Date	Gibt das aktuelle Systemdatum als Date zurück.
DateSerial(Jahr, Monat, Tag)	Date	Erzeugt einen Datumswert für eine Datumsangabe aus den numerischen Werten Jahr, Monat und Tag.
DateValue(Date)	Date	Extrahiert das Datum aus einem Datum/Uhrzeit-Wert. Trennt dazu den Dezimalteil ab.
Day(Date)	Integer	Gibt aus einem Datumswert den Monatstag als Integer zurück.
GetSystemTicks()	Long	Gibt die vom Betriebssystem angegebene Anzahl von Systemzeit-Perioden (Systemticks) als Long zurück.
Hour(Date)	Integer	Gibt aus einem Datumswert die Stunde als Integer zurück.
IsDate(Wert)	Boolean	Ist dies ein Datumswert?
Minute(Date)	Integer	Gibt aus einem Datumswert die Minute als Integer zurück.
Month(Date)	Integer	Gibt aus einem Datumswert den Monat als Integer zurück.
Now()	Date	Gibt das aktuelle Systemdatum und die aktuelle Systemzeit als Date zurück.
Second(Date)	Integer	Gibt aus einem Datumswert die Sekunde als Integer zurück.
Time()	String	Gibt die aktuelle Systemzeit als Zeichenfolge zurück.
Timer()	Date bzw. Long	Gibt die seit Mitternacht vergangene Zeit in Sekunden zurück, aber nur, wenn die Funktion in eine Long-Variable geschrieben wird. Ansonsten wird ein Wert vom Typ Date zurückgegeben.
TimeSerial(Stunde, Minute, Sekunde)	Date	Erzeugt einen Datumswert für eine Uhrzeitangabe aus den numerischen Werten Stunde, Minute und Sekunde.
WeekDay(Date)	Integer	Gibt aus einem Datumswert eine Zahl zwischen 1 und 7 zurück, entsprechend den Wochentagen Sonntag bis Samstag.
Year(Date)	Integer	Gibt aus einem Datumswert das Jahr als Integer zurück.

3.3.8. Eigene Datentypen erzeugen

In den meisten Ausprägungen der Programmiersprache BASIC können Sie Ihre eigenen Datentypen erzeugen. Auch StarBasic erlaubt die Verwendung selbst definierter Datentypen.

Sie starten einen eigenen Datentyp mit dem Schlüsselwort Type und anschließendem selbst gewählten Namen. Die Regeln zur Namensgebung sind dieselben wie bei Variablen. End Type bestimmt das Ende. Dazwischen definieren Sie zeilenweise die enthaltenen Namen mit ihren Datentypen.

Listing 16. *Demonstration benutzerdefinierter Typen.*

```
Type PersonType
  FirstName As String  'Vorname
  LastName As String   'Nachname
End Type

Sub ExampleCreateNewType
  Dim Person As PersonType
  Person.FirstName = "Andrew"
  Person.LastName  = "Pitonyak"
```

```
  PrintPerson(Person)
End Sub

Sub PrintPerson(x)
  Print "Person = " & x.FirstName & " " & x.LastName
End Sub
```

Tipp Obwohl benutzerdefinierte Typen nicht direkt ein Array enthalten können, so kann man es aber über den Typ Variant einrichten.

Es gibt drei Wege, eine Instanz eines benutzerdefinierten Typs zu deklarieren. Im folgenden Beispiel wird der Doppelpunkt verwendet, um zwei Anweisungen in einer Zeile zu platzieren.

```
Dim x As New PersonType                 ' Der ursprüngliche Weg.
Dim y As PersonType                     ' New ist nun nicht mehr erforderlich.
Dim z : z = CreateObject("PersonType")  ' Das Objekt wird erstellt, wenn es nötig ist.
```

Wenn Sie Ihren eigenen Typ erstellen, verwenden Sie eine Struktur (häufig auch Struct genannt). OOo hat viele vordefinierte interne Strukturen. Eine häufig verwendete Struktur ist „com.sun.star.beans.PropertyValue“. Die internen OOo-Strukturen können genauso wie benutzerdefinierte Typen erstellt werden, aber auch mit der Funktion CreateUnoStruct (s. Kapitel 10. Universal Network Objects (UNO).

```
Dim a As New com.sun.star.beans.PropertyValue
Dim b As New com.sun.star.beans.PropertyValue
Dim c : c = CreateObject("com.sun.star.beans.PropertyValue")
Dim d : d = CreateUnoStruct("com.sun.star.beans.PropertyValue")
```

Obwohl der Typ der Struktur „com.sun.star.beans.PropertyValue“ ist, ist es üblich, in Beschreibungen (nicht in den Deklarierungen) den Typnamen zum letzten Namensteil abzukürzen – in diesem Fall zu „PropertyValue“. Viele Objekte in OOo haben ähnlich lange sperrige Namen, die in diesem Buch auf ähnliche Weise abgekürzt werden.

Die meisten Variablen werden mit ihrem Wert kopiert. Das bedeutet, dass wenn ich eine Variable einer anderen zuweise, der Wert der einen in die andere kopiert wird. Sie verweisen nicht auf dieselben Daten, sondern sie enthalten je eine eigene Kopie der Daten. Das gilt auch für benutzerdefinierte Typen und interne OOo-Strukturen. Auf diesem Wege deklarierte Variablen werden mit ihrem Wert kopiert. Andere intern von OOo genutzte Typen, so genannte Universal Network Objects, werden als Referenz kopiert. Obwohl sie erst an späterer Stelle erläutert werden, ist es jetzt schon wichtig, darüber nachzudenken, was geschieht, wenn eine Variable einer anderen zugewiesen wird. Wenn ich eine Variable einer anderen als Referenz zuweise, verweisen beide Variablen auf dieselben Daten. Wenn zwei Variablen auf dieselben Daten verweisen und ich eine ändere, ändere ich beide.

3.3.9. Variablen mit speziellen Typen deklarieren

Sie können mit den Schlüsselwörtern „As New“ eine Variable als bekanntes UNO-Struct definieren. Das Wort „Struct“ ist abgekürzt von dem Wort „Structure“, das häufig von Computerprogrammierern verwendet wird. Ein Struct enthält einen oder mehrere Datenelemente (so genannte Members), die von unterschiedlichem Typ sein können. Structs werden eingesetzt, um zusammengehörende Daten zu gruppieren.

Option Compatible bietet eine neue Syntax zur Definition von bekannten und unbekannten Typen. Ein einfaches Beispiel deklariert eine Variable eines speziellen Typs, auch wenn StarBasic den Typ nicht kennt.

```
Option Compatible          'Unterstützt seit OOo 2.0
Sub Main
  Dim oVar1 As Object
  Dim oVar2 As MyType
  Set oVar1 = New MyType   ' Unterstützt seit OOo 2.0
```

```
  Set oVar2 = New MyType    ' Unterstützt seit OOo 2.0
  Set oVar2 = New YourType ' Fehler, als MyType deklariert, nicht als YourType.
```

Mit OOo 2.0 wurde eine neue OLE-Objekt-Factory geschaffen, mit der neue Typen erstellt werden können. Die neue Funktionalität macht es möglich, mit StarBasic Microsoft-Word-Dokumente unter Microsoft Windows zu bearbeiten, vorausgesetzt Microsoft Office ist auch installiert.

```
Sub Main
  Dim W As Word.Application
  Set W = New Word.Application
  REM Dim W As New Word.Application        'Funktioniert seit OOo 2.0
  REM W = CreateObject("Word.Application") 'Funktioniert seit OOo 2.0
  W.Visible = True
End Sub
```

Zum Einsatz von CreateObject() benötigen Sie kein „Option Compatible“, weil diese Funktionalität von der OLE-Objekt-Factory beginnend mit OOo 2.0 bereitgestellt wird.

3.3.10. Objekt-Variablen

Ein Object ist ein komplexer Datentyp, der mehr als ein einzelnes Datenelement enthalten kann. Der Code in Listing 16 zeigt ein Beispiel eines komplexen Datentyps. Object-Variablen dienen dazu, mit Basic erstellte und definierte komplexe Datentypen aufzunehmen. Bei der Deklarierung einer Variablen vom Typ Object wird sie mit dem Spezialwert Null initialisiert, der signalisiert, dass die Variable momentan keinen gültigen Wert enthält.

Im Gegensatz zu Variant-Variablen (s. unten) können Sie einer Object-Variablen keinen Wert eines einfachen Datentyps wie zum Beispiel String oder Integer zuweisen. Sie erhalten den Laufzeitfehler „Objektvariable nicht belegt“.

In alten OOo-Versionen war es sicherer, den Typ Variant statt Object zu verwenden, wenn Sie auf interne OOo-Objekte verweisen. Das Thema wird im Abschnitt 10.6. Typdefinition Object oder Variant erörtert.

3.3.11. Variant-Variablen

Variablen vom Typ Variant können jeden Datentyp aufnehmen. Welche Daten auch immer ihnen zugewiesen werden, sie übernehmen deren Typ. Bei der Deklarierung einer Variant-Variablen wird sie zu dem Spezialwert Empty initialisiert, der signalisiert, dass die Variable momentan keinen Wert enthält. Eine Variant-Variable kann in der einen Anweisung einen Integer-Wert erhalten und in der nächsten einen Text. Wird einer Variant-Variablen ein Wert zugewiesen, wird auch nicht automatisch konvertiert. Sie erhält einfach nur den passenden Typ.

Wegen des chamäleonähnlichen Verhaltens der Variant-Variablen kann man sie wie jeden anderen Variablentyp verwenden. Diese Flexibilität hat aber seinen Preis: Zeit. Und schließlich hat man noch das Problem, dass nach einigen Zuweisungen nicht immer deutlich ist, welchen Typ eine Variant-Variable gerade repräsentiert.

Listing 17. *Demonstration von Variant-Variablen.*

```
Sub ExampleTestVariants
  Dim s As String
  Dim v As Variant
  REM v ist beim Start leer
  s = s & "1   : TypeName = " & TypeName(v) & " Wert = " & v & Chr$(10)
  v = "ab217" : REM v wird zu String
  s = s & "2   : TypeName = " & TypeName(v) & " Wert = " & v & Chr$(10)
  v = True    : REM v wird zu Boolean
```

```
    s = s & "3   : TypeName = " & TypeName(v) & " Wert = " & v & Chr$(10)
    v = (5=5)    : REM v wird zu Integer statt zu Boolean
    s = s & "4   : TypeName = " & TypeName(v) & " Wert = " & v & Chr$(10)
    v = 123.456 : REM Double
    s = s & "5   : TypeName = " & TypeName(v) & " Wert = " & v & Chr$(10)
    v =123       : REM Integer
    s = s & "6   : TypeName = " & TypeName(v) & " Wert = " & v & Chr$(10)
    v = 1217568942 : REM Es könnte Long sein, tatsächlich aber wird es Double
    s = s & "7   : TypeName = " & TypeName(v) & " Wert = " & v & Chr$(10)
    MsgBox s, 0, "Variant nimmt viele Typen an"
End Sub
```

Visual Basic .NET unterstützt den Typ Variant nicht. Nicht typisierte Variablen sind vom Typ Object.

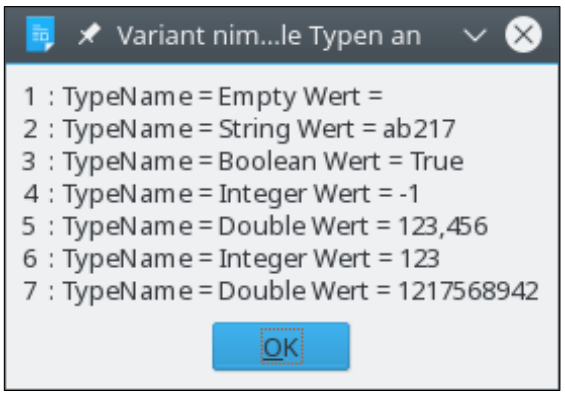


Bild 21. *Variant übernimmt den zugewiesenen Typ.*

Bei der Wertzuweisung in eine Variant-Variable wird der Wert nicht zu einem passenden Typ konvertiert. Stattdessen wird Variant zu dem Typ des Wertes. In Zeile 6 im Bild 21 ist Variant ein Integer. In Zeile 7 ist die Zahl zu groß für Integer, aber klein genug für Long. Basic zieht es aber vor, alle Zahlen, die größer als Integer sind und alle Fließkommazahlen nach Double zu konvertieren, auch wenn sie als Single oder Long kodiert werden könnten.

3.3.12. Konstanten

Eine Konstante ist eine Variable ohne Typ, deren Wert nicht geändert werden kann. Die Variable ist als Platzhalter bestimmt, der durch den Ausdruck ersetzt wird, mit dem sie definiert wird. Konstanten werden mit dem Schlüsselwort Const definiert. Die Regeln für Konstantennamen sind dieselben wie für gültige Variablennamen.

```
Const ConstName=Expression
```

Konstanten bereichern Makros auf manche Weise. Nehmen Sie einmal eine Schwerkraftkonstante, wie sie oft in der Physik benötigt wird. Physiker werden sie als Erdbeschleunigung in Metern pro Sekundenquadrat erkennen.

```
Const Gravity = 9.81
```

Ein paar konkrete Vorteile bei der Verwendung von Konstanten:

- Konstanten verbessern die Lesbarkeit eines Makros. Das Wort Gravity (Schwerkraft) ist leichter zu erkennen als der Wert 9,81.
- Konstanten sind leicht zu pflegen. Wenn ich eine größere Genauigkeit brauche oder wenn sich die Anziehungskraft ändert, muss ich den Wert nur an einer einzigen Stelle ändern.
- Konstanten helfen, schwer zu findende Fehler dadurch zu vermeiden, dass Laufzeitfehler in Fehler zur Kompilierungszeit umgewandelt werden. Wenn Sie „Grevity“ schreiben statt „Gravity“, ist das ein Kompilierungsfehler, der Schreibfehler „9.18“ statt „9.81“ aber nicht.

- Ein Wert wie 9,81 mag für Sie unverwechselbar sein, für andere aber möglicherweise nicht, die Ihren Code später einmal lesen. Die Zahl wird zu dem, was Programmierer eine „magische Zahl“ nennen, und erfahrene Programmierer versuchen, solche magischen Zahlen unter allen Umständen zu vermeiden. Durch den Umstand, dass sie nicht erläutert sind, ergeben sich später Schwierigkeiten bei der Programmpflege, wenn nämlich der Programmautor nicht für eine Erklärung verfügbar ist – oder die Einzelheiten komplett vergessen hat.

Tipp OOo definiert die Konstante Pi, eine mathematische Konstante mit einem Wert von angenähert 3,1415926535897932385.

3.4. Die Anweisung With

Die Anweisung With vereinfacht den Zugriff auf komplexe Datentypen. Listing 16 definiert einen Datentyp, der zwei verschiedene Elemente enthält: FirstName und LastName. Sie greifen auf diese Elemente zu, indem Sie einen Punkt zwischen Variablennamen und Datenelement setzen.

```
Sub ExampleCreateNewType
  Dim Person As PersonType
  Person.FirstName = "Andrew"
  Person.LastName  = "Pitonyak"
End Sub
```

With bietet eine Kurzschreibweise des Zugriffs auf mehrere Datenelemente derselben Variablen.

```
Sub ExampleCreateNewType
  Dim Person As PersonType
  With Person
    .FirstName = "Andrew"
    .LastName  = "Pitonyak"
  End With
End Sub
```

Vergleichbar:

```
Dim oProp As New com.sun.star.beans.PropertyValue
oProp.Name  = "Person"      'Setzt die Eigenschaft Name
oProp.Value = "Boy Bill"    'Setzt die Eigenschaft Value
```

Mit With:

```
Dim oProp As New com.sun.star.beans.PropertyValue
With oProp
  .Name  = "Person"      'Setzt die Eigenschaft Name
  .Value = "Boy Bill"    'Setzt die Eigenschaft Value
End With
```

3.5. Arrays

Ein Array ist eine Datenstruktur, in der gleichartige Datenelemente in einer indexierten Listenstruktur angeordnet sind – zum Beispiel eine Spalte mit Namen oder eine Tabelle mit Zahlen, s. Tabelle 12. Ein Array bietet Ihnen die Möglichkeit, viele verschiedene Werte in einer einzigen Variablen zu speichern. Zur Definition der Array-Elemente und zum Zugriff auf die Elemente werden runde Klammern verwendet. Basic kennt im Gegensatz zu anderen Sprachen wie zum Beispiel C oder Java keine eckigen Klammern.

Array-Variablen werden mit der Anweisung Dim deklariert. Stellen Sie sich ein eindimensionales Array als eine Wertespalte vor und ein zweidimensionales Array als eine Wertetabelle. Das System erlaubt auch höher dimensionierte Arrays, die man sich aber nur schwer bildhaft machen kann. Der

Index eines Arrays ist nicht mehr auf den Wertebereich des Typs Integer von -32.768 bis 32.767 beschränkt.

Tabelle 12. *Einfache Beispiele, ein Array zu deklarieren.*

Definition	Elemente	Beschreibung
`Dim a(5) As Integer`	6	Von 0 bis 5 inklusive.
`Dim b(5 To 10) As String`	6	Von 5 bis 10 inklusive.
`Dim c(-5 To 5) As String`	11	Von -5 bis 5 inklusive.
`Dim d(5, 1 To 6) As Integer`	36	Sechs Reihen von 0 bis 5 mit sechs Spalten von 1 bis 6.
`Dim e(5 To 10, 20 To 25) As Long`	36	Sechs Reihen von 5 bis 10 mit sechs Spalten von 20 bis 25.

Tipp Bevor Sie Array-Variablen verwenden, müssen Sie sie deklarieren, auch wenn Sie „Option Explicit“ nicht verwenden.

Wenn das untere Bereichsende eines Arrays nicht angegeben ist, wird der Standardwert null gesetzt – im Sprachgebrauch der Programmierer „nullbasiert“. Somit hat ein Array mit fünf Elementen die Indexnummerierung von a(0) bis a(4). Wenn Sie den Standard der unteren Arraybereichsgrenze auf 1 statt 0 ändern wollen, setzen Sie die Schlüsselwörter „Option Base 1“ vor alle anderen ausführbaren Anweisungen im Programm.

```
Option Base { 0 | 1 }
```

Tipp Geben Sie besser die untere Bereichsgrenze eines Arrays an, als auf das Standardverhalten zu setzen. Das ist übertragbar und ändert sich auch dann nicht, wenn Option Base verwendet wird.

Dim a(3) sieht vier Elemente vor: a(0), a(1), a(2) und a(3). Option Base bringt keine Änderung der Anzahl der Elemente, es ändert nur die Indexierung. Mit Option Base 1 ergibt dieselbe Deklarierung immer noch vier Elemente: a(1), a(2), a(3) und a(4). Dieses Verhalten ist für mich nicht gerade intuitiv. Ich kann daher den Gebrauch von Option Base nicht empfehlen. Wenn Sie spezifische Arraybereichsgrenzen benötigen, ist am besten, sie explizit zu deklarieren, zum Beispiel Dim a(1 To 4). Option Base hat ihre Tücken, wenn es um die Erstellung einer sauberen Dokumentierung oder um eine sichere Portierung geht.

Visual Basic behandelt Option Base 1 anders als StarBasic. VB setzt die untere Grenze auf 1, ändert aber die obere Grenze nicht. Visual Basic .NET unterstützt Option Base gar nicht mehr. Wenn Sie „Option Compatible“ verwenden, wird mit „Option Base 1“ die obere Grenze nicht um 1 angehoben. StarBasic verhält sich dann wie VB.

Es ist einfach, lesend oder schreibend auf die Werte in einem Array zuzugreifen. Ein Array auf diesem Weg zu initialisieren ist aber mühselig.

Listing 18. *Demonstration eines einfachen Arrays.*

```
Sub ExampleSimpleArray1
  Dim a(2) As Integer, b(-2 To 1) As Long
  Dim m(1 To 2, 3 To 4)

  REM Erinnern Sie sich, dass mehrere Anweisungen
  REM auf einer Zeile stehen können, mit Doppelpunkt getrennt?
  a(0) = 0    : a(1) = 1   : a(2) = 2
  b(-2) = -2  : b(-1) = -1 : b(0) = 0 : b(1) = 1
  m(1, 3) = 3 : m(1, 4) = 4
  m(2, 3) = 6 : m(2, 4) = 8
  Print "m(2,3) = " & m(2,3)
```

```
  Print "b(-2) = " & b(-2)
End Sub
```

Zum schnellen Füllen eines Arrays vom Typ Variant dient die Funktion Array (s. Listing 19), die ein Variant-Array mit den aufgeführten Werten zurückgibt. Die Funktionen LBound und UBound geben die untere und die obere Grenze des Arraybereichs zurück. Alle von Basic bereitgestellten Array-Routinen sind in Tabelle 13 aufgelistet und werden an späterer Stelle ausführlich erörtert.

Listing 19. *Verwenden Sie Array(), um auf schnelle Art ein Array zu füllen.*

```
Sub ExampleArrayFunction
  Dim a, i%, s$
  a = Array("Null", 1, Pi, Now)
  REM String, Integer, Double, Date
  For i = LBound(a) To UBound(a)
    s$ = s$ & i & " : " & TypeName(a(i)) & " : " & a(i) & Chr$(10)
  Next
  MsgBox s$, 0, "Beispiel für die Funktion Array"
End Sub
```

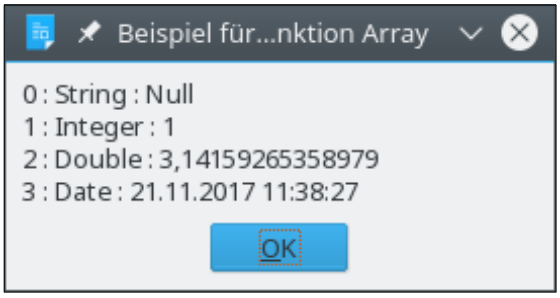


Bild 22. *Unterschiedliche Variablentypen in ein und demselben Array.*

Eine als Array definierte, aber nicht dimensionierte Variable, wie zum Beispiel Dim a(), heißt leeres Array. Prüfen Sie, ob ein Array leer ist, indem Sie die obere mit der unteren Grenze des Arraybereichs vergleichen. Das Array ist leer, das heißt, es ist nicht dimensioniert, wenn die obere Grenze kleiner ist als die untere. Ein dimensioniertes Array, wie Dim a(5), ist nicht leer.

Das Verhalten von LBound und UBound hat sich mit der Zeit geändert. Frühe OOo-Versionen generieren für UBound(b) einen Fehler, andere wiederum nicht. Alle Versionen sollten aber problemlos mit UBound(b()) arbeiten. Zu dem Zeitpunkt, zu dem diese Zeilen geschrieben werden, gibt es einen Fehler bei der Ermittlung der oberen und unteren Grenze für c (in Listing 20), weil c ein leeres Objekt ist.

Listing 20. *Runde Klammern sind nicht immer erforderlich, aber immer erlaubt.*

```
Sub ArrayDimensionError
  On Error Goto ErrorHandler
  Dim a(), b(1 To 2), c
  Dim iLine As Integer
  Dim s$
  REM Gültige Konstrukte
  iLine = 1 : s = "a = (" & LBound(a()) & ", "
  iLine = 2 : s = s & UBound(a) & ")"
  iLine = 3 : s = s & Chr$(10) & "b = (" & LBound(b()) & ", "
  iLine = 4 : s = s & UBound(b) & ")"
  REM Ungültige Konstrukte
  iLine = 5 : s = s & Chr$(10) & "c = (" & LBound(c()) & ", "
  iLine = 6 : s = s & UBound(c) & ")"
  MsgBox s, 0, "LBound und UBound"
```

```
    Exit Sub
    ErrorHandler:
    s = s & Chr$(10) & "Fehler " & Err & ": " & Error$ & " (Zeile : " & iLine & ")"
    Resume Next
End Sub
```

Tabelle 13. *Liste der zu Arrays gehörenden Subroutinen und Funktionen.*

Funktion	Beschreibung
Array(args)	Gibt ein Variant-Array zurück, mit den Argumenten als Elemente.
DimArray(args)	Gibt ein leeres Variant-Array zurück, dimensioniert durch die Argumente.
IsArray(var)	Gibt True zurück, wenn die Variable ein Array ist, ansonsten False.
Join(array) Join(array, trenner)	Gibt einen String zurück, der die einzelnen Array-Elemente hintereinander enthält, jeweils getrennt durch den optionalen Trennstring. Standardtrenner ist das Leerzeichen.
LBound(array) LBound(array, dimension)	Gibt die untere Bereichsgrenze des Arrays zurück. Die optionale Dimensionsangabe bestimmt die zu berücksichtigende Dimension. Die erste Dimension ist 1.
ReDim var(args) As Type	Ändert die Array-Dimensionen mit derselben Syntax wie Dim. Mit dem Schlüsselwort Preserve bleiben die bestehenden Werte erhalten, z.B. ReDim Preserve x(1 To 4) As Integer.
Split(str) Split(str, trenner) Split(str, trenner, n)	Splittet den String in ein Array von Strings. Der Standardtrenner ist ein Leerzeichen. Das optionale Argument „n“ begrenzt die Anzahl der entnommenen Stringelemente.
UBound(array) UBound(array, dimension)	Gibt die obere Bereichsgrenze des Arrays zurück. Die optionale Dimensionsangabe bestimmt die zu berücksichtigende Dimension. Die erste Dimension ist 1.

3.5.1. Die Dimensionen eines Arrays ändern

Die gewünschte Dimensionierung eines Arrays ist nicht immer von vornherein bekannt. Manchmal ist sie zwar bekannt, ändert sich aber periodisch, und der Code muss geändert werden. Eine Array-Variable kann mit oder ohne spezifizierte Dimensionen deklariert werden. Basic bietet ein paar unterschiedliche Methoden, Array-Dimensionen zu setzen oder zu ändern.

Die Funktion Array erstellt ein Array vom Typ Variant, das schon Werte enthält. So hat man schnell ein Array initialisiert. Sie brauchen die Array-Dimensionen nicht zu setzen, aber wenn Sie es tun, werden sie sich ändern und die Dimensionen annehmen, die von der Funktion Array bestimmt sind.

```
Dim a()
a = Array(3.141592654, "PI", 9.81, "Schwerkraft")
```

Die an die Funktion Array übergebenen Argumente werden zu den Elementen des erzeugten Variant-Arrays. Die Funktion DimArray andererseits interpretiert die Argumente als Dimensionierung des zu erzeugenden Arrays (s. Listing 21). Die Argumente können aus Ausdrücken bestehen, somit kann die Dimension mit Hilfe einer Variablen gesetzt werden.

Listing 21. *Ein Array neu dimensionieren.*

```
Sub ExampleDimArray
  Dim a(), i%
  Dim s$
  a = Array(10, 11, 12)
  s = "" & LBound(a()) & " " & UBound(a())       REM 0 2
  a() = DimArray(3)                              REM Dasselbe wie Dim a(3)
  a() = DimArray(2, 1)                           REM Dasselbe wie Dim a(2,1)
  i = 4
  a = DimArray(3, i)                             REM Dasselbe wie Dim a(3,4)
  s = s & Chr$(10) & LBound(a(), 1) & " " & UBound(a(), 1)   REM 0, 3
  s = s & Chr$(10) & LBound(a(), 2) & " " & UBound(a(), 2)   REM 0, 4
```

```
  a() = DimArray()      REM Ein leeres Array
  MsgBox s, 0, "Beispiel für DimArray"
End Sub
```

Die Funktionen Array und DimArray geben beide ein Array von Variant-Elementen zurück. Die Anweisung ReDim ändert die Dimensionen eines bestehenden Arrays. Die Änderung kann sich gleichermaßen auf die einzelnen Dimensionen beziehen wie auch auf die Anzahl der Dimensionen. Die Argumente können aus Ausdrücken bestehen, denn die Anweisung ReDim wird zur Laufzeit ausgewertet.

```
Dim e() As Integer, i As Integer
i = 4
ReDim e(5) As Integer       REM Dimension ist 1, mit gültiger Größe 0 To 5.
ReDim e(3 To 10) As Integer REM Dimension ist 1, mit gültiger Größe 3 To 10.
ReDim e(3, i) As Integer    REM Dimension ist 2, mit gültiger Größe (0 To 3, 0 To 4).
```

Ein paar Tipps zu Arrays:

- LBound und UBound funktionieren mit leeren Arrays.
- Ein leeres Array hat nur eine Dimension, mit der unteren Grenze 0 und der oberen Grenze -1.
- Mit ReDim können Sie ein bestehendes Array leeren.

Die Anweisung ReDim kann mit dem Schlüsselwort Preserve aufgerufen werden. Damit werden nach Möglichkeit bei der Änderung der Dimensionen die Werte der Elemente erhalten. Bei einer Erweiterung der Dimensionierung bleiben alle Daten erhalten, bei einer Einschränkung gehen jedoch Daten verloren, da Elemente abgetrennt werden. Das kann an beiden Enden geschehen. Wenn ein Element des neuen Arrays im alten existierte, bleibt der Wert unverändert. Im Gegensatz zu manchen BASIC-Dialekten erlaubt StarBasic die Änderung aller Dimensionen eines Arrays bei gleichzeitigem Datenerhalt.

```
Dim a() As Integer
ReDim a(3, 3, 3) As Integer
a(1, 1, 1) = 1 : a(1, 1, 2) = 2 : a(2, 1, 1) = 3
ReDim Preserve a(-1 To 4, 4, 4) As Integer
Print "(" & a(1, 1, 1) & ", " & a(1, 1, 2) & ", " & a(2, 1, 1) & ")"
```

ReDim spezifiziert sowohl die Dimensionen als auch einen optionalen Typ. Wenn der Typ mit angegeben ist, muss er dem Typ entsprechen, mit dem die Variable deklariert wurde. Ansonsten erzeugt OOo einen Fehler zur Kompilierungszeit.

Listing 22 ist eine Dienstfunktion, der ein einfaches Array übergeben wird und die einen String mit allen Elementen des Arrays zurückgibt. Der Code des ReDim-Beispiels, auch in Listing 22, verwendet ArrayToString.

Listing 22. *Dienstfunktion Array zu String.*

```
REM ArrayToString übernimmt ein einfaches Array und schreibt den Wert
REM eines jeden Elements des Arrays in einen String.
Function ArrayToString(a() As Variant) As String
  Dim i%, s$
  For i% = LBound(a()) To UBound(a())
    s$ = s$ & i% & " : " & a(i%) & Chr$(10)
  Next
  ArrayToString = s$
End Function

Sub ExampleReDimPreserve
  Dim a(5) As Integer, b(), c() As Integer
  a(0) = 0 : a(1) = 1 : a(2) = 2 : a(3) = 3 : a(4) = 4 : a(5) = 5
```

```
  REM a ist dimensioniert von 0 bis 5, worin gilt a(i) = i
  MsgBox ArrayToString(a()), 0 , "a() zu Anfang"

  REM a wird umdimensioniert von 1 bis 3, worin gilt a(i) = i
  ReDim Preserve a(1 To 3) As Integer
  MsgBox ArrayToString(a()), 0 , "a() nach ReDim"

  REM Array() gibt den Typ Variant zurück.
  REM b ist dimensioniert von 0 bis 9, worin gilt b(i) = i+1
  b = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
  MsgBox ArrayToString(b()), 0 , "b() nach der ersten Zuweisung"

  REM b ist dimensioniert von 1 bis 3, worin gilt b(i) = i+1
  ReDim Preserve b(1 To 3)
  MsgBox ArrayToString(b()), 0 , "b() nach ReDim"

  REM Das folgende ist NICHT gültig, weil das Array schon
  REM auf eine andere Größe dimensioniert ist.
  REM a = Array(0, 1, 2, 3, 4, 5)

  REM c ist dimensioniert von 0 bis 5, worin gilt c(i) = i
  REM Wäre „ReDim" auf c angewendet worden, dann würde folgendes NICHT funktionieren.
  c = Array(0, 1, 2, "drei", 4, 5)
  MsgBox ArrayToString(c()), 0 , "Integer-Array c() dem Typ Variant zugewiesen"

  REM Ironischerweise ist das folgende erlaubt, aber c wird keine Daten enthalten!
  ReDim Preserve c(1 To 3) As Integer
  MsgBox ArrayToString(c()), 0 , "ReDim Integer c() nach der Zuweisung zu Variant"
End Sub
```

In Visual Basic gibt es verschiedene Regelungen, die Dimensionen eines Arrays zu ändern, und diese Regelungen unterscheiden sich von Version zu Version von Visual Basic. Ganz allgemein ist StarBasic flexibler.

3.5.2. Unerwartetes Verhalten von Arrays

Wenn man eine Integer-Variable einer anderen zuweist, wird der Wert kopiert, und die Variablen haben keinen weiteren Bezug mehr zueinander. Anders gesagt, wenn man den Wert der ersten Variablen ändert, ändert sich nichts am Wert der zweiten Variablen. Das gilt nicht für Array-Variablen. Wenn man eine Array-Variable einer anderen zuweist, wird statt einer Kopie eine so genannte Referenz auf das erste Array erstellt. Alle Änderungen an der einen Variablen werden automatisch auch von der anderen erkannt. Es ist unerheblich, welche der beiden geändert wird, immer sind beide betroffen. Das ist der Unterschied zwischen „Argumente übergeben als *Wert*“ (Integers) und „Argumente übergeben als *Referenz*“ (Arrays).

Listing 23. *Arrays werden als Referenz kopiert.*

```
Sub ExampleArrayCopyIsRef
  Dim a(5) As Integer, c(4) As Integer, s$
  c(0) = 4 : c(1) = 3 : c(2) = 2 : c(3) = 1 : c(4) = 0
  a() = c()
  a(1) = 7
  c(2) = 10
  s$ = "**** a() *****" & Chr$(10) & ArrayToString(a()) & Chr$(10) &_
       "**** c() *****" & Chr$(10) & ArrayToString(c())
  MsgBox s$, 0 , "Ändern Sie eins, ändern Sie beide"
End Sub
```

Zur Verdeutlichung, dass Arrays als Referenz zugewiesen werden, erstellen wir drei Arrays – a(), b() und c() – wie in Bild 23 gezeigt. Intern erstellt Basic drei Arrays, die durch a(), b() und c() referenziert werden.

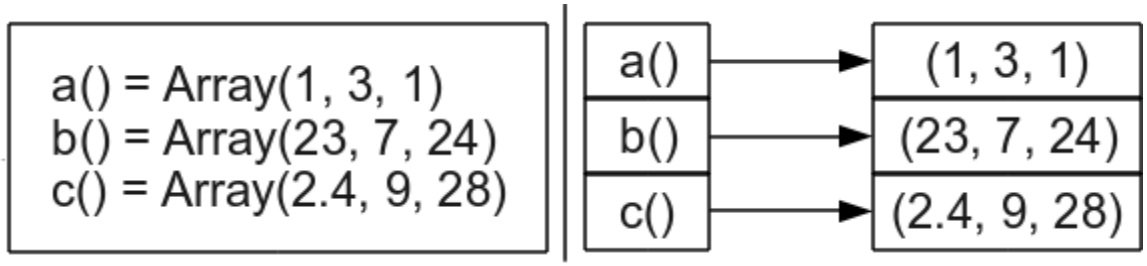


Bild 23. *Die Zuweisung eines Arrays weist eine Referenz zu.*

Weist man das Array a() dem Array b() zu, dann referenzieren a() und b() dieselben Daten. Es ist nicht so, dass die Variable a() die Variable b() referenziert, sondern sie referenziert dieselben Daten, die auch b() referenziert (s. Bild 24). Daher ändert man mit a() gleichzeitig auch b(). Das von a() ursprünglich referenzierte Array wird nun nicht mehr referenziert.

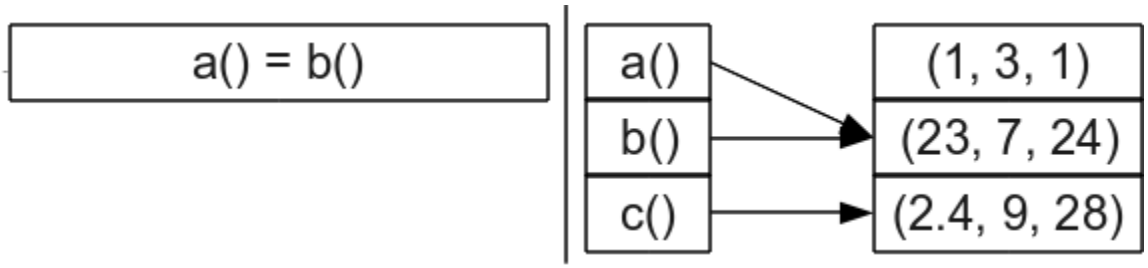


Bild 24. *Die Zuweisung eines Arrays weist eine Referenz zu.*

Weist man das Array b() dem Array c() zu, dann referenzieren b() und c() dieselben Daten. Die Variable a() bleibt unverändert, wie in Bild 25 zu sehen ist.

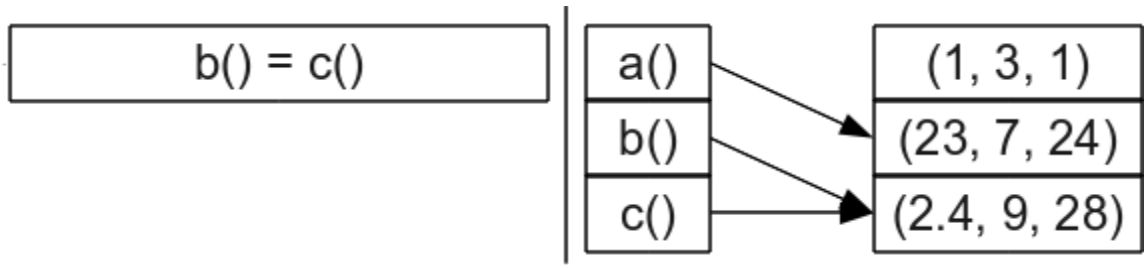


Bild 25. *Die Zuweisung eines Arrays weist eine Referenz zu.*

Achtung Wenn ein Array einem anderen zugewiesen wird, gibt es keine Typüberprüfung. Weisen Sie also keine Arrays unterschiedlichen Typs einander zu.

Weil es keine Typüberprüfung bei der Zuweisung eines Arrays zu einem anderen gibt, können unerwartete und undurchsichtige Probleme entstehen. Die Funktion Array gibt ein Variant-Array zurück und ist die schnellste Methode, mehrere Werte einer Array-Variablen zuzuweisen. Ein offensichtliches Problem ist, dass ein Integer-Array plötzlich String-Werte enthalten kann, wenn es ein Variant-Array referenziert. Ein weniger offensichtliches Problem ist, dass die Anweisung ReDim auf dem deklarierten Typ aufsetzt. Die Anweisung „ReDim Preserve“, angewendet auf ein Integer-Array, das einem Variant-Array zugewiesen wurde, versagt darin, die bestehenden Werte zu erhalten.

```
Dim a() As Integer                  REM Deklariert a() als Integer()
a() = Array(0, 1, 2, 3, 4, 5, 6)    REM Zuweisung eines Variant() zu einem Integer()
ReDim Preserve a(1 To 3) As Integer REM Leert das Array
```

Es braucht eine andere Methode zur sicheren Array-Zuweisung, wenn man den korrekten Datentyp erhalten möchte. Kopieren Sie jedes Element des Arrays einzeln. So wird auch verhindert, dass zwei Array-Variablen dasselbe Array referenzieren.

Listing 24. *Komplexeres Array-Beispiel.*

```
Sub ExampleSetIntArray
  Dim iA() As Integer
  SetIntArray(iA, Array(9, 8, "7", "sechs"))
  MsgBox ArrayToString(iA), 0, "Ein Variant-Array einem Integer-Array zuweisen"
End Sub

REM Das erste Array erhält dieselben Dimensionen wie das zweite.
REM Dann wird eine elementweise Kopie des Arrays vorgenommem.
Sub SetIntArray(iArray() As Integer, v() As Variant)
  Dim i As Long
  ReDim iArray(LBound(v()) To UBound(v())) As Integer
  For i = LBound(v) To UBound(v)
    iArray(i) = v(i)
  Next
End Sub
```

3.6. Subroutinen und Funktionen

Subroutinen sind Codezeilen, die zu sinnvollen Arbeitsabschnitten gruppiert sind. Eine Funktion ist eine Subroutine, die einen Wert zurückgibt. Der Einsatz von Subroutinen und Funktionen erleichtert die Fehlersuche, die Mehrfachverwendung und die Lesbarkeit des Codes. So werden Fehlerquellen reduziert.

Das Schlüsselwort Sub bestimmt den Beginn einer Subroutine, End Sub bestimmt das Ende.

```
Sub FirstSub
  Print "Führe FirstSub aus"
End Sub
```

Zum Aufruf einer Subroutine schreiben Sie den Namen der Subroutine in eine Zeile. Optional können Sie das Schlüsselwort Call davorsetzen.

```
Sub Main
  Call FirstSub ' Ruft das Sub FirstSub auf.
  FirstSub      ' Ruft das Sub FirstSub noch einmal auf.
End Sub
```

Die Namen von Subroutinen und Funktionen dürfen in einem Modul nur einmal vorkommen. Die Regeln zur Namensgebung sind dieselben wie bei Variablen, auch der Umgang mit Leerzeichen darin.

```
Sub One
  [Name mit Leerzeichen]
End Sub

Sub [Name mit Leerzeichen]
  Print "Hier bin ich"
End Sub
```

In Visual Basic darf einer Subroutine ein optionales Schlüsselwort wie Public oder Private vorangehen. Seit OOo 2.0 können Sie eine Routine als Public oder Private definieren, aber die Routine ist immer Public, außer wenn vorher CompatibilityMode(True) gesetzt ist.

Deklarieren Sie eine Subroutine als Private, indem Sie dem Schlüsselwort Sub das Schlüsselwort Private voranstellen. Dann kann die Subroutine nur innerhalb des Moduls aufgerufen werden.

```
Private Sub PrivSub
  Print "In Private Sub"
  bbxx = 4
End Sub
```

Option Compatible reicht nicht, es muss CompatibilityMode(True) gesetzt sein, um den Private-Status zu ermöglichen,.

```
Sub TestPrivateSub
  CompatibilityMode(False) 'Nur nötig, wenn vorher CompatibilityMode(True) gesetzt war.
  Call PrivSub()           'Dieser Aufruf funktioniert immer.
  CompatibilityMode(True)  'Erforderlich, auch wenn Option Compatible gesetzt ist.
  Call PrivSub()           'Laufzeitfehler (wenn PrivSub in einem anderen Modul steht).
End Sub
```

Mit dem Schlüsselwort Function wird eine Funktion deklariert, die wie eine Variable ihren Rückgabetyp definieren kann. Ohne deklarierten Typ wird der Standardtyp Variant zurückgegeben. Sie können den Rückgabewert innerhalb des Funktionscodes an jeder Stelle und so oft Sie wollen zuweisen. Der letzte zugewiesene Wert wird zurückgegeben.

```
Sub Test
  Print "Die Funktion gibt " & TestFunc & " zurück."
End Sub

Function TestFunc As String
  TestFunc = "Hallo"
End Function
```

3.6.1. Argumente

Eine Variable, die einer Routine übergeben wird, nennt man Argument. Argumente müssen deklariert werden. Dann spricht man auch von Parametern. Für die Deklarierung von Parametern mit Namen und Typen gelten dieselben Regeln wie für Variablen.

Einer Routine kann optional ein Paar runder Klammern folgen, sowohl bei der Definition als auch beim Aufruf. Bei einer Routine, die Argumente übernimmt, kann die Parameterliste optional in runden Klammern stehen. Diese Liste folgt dem Namen der Routine direkt in derselben Zeile. Zwischen dem Namen und den Parametern dürfen Leerzeichen stehen.

Listing 25. *Einfacher Argumentetest.*

```
Sub ExampleParamTest1()
  Call ParamTest1(2, "Zwei")
  Call ParamTest1 1, "Eins"
End Sub

Sub ParamTest1(i As Integer, s$)
  Print "Integer = " & i & " String = " & s$
End Sub
```

Übergabe als Referenz oder als Wert

Standardmässig werden Argumente als Referenz übergeben und nicht als Wert. Anders gesagt, wenn die aufgerufene Subroutine ein Argument verändert, sieht die aufrufende Subroutine die Änderung. Man kann mit dem Schlüsselwort ByVal dieses Verhalten dahin abwandeln, dass eine Kopie des Arguments (statt einer Referenz auf das Argument) versendet wird (s. Listing 26 und Bild 26).

Achtung Wenn der Wert von Konstanten, die als Referenz-Argumente übergeben werden, in der aufgerufenen Routine modifiziert wird, kann es zu unerwartetem Verhalten kommen. Der Wert kann innerhalb der aufgerufenen Routine beliebig zurückgehen. Ich hatte zum Beispiel eine Subroutine, die in einer Schleife ein Integer-Argument bis null zurückzählen sollte. Das Argument kam nie auf null.

Listing 26. *Argumente als Referenz und als Wert.*

```
Sub ExampleParamValAndRef()
  Dim i1%, i2%
  i1 = 1 : i2 = 1
  ParamValAndRef(i1, i2)
  MsgBox "Das als Referenz übergebene Argument war 1 und ist nun " & i1 & Chr$(10) &_
         "Das als Wert übergebene Argument war 1 und ist immer noch " & i2 & Chr$(10)
End Sub

Sub ParamValAndRef(iRef%, ByVal iVal)
  iRef = iRef + 1  ' Dies wird sich auf die aufrufende Routine auswirken.
  iVal = iVal - 1  ' Dies wird sich nicht auf die aufrufende Routine auswirken.
End Sub
```

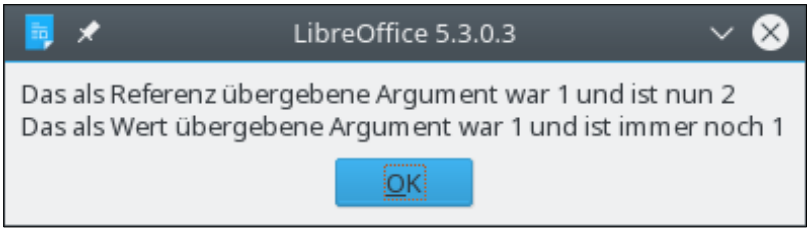


Bild 26. *Durch die Übergabe als Referenz können Änderungen an die aufrufende Routine zurückgegeben werden.*

Achtung In Visual Basic ist es Standard, Argumente als Wert zu übergeben, optional auch mit dem Schlüsselwort ByVal.. Die Übergabe als Referenz benötigt das Schlüsselwort ByRef. Dieses Schlüsselwort wurde mit OOo 2.0 auch in StarBasic eingeführt. Doch beachten Sie, dass in StarBasic die Übergabe als Referenz das Standardverhalten ist.

Eine Variable kann nicht als Referenz übergeben werden, wenn die Typen nicht übereinstimmen. Die Makros im Listing 27 und im Listing 28 unterscheiden sich durch den deklarierten Parametertyp. (Zu den Funktionen Split() und Join() s. Abschnitt 5.4. Array zu String und wieder zurück.)

Listing 27. *Einfacher Swap (Tausch) mit einem String-Argument.*

```
Sub sSwap(sDatum As String)
  Dim asDatum(1 To 3) As String
  Dim sDummy As String

  asDatum = Split(sDatum, ".")  '-> Variant-Array(0 To 2): z. B. ("23", "10", "2015")
  sDummy = asDatum(0)
  asDatum(0) = asDatum(2)
  asDatum(2) = sDummy
  sDatum = Join(asDatum,"-")    '-> String z.B. "2015-10-23"
End Sub
```

Listing 28. *Einfacher Swap (Tausch) mit einem Variant-Argument.*

```
Sub vSwap(vDatum As Variant)
  Dim asDatum(1 To 3) As String
  Dim sDummy As String

  asDatum = Split(vDatum, ".")  '-> Variant-Array(0 To 2): z. B. ("23", "10", "2015")
```

```
    sDummy = asDatum(0)
    asDatum(0) = asDatum(2)
    asDatum(2) = sDummy
    vDatum = Join(asDatum,"-")     '-> String z.B. "2015-10-23"
End Sub
```

Das folgende Makro ruft ein Makro, das ein Variant-Argument erwartet, und ein Makro, das ein String-Argument erwartet, jeweils mit einer Variant- und einer Stringvariablen auf. Wenn die Variant-Variable derjenigen Routine übergeben wird, die ein String-Argument erwartet, wird die Variable als Wert übergeben. Dies ist überraschend, denn das Standardverhalten ist die Übergabe als Referenz.

Listing 29. *Test der Referenzübergabe mit verschiedenen Argumenttypen.*

```
Sub PassByReferenceTester
  Dim vVar As Variant
  Dim sVar As String
  Dim s As String

  vVar = "01.02.2011"
  sVar = "01.02.2011"

  s = vVar & " sSwap( variant var string arg ) ==> "
  sSwap(vVar)
  s = s & vVar & Chr$(10)

  s = s & sVar & " sSwap( string var string arg ) ==> "
  sSwap(sVar)
  s = s & sVar & Chr$(10)

  vVar = "01.02.2011"
  sVar = "01.02.2011"
  s = s & vVar & " vSwap( variant var variant arg ) ==> "
  vSwap(vVar)
  s = s & vVar & Chr$(10)

  s = s & sVar & " vSwap( string var variant arg ) ==> "
  vSwap(sVar)
  s = s & sVar & Chr$(10)

  MsgBox(s)
End Sub
```

Es ist wichtig zu verstehen, wann eine Variable an eine Routine als Referenz und wann als Wert übergeben wird. Gleichermaßen wichtig ist es zu verstehen, wann ein Wert in einer Variablen mit seinem Wert oder als Referenz kopiert wird.

- Variablen mit einfachen Typen werden als Wert kopiert, zum Beispiel wenn eine Integer-Variable einer anderen zugewiesen wird.
- Arrays werden immer als Referenz kopiert. Wenn man ein Array einem anderen zuweist, referenzieren und modifizieren beide Variablen dasselbe Array.
- UNO Services werden als Referenz kopiert. Das bedeutet, dass man so etwas wie oDoc = ThisComponent machen kann und dann beide Variablen dasselbe Objekt referenzieren.
- Structs werden mit ihren Werten kopiert. Viele Nutzer sind frustriert, wenn sie zum ersten Mal auf dieses Verhalten stoßen, aber es gibt einen sehr guten Grund dafür. Zuerst einmal das

Problem: `oBorder.TopLine.OuterLineWidth = 2` macht nicht das, was man wollte, weil TopLine ein Struct ist und der Wert als Kopie und nicht als Referenz zurückgegeben wird. Das Codebröckchen ändert in dieser Form die äußere Randbreite in einer Kopie des Structs und nicht in dem Struct, das mit dem Border-Objekt verknüpft ist. Folgendes ist der richtige Weg, den Rahmen zu ändern:

```
v = oBorder.TopLine : v.OuterLineWidth = 2 : oBorder.TopLine = v.
```

Laut der Aussage eines führenden Entwicklers des originalen OOo gibt es ein oder zwei Services und Structs, die nicht wie erwartet zugewiesen bzw. kopiert werden, aber er wusste nicht mehr welche. Mir sind die Objekte, die die Regeln verletzen, nicht untergekommen, aber ich versuche, das Problem im Kopf zu behalten, um nicht auf dem falschen Fuß erwischt zu werden.

Optionale Argumente

Man kann Parameter als optional deklarieren, indem man ihnen das Schlüsselwort Optional voranstellt. Alle auf einen optionalen Parameter folgenden Parameter müssen auch optional sein. Mit der Funktion IsMissing finden Sie heraus, ob ein optionales Argument fehlt oder nicht. (Zu der Funktion IIf() s. Abschnitt 3.9.4. IIf.)

Listing 30. *Optionale Argumente.*

```
REM Testaufrufe mit optionalen Argumenten.
REM Aufrufe mit Argumenten vom Typ Integer und Variant sollten
REM dieselben Ergebnisse zeigen. Leider tun sie es nicht.
Sub ExampleArgOptional()
  Dim s$
  s = "Variant-Argumente () => "     & TestOpt()      & Chr$(10) &_
      "Integer-Argumente () => "     & TestOptI()     & Chr$(10) &_
      "-----------------------------------------" & Chr$(10) &_
      "Variant-Argumente (,,) => "   & TestOpt(,,)    & Chr$(10) &_
      "Integer-Argumente (,,) => "   & TestOptI(,,)   & Chr$(10) &_
      "-----------------------------------------" & Chr$(10) &_
      "Variant-Argumente (1) => "    & TestOpt(1)     & Chr$(10) &_
      "Integer-Argumente (1) => "    & TestOptI(1)    & Chr$(10) &_
      "-----------------------------------------" & Chr$(10) &_
      "Variant-Argumente (,2) => "   & TestOpt(,2)    & Chr$(10) &_
      "Integer-Argumente (,2) => "   & TestOptI(,2)   & Chr$(10) &_
      "-----------------------------------------" & Chr$(10) &_
      "Variant-Argumente (1,2) => "  & TestOpt(1,2)   & Chr$(10) &_
      "Integer-Argumente (1,2) => "  & TestOptI(1,2)  & Chr$(10) &_
      "-----------------------------------------" & Chr$(10) &_
      "Variant-Argumente (1,,3) => " & TestOpt(1,,3)  & Chr$(10) &_
      "Integer-Argumente (1,,3) => " & TestOptI(1,,3) & Chr$(10)
  MsgBox s, 0, "Optionale Argumente vom Typ Variant oder Integer"
End Sub

REM Gibt einen String zurück, der jedes Argument enthält.
REM Wenn das Argument fehlt, wird ein M an seine Stelle gesetzt.
Function TestOpt(Optional v1, Optional v2, Optional v3) As String
  TestOpt = "" & IIf(IsMissing(v1), "M", Str(v1)) &_
                 IIf(IsMissing(v2), "M", Str(v2)) &_
                 IIf(IsMissing(v3), "M", Str(v3))
End Function

REM Gibt einen String zurück, der jedes Argument enthält.
REM Wenn das Argument fehlt, wird ein M an seine Stelle gesetzt.
Function TestOptI(Optional i1%, Optional i2%, Optional i3%) As String
  TestOptI = "" & IIf(IsMissing(i1), "M", Str(i1)) &_
                  IIf(IsMissing(i2), "M", Str(i2)) &_
```

```
                IIf(IsMissing(i3), "M", Str(i3))
End Function
```

Sie können jedes optionale Argument weglassen. Listing 30 zeigt zwei Funktionen, die optionale Argumente akzeptieren. Die Funktionen sind identisch bis auf die Argument-Typen. Jede Funktion gibt einen String zurück, der die Werte der Argumente aneinanderfügt. Weggelassene Argumente werden durch den Buchstaben „M“ repräsentiert. Obwohl die Rückgabestrings von TestOpt und TestOpt1 für dieselbe Argumentliste identisch sein sollten, sind sie es nicht (s. Bild 27). Dies ist ein Bug.

Achtung Die Funktion IsMissing gibt falsche Ergebnisse für Variablen, die nicht vom Typ Variant sind, wenn dem fehlenden Argument ein Komma folgt.

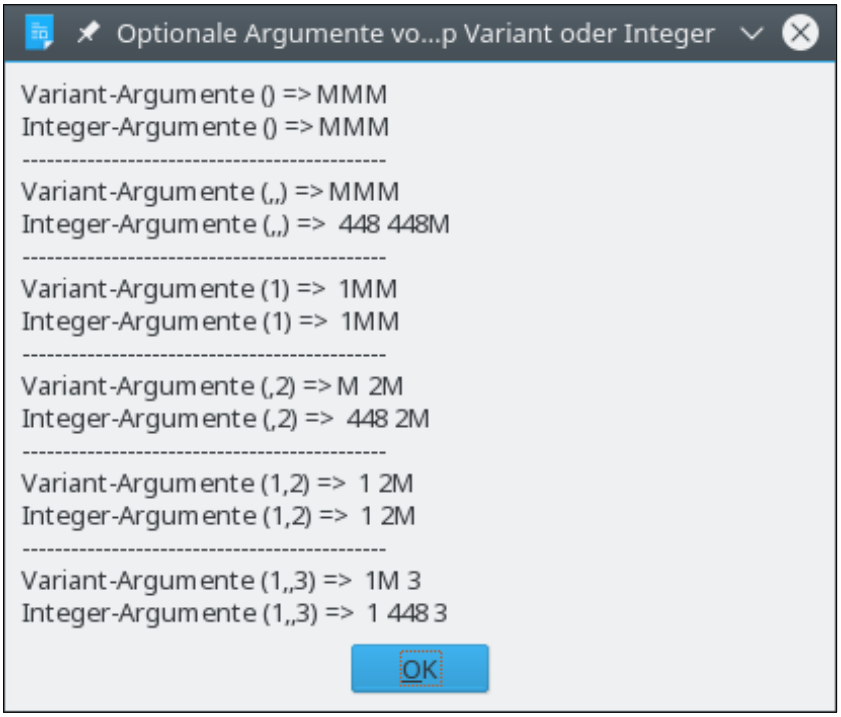


Bild 27. *In seltenen Fällen gibt es Fehler bei optionalen Argumenten, die nicht den Typ Variant haben.*

Vorgegebene Argumentwerte

Seit OOo 2.0 sind vorgegebene Werte für weggelassene Argumente möglich. Das heißt, man kann einen Wert vorgeben für den Fall, dass ein optionales Argument fehlt. Vorgabewerte funktionieren aber nur, wenn „Option Compatible“ gesetzt ist.

```
Option Compatible
Sub DefaultExample(Optional n As Integer=100)
  REM If IsMissing(n) Then n = 100 'Das brauche ich nie mehr!
  Print n
End Sub
```

3.6.2. Rekursive Routinen

Eine rekursive Routine ruft sich selber auf. Betrachten Sie einmal die Berechnung der mathematischen Funktion Fakultät für positive Ganzzahlen. Die übliche Definition ist rekursiv.

Listing 31. *Rekursive Berechnung der Fakultät.*

```
Sub DoFactorial
  Print "Rekursive Fakultät = " & RecursiveFactorial(4)
  Print "Iterative Fakultät = " & IterativeFactorial(4)
End Sub

Function IterativeFactorial(ByVal n As Long) As Long
  Dim answer As Long
  answer = 1
  Do While n > 1
    answer = answer * n
    n = n - 1
  Loop
  IterativeFactorial = answer
End Function

Function RecursiveFactorial(ByVal n As Long) As Long
  RecursiveFactorial = 1
  If n > 1 Then RecursiveFactorial = n * RecursiveFactorial(n-1)
End Function
```

Rechner verwenden eine Datenstruktur, die Stapel heißt. Zuhause habe ich einen Stapel Bücher, die ich noch lesen will. Wenn ein neues Buch hinzukommt, lege ich es oben auf den Stapel. Wenn ich Zeit zum Lesen finde, nehme ich das Buch oben vom Stapel. So ähnlich funktioniert die Datenstruktur im Rechner: ein Stapel ist ein Speicherbereich zur temporären Speicherung, in dem der zuletzt gespeicherte Wert der erste ist, der zurückgeholt wird. Stapel werden üblicherweise eingesetzt, wenn ein Rechner eine Routine aufruft und Argumente übergibt. Im Folgenden eine typische Prozedur:

1. Die aktuelle Programmposition wird auf den Stapel geschoben.
2. Jedes Argument wird auf den Stapel geschoben.
3. Die gewünschte Funktion oder Subroutine wird aufgerufen.
4. Die aufgerufene Routine verwendet die Argumente vom Stapel.
5. Die aufgerufene Routine nutzt häufig den Stapel zur Speicherung der eigenen Variablen.
6. Die aufgerufene Routine entfernt die Argumente vom Stapel.
7. Die aufgerufene Routine entfernt und sichert die Position der aufrufenden Routine vom Stapel.
8. Wenn die aufgerufene Routine eine Funktion ist, wird der Rückgabewert auf den Stapel geschoben.
9. Die aufgerufene Routine kehrt über die vom Stapel gesicherte Position zur aufrufenden Routine zurück.
10. Wenn die aufgerufene Routine eine Funktion ist, wird der Rückgabewert vom Stapel geholt.

Obwohl eine Reihe von Optimierungen wirken, gibt es immer einen Überhang im Zusammenhang mit dem Aufruf von Subroutinen und Funktionen. Der Überhang besteht in der Verarbeitungszeit und im benötigten Speicherplatz. Die rekursive Fakultät-Version ruft sich kontinuierlich selber auf. Bei der Berechnung der Fakultät von vier (4!) enthält der Stapel zu einem Zeitpunkt gleichzeitig die Informationen für die Aufrufe für 4, 3, 2 und 1. Für manche Funktionen – zum Beispiel für die Fibonacci-Reihe – dürfte sich dieses Verhalten verbieten und stattdessen einen nicht-rekursiven Algorithmus erfordern, damit es nicht zu einem Speicherüberlauf kommt.

3.7. Gültigkeitsbereich von Variablen, Subroutinen und Funktionen

Bei dem Konzept des Gültigkeitsbereichs geht es um die Lebenszeit und Sichtbarkeit von Variablen, Subroutinen und Funktionen in Basic. Der Gültigkeitsbereich hängt ab vom Ort der Deklarierung

und den Schlüsselwörtern Public, Private, Static und Global. Dim ist äquivalent zu Private, aber Variablen sind nur Private, wenn CompatibilityMode(True) gesetzt ist.

3.7.1. Lokale Variablen, in einer Subroutine oder Funktion deklariert

Variablen, die innerhalb einer Subroutine oder Funktion deklariert werden, heißen lokale Variablen. Häufig wird auch gesagt, dass eine Variable zu der Routine gehört, in der sie deklariert wird.

Sie können eine Variable mit Hilfe des Schlüsselworts Dim innerhalb einer Subroutine oder Funktion deklarieren. Solche Variablen sind nur innerhalb der Routine sichtbar. Es ist nicht möglich, auf eine in einer Routine deklarierte Variable von außerhalb der Routine direkt zuzugreifen. Es ist jedoch möglich, von innerhalb einer Routine auf eine Variable zuzugreifen, die außerhalb jeglicher Routine deklariert wurde – zum Beispiel im Kopf eines Moduls. Wenn innerhalb einer Routine der Name einer Variablen oder einer Routine erscheint, sucht Basic in folgender Reihenfolge nach der Variablen oder Routine: aktuelle Routine, Modul, Bibliothek und andere offene Bibliotheken. Mit anderen Worten, Basic startet innen und arbeitet sich nach außen vor.

In einer Routine definierte Variablen werden jedes Mal geschaffen und initialisiert, wenn die Routine gestartet wird. Jedes Mal, wenn die Routine verlassen wird, werden die Variablen entfernt, weil die Routine nicht mehr existiert. Wenn allerdings eine Routine verlassen wird, um eine weitere Routine aufzurufen, ist das kein Anlass zur Reinitialisierung der Variablen.

Mit dem Schlüsselwort Static ändern Sie den Zeitpunkt, zu dem eine Variable erstellt und entfernt wird, auf den Moment, in dem das Makro startet beziehungsweise endet. Auch wenn die Variable immer noch nur innerhalb der aktuellen Routine sichtbar ist, wird sie nur einmal initialisiert, nämlich wenn das Makro gestartet wird, und sie behält ihren Wert über mehrfache Aufrufe dieser Routine hinweg. Sagen wir einmal, Sie starten, wenn noch kein Makro läuft. Wenn nun eine Subroutine oder eine Funktion, die darin enthaltene statische Variablen nutzt, zum ersten Mal aufgerufen wird, erhalten die Variablen ihre Initialwerte gemäß ihren Typen. Die statischen Variablen behalten ihren Wert zwischen den Aufrufen, solange das Makro als Ganzes nicht beendet ist. Die Syntax des Schlüsselworts Static ist dieselbe wie bei dem Schlüsselwort Dim, und es ist nur innerhalb einer Subroutine oder Funktion gültig. Listing 32 ruft eine Routine auf, die eine statische Variable verwendet.

Listing 32. *Beispiel für Static.*

```
Sub ExampleStatic
  ExampleStaticWorker()
  ExampleStaticWorker()
End Sub

Sub ExampleStaticWorker
  Static iStatic1 As Integer
  Dim iNonStatic As Integer

  iNonStatic = iNonStatic + 1
  iStatic1 = iStatic1 + 1
  Msgbox "iNonStatic = " & iNonStatic & Chr$(10) &_
         "iStatic1 = " & iStatic1
End Sub
```

3.7.2. In einem Modul definierte Variablen

Die Anweisungen Dim, Global, Public oder Private werden für die Variablendeklarierung im Kopf eines Moduls verwendet. Global, Public und Private haben dieselbe Syntax wie Dim, können aber keine Variablen innerhalb einer Subroutine oder Funktion deklarieren. Jede Variablenart hat eine andere Lebensdauer, wie Tabelle 14 zeigt.

Die Schlüsselwörter Static, Public, Private und Global sind keine Modifizierer für das Schlüsselwort Dim, sie werden anstelle von Dim verwendet.

Obwohl es manchmal notwendig ist, eine Variable im Modulkopf zu deklarieren, sollten Sie aber nach Möglichkeit davon absehen. Im Kopf deklarierte Variablen sind auch in anderen Modulen sichtbar, wo sie vielleicht nicht erwartet werden. Es ist nicht einfach herauszufinden, warum sich der Kompilierer beschwert, dass eine Variable schon definiert sei, wenn sie in einer anderen Bibliothek oder einem anderen Modul deklariert ist. Schlimmer noch könnten zwei aktive Bibliotheken die Arbeit wegen Namenskonflikten ganz einstellen.

Tabelle 14. *Lebensdauer einer Variablen, die im Kopf eines Moduls definiert ist.*

Schlüsselwort	Initialisiert	Stirbt	Gültigkeitsbereich
Global	Kompilierung	Kompilierung	Alle Module und Bibliotheken.
Public	Makrostart	Makroende	Bibliothekscontainer, in der die Deklarierung erfolgt
Dim	Makrostart	Makroende	Bibliothekscontainer, in der die Deklarierung erfolgt
Private	Makrostart	Makroende	Modul, in der die Deklarierung erfolgt

Global

Eine als Global deklarierte Variable ist für jedes Modul in jeder Bibliothek erreichbar. Die Bibliothek, die die globale Variable enthält, muss geladen sein, damit die Variable sichtbar ist.

Wenn eine Bibliothek geladen wird, wird sie automatisch kompiliert und gebrauchsfertig gemacht. Zu diesem Zeitpunkt wird die globale Variable initialisiert. Änderungen an einer globalen Variablen werden von jedem Modul gesehen und bleiben erhalten, auch nachdem das Makro beendet ist. Globale Variablen werden zurückgesetzt, wenn die besitzende Bibliothek kompiliert wird. OOo zu schließen und neu zu starten bewirkt die Neukompilierung aller Bibliotheken und die Initialisierung aller globalen Variablen. Änderungen am Quelltext des Moduls, das die globale Definition enthält, erzwingt gleichermaßen die Neukompilierung des Moduls.

```
Global iNumberOfTimesRun
```

Als global deklarierte Variablen ähneln denen, die als Static deklariert sind. Der Unterschied liegt darin, dass Static nur für lokale Variablen gilt und Global nur für Variablen, die im Kopf deklariert werden.

Public

Mit Public deklarierte Variablen sind für alle Module des Bibliothekscontainers sichtbar, in dem sie deklariert sind. Außerhalb dieses Bibliothekscontainers sind Public-Variablen nicht sichtbar. Public-Variablen werden bei jedem Aufruf eines Makros initialisiert.

Eine Anwendungsbibliothek ist eine Bibliothek, die im Bibliothekscontainer der Anwendung (OpenOffice oder LibreOffice) aufgeführt ist. Sie ist verfügbar, wenn OOo läuft, ist in ihrem eigenen Verzeichnis gespeichert, und jedes Dokument kann sie sehen. Bibliotheken auf Dokumentbasis sind in OOo-Dokumenten gespeichert. Die Bibliotheken werden als Teil des Dokuments gespeichert und sind von außerhalb des Dokuments nicht sichtbar.

Public-Variablen, die in einer Anwendungsbibliothek deklariert werden, sind in jeder Dokumentbibliothek sichtbar. Public-Variablen, die in einer Dokumentbibliothek deklariert werden, sind in Anwendungsbibliotheken nicht sichtbar. Mit der Deklarierung einer Public-Variablen in einer Dokumentbibliothek wird eine in einer Anwendungsbibliothek deklarierte Public-Variable effektiv verborgen. Schlicht und einfach (s. Tabelle 15), wenn Sie eine Public-Variable in einem Dokument deklarieren, ist sie nur im Dokument sichtbar und wird eine Public-Variable verbergen, die mit demselben Namen außerhalb des Dokuments deklariert wurde. Eine in der Anwendung deklarierte Public-Variable ist überall sichtbar – falls nicht eine Variablendeklarierung mit eher lokalem Gültigkeitsbereich die Oberhand über die Deklarierung mit eher globalem Gültigkeitsbereich gewinnt.

```
Public oDialog As Variant
```

Tabelle 15. *Der Gültigkeitsbereich einer Public-Variablen je nach Ort der Deklarierung.*

Ort der Deklarierung	Gültigkeitsbereich
Anwendung	Überall sichtbar.
Dokument	Sichtbar nur im Dokument der Deklarierung.
Anwendung und Dokument	Makros im Dokument können die Variable der Anwendungsebene nicht sehen.

Private oder Dim

Deklarieren Sie eine Variable mit Private oder Dim, wenn Sie sie nur in einem Modul verwenden wollen. Private-Variablen werden wie Public-Variablen bei jedem Start eines Makros initialisiert. Ein und derselbe Variablenname kann in zwei verschiedenen Modulen als jeweils eigener Name verwendet werden, wenn die Variable als Private deklariert wird.

```
Private oDialog As Variant
```

- Die Deklarierung einer Variablen mit Dim ist äquivalent zur Deklarierung mit Private.
- Private-Variablen sind jedoch nur privat nach der Anweisung CompatibilityMode(True).
- Option Compatible hat keinen Effekt auf Private-Variablen.

Eine Private-Variable ist außerhalb des deklarierenden Moduls sichtbar, es sei denn, CompatibilityMode(True) ist gesetzt. Schauen Sie selbst, erstellen Sie zwei Module – Modul1 und Modul2 – in derselben Bibliothek. Fügen Sie in Modul1 die Deklarierung „Private priv_var As Integer“ ein. Makros in Modul2 können auf die Variable „priv_var“ zugreifen. Sogar wenn Modul2 in einer anderen Bibliothek desselben Dokuments liegt, ist die Variable „priv_var“ sichtbar und nutzbar. Wenn jedoch CompatibilityMode(True) gesetzt ist, dann ist die Private-Variable nicht mehr außerhalb des deklarierenden Moduls sichtbar.

Deklarieren Sie in Modul1 die Variable „Private priv_var As Double“. In Modul2 wird eine Variable mit demselben Namen deklariert, aber als Integer-Variable. Jedes Modul sieht seine eigene private Variable. Wenn man diese Variablen nicht als Private, sondern als Public deklariert, tritt eine unschöne Situation ein: nur eine dieser Variablen ist sichtbar und nutzbar, aber man weiß nicht welche, außer man führt einen Test durch. Weisen Sie der Variablen 4.7 zu und schauen Sie, ob es Integer oder Double wird.

3.8. Operatoren

Ein Operator ist ein Symbol, das eine mathematische oder logische Operation kennzeichnet oder durchführt. Ein Operator gibt wie eine Funktion ein Resultat zurück. Zum Beispiel addiert der Operator + zwei Zahlen. Die Argumente des Operators heißen Operanden. Operatoren haben Prioritäten. Ein Operator mit der Priorität 1 steht sozusagen auf der höchsten Prioritätsstufe. Schließlich ist es die Nummer 1.

Tipp Beim Druck mathematischer Gleichungen wird das Minuszeichen durch das Unicodezeichen U+2212 (−) dargestellt. In Basic muss stattdessen das ASCII-Zeichen 45 (-) verwendet werden.

In Basic (s. Tabelle 16) werden Operatoren von links nach rechts ausgewertet, mit der Einschränkung, dass ein Operator mit einer höheren Priorität vor einem Operator mit einer niedrigeren Priorität wirkt. 1 + 2 * 3 ergibt 7, weil die Multiplikation eine höhere Priorität hat als die Addition. Durch die Verwendung runder Klammern können Sie die Reihenfolge ändern. Zum Beispiel ergibt (1 + 2) * 3 den Wert 9, weil der Ausdruck innerhalb der runden Klammern zuerst ausgewertet wird.

Tabelle 16. *Operatoren in StarBasic.*

Priorität	Operator	Typ	Beschreibung
1	Not	Unär	Logisches oder bitweises Not
1	-	Unär	Minus als Vorzeichen, Negation
1	+	Unär	Plus als Vorzeichen
2	^	Binär	Numerische Potenzierung. In der Mathematik hätte die Potenzierung eine höhere Priorität als die Negation.
3	*	Binär	Numerische Multiplikation
3	/	Binär	Numerische Division
4	Mod	Binär	Numerischer Rest nach Division
5	\	Binär	Ganzzahlige Division
6	-	Binär	Numerische Subtraktion
6	+	Binär	Numerische Addition und String-Verkettung
7	&	Binär	String-Verkettung
8	Is	Binär	Referenzieren beide Operanden dasselbe Objekt?
8	=	Binär	Gleich
8	<	Binär	Kleiner als
8	>	Binär	Größer als
8	<=	Binär	Kleiner als oder gleich
8	>=	Binär	Größer als oder gleich
8	<>	Binär	Ungleich
9	And	Binär	Bitweises UND für Zahlen, logisches UND für Boolean
9	Or	Binär	Bitweises ODER für Zahlen, logisches ODER für Boolean
9	Xor	Binär	Exklusives ODER, bitweise für Zahlen, logisch für Boolean
9	Eqv	Binär	Äquivalenz, bitweise für Zahlen, logisch für Boolean
9	Imp	Binär	Implikation, bitweise für Zahlen, logisch für Boolean

Achtung Die Prioritäten richten sich nicht unbedingt nach dem mathematischen Standard. Zum Beispiel sollte die Negation eine niedrigere Priorität haben als die Potenzierung: -1^2 sollte -1 ergeben, nicht 1.

Visual Basic hat andere Prioritäten für Operatoren – zum Beispiel ist die Reihenfolge der numerischen Potenzierung und Negation umgekehrt, wie auch der ganzzahligen Division und des Rests nach Division.

Das Wort „Binär“ bezeichnet etwas, das auf zwei Sachen basiert. „Unär“ bezeichnet etwas, das auf nur einer Sache basiert. Ein binärer Operator, nicht zu verwechseln mit einer binären Zahl, steht zwischen zwei Operanden. Zum Beispiel 1+2: da verwendet der Additionsoperator zwei Operanden. In Basic werden binäre Operatoren immer von links nach rechts ausgewertet, nach Maßgabe der Operatorenprioritäten. Ein unärer Operator benötigt nur einen Operanden, und zwar direkt rechts neben dem Operator, zum Beispiel – (1+3). Notwendigerweise wird eine Reihe von unären Operatoren von rechts nach links ausgewertet. Zum Beispiel muss in + - (1+3) der rechtsstehende Negationsoperator zuerst ausgewertet werden, so dass sich der Wert -4 ergibt.

3.8.1. Mathematische und String-Operatoren

Mathematische Operatoren können mit allen numerischen Datentypen verwendet werden. Wenn Operanden unterschiedlichen Typs gemischt vorkommen, wird konvertiert, um Genauigkeitsverluste

zu minimieren. Zum Beispiel bewirkt 1 + 3.443 die Konvertierung zu einer Fließkommazahl und nicht zu einer Ganzzahl. Wenn der erste Operand eine Zahl ist und der zweite ein String, wird der String zu einer Zahl konvertiert. Wenn der String nun keine gültige Zahl enthält, wird null zurückgegeben, ohne Fehlermeldung. Einen String direkt einer numerischen Variablen zuzuweisen, ergibt die Zuweisung von null, wiederum ohne Fehlermeldung.

Listing 33. *Strings werden automatisch zu Zahlen konvertiert, wenn es nötig ist.*

```
Dim i As Integer
i = "abc"         'Die Zuweisung eines Strings ohne Zahl ergibt null,
                  'keine Fehlermeldung
Print i           '0
i = "3abc"        'Zuweisung von 3, automatisch konvertiert, so gut es geht
Print i           '3
Print 4 + "abc"   '4
```

Basic versucht automatisch, Typen zu konvertieren. Es gibt keine Fehlermeldungen, wenn ein String verwendet wird, wo eine Zahl benötigt wird. Dazu mehr an späterer Stelle.

Unäres Plus (+) und Minus (-)

Basic erlaubt Leerzeichen zwischen unärem Operator und dem Operanden (s. Tabelle 9). Unäre Operatoren haben auch die höchste Priorität und werden von rechts nach links ausgewertet. Ein Plus als Vorzeichen ist sicherlich nutzlos – es unterstreicht, dass eine Konstante nicht negativ ist, wird aber ansonsten schlicht ignoriert. Ein Minus als Vorzeichen steht für numerische Negation.

Potenzierung (^)

Die numerische Potenzierung unterstützt ganzzahlige und Fließkommaexponenten. Der Potenzierungsoperator kann nur dann mit einer negativen Zahl verwendet werden, wenn der Exponent ganzzahlig ist.

```
result = number^exponent
```

Ein positiver ganzzahliger Exponent arbeitet nach einem einfachen Prinzip. Die Zahl wird mit sich selbst Exponent-mal multipliziert, zum Beispiel 2^4 = 2 * 2 * 2 * 2.

- OOo folgt nicht unbedingt den mathematischen Standardregeln für die Potenzierung:
- Potenzierung hat eine niedrigere Priorität als Negation, so wird -1^2 fälschlicherweise zu 1 ausgewertet. Basic wertet mehrere Exponenten (2^3^4) von links nach rechts aus ((2^3)^4), wohingegen der mathematische Prioritätenstandard von rechts nach links geht (2^(3^4)).

Listing 34. *Demonstration der Potenzierung.*

```
Sub ExampleExponent
  Dim s$
  s = "2^3 = " & 2^3                    REM 2*2*2 = 8
  s = s & Chr$(10) & "3^2 = " & 3^2     REM 3*3 = 9
  s = s & Chr$(10) & "-3^2 = " & -3^2   REM (-3) * (-3) = 9
  s = s & Chr$(10) & "2^3^2 = " & 2^3^2 REM 2^3^2 = 8^2 = 64
  s = s & Chr$(10) & "4^0.5 = " & 4^.5  REM 2
  s = s & Chr$(10) & "4^-0.5 = " & 4^-.5 REM .5
  s = s & Chr$(10) & "-1^2 = " & -1^2   REM 1
  s = s & Chr$(10) & "-(1^2) = " & -(1^2) REM -1
  MsgBox s
End Sub
```

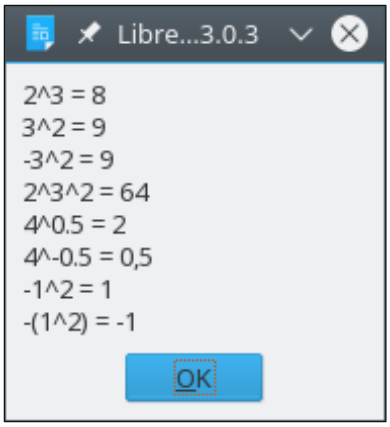


Bild 28. *Der Gebrauch des Potenzierungsoperators.*

Multiplikation (*) und Division (/)

Multiplikation und Division haben dieselbe Priorität.

Listing 35. *Demonstration der Multiplikation und der Division.*

```
Sub ExampleMultDiv
  Print "2*3 = " & 2*3        REM 6
  Print "4/2.0 = " & 4/2.0    REM 2
  Print "-3/2 = " & -3/2      REM -1.5
  Print "4*3/2 = " & 4*3/2    REM 6
End Sub
```

Rest nach Division (Mod)

Der Operator Mod wird auch „Rest nach Division“ genannt. Zum Beispiel hat 5 Mod 2 das Ergebnis 1, weil 5 dividiert durch 2 gleich 2 ist mit einem Rest von 1. Alle Operanden werden zu Ganzzahlen gerundet, bevor die Rechnung durchgeführt wird.

Listing 36. *Definition des Operators Mod für die ganzzahligen Operanden x und y.*

```
x Mod y = x - (y * (x\y))
```

Listing 37. *Demonstration des Operators Mod.*

```
REM x Mod y kann auch so geschrieben werden:
REM CInt(x) - (CInt(y) * (CInt(x)\CInt(y)))
REM CInt ist nötig, weil die Zahlen gerundet werden müssen,
REM bevor die Berechnungen ausgeführt werden.
Sub ExampleMod
  Dim x(), y(), s$, i%
  x() = Array( 4, 15, 6, 6.4, 6.5, -15, 15, -15)
  y() = Array(15,  6, 3,   3,   3,   8, -8,  -8)
  For i = LBound(x()) To UBound(x())
    s = s & x(i) & " Mod " & y(i) & " = " & (x(i) Mod y(i)) & Chr$(10)
  Next
  MsgBox s, 0, "Operator Mod"
End Sub
```

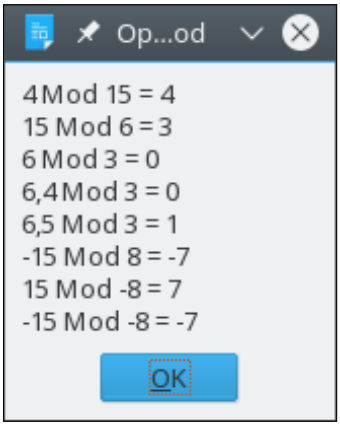


Bild 29. *Der Gebrauch des Operators Mod.*

Tipp	Die Operanden für Mod werden vor der Division zu ganzzahligen Werten gerundet.

Ganzzahlige Division (\)

Bei einer normalen Division werden Operanden vom Typ Double erwartet, und das Ergebnis ist wiederum ein Double. Zum Beispiel ergibt 7.0 / 4.0 den Wert 1.75. Ganzzahlige Division andererseits erwartet die Division zweier Integer-Werte, und das Ergebnis wird auch ein Integer sein. Im Beispiel 7.2 \ 4.3 werden die Operanden zu 7 \ 4 konvertiert, und als Ergebnis kommt 1 heraus. Konstante numerische Operanden des Operators der ganzzahligen Division werden zu Integer-Werten beschnitten, bevor die ganzzahlige Division erfolgt. Wenn aber in einem Operanden Variablen vorkommen, wird das Resultat des Operanden gerundet. Das Ergebnis der Division ist ein beschnittener, nicht gerundeter Wert. Listing 38 zeigt den Unterschied zwischen einer ganzzahligen und einer normalen Division.

Listing 38. *Demonstration einer ganzzahligen Division.*

```
Sub ExampleIntDiv
  Dim f As Double
  Dim s$
  f = 5.9
  s = "5/2 = " & 5/2                      REM 2.5
  s = s & Chr$(10) & "5\2 = " & 5\2       REM 2
  s = s & Chr$(10) & "5/3 = " & 5/3       REM 1.666666667
  s = s & Chr$(10) & "5\3 = " & 5\3       REM 1
  s = s & Chr$(10) & "5/4 = " & 5/4       REM 1.25
  s = s & Chr$(10) & "5\4 = " & 5\4       REM 1
  s = s & Chr$(10) & "-5/2 = " & -5/2     REM -2.5
  s = s & Chr$(10) & "-5\2 = " & -5\2     REM -2
  s = s & Chr$(10) & "-5/3 = " & -5/3     REM -1.666666667
  s = s & Chr$(10) & "-5\3 = " & -5\3     REM -1
  s = s & Chr$(10) & "-5/4 = " & -5/4     REM -1.25
  s = s & Chr$(10) & "-5\4 = " & -5\4     REM -1
  s = s & Chr$(10) & "17/6 = "   & 17/6   REM 2.83333333333333
  s = s & Chr$(10) & "17\6 = "   & 17\6   REM 2
  s = s & Chr$(10) & "17/5.9 = " & 17/5.9 REM 2.88135593220339
  s = s & Chr$(10) & "17\5 = " & 17\5     REM 3
  s = s & Chr$(10) & "17\5.9 = " & 17\5.9 REM 3, weil 5.9 zu 5 beschnitten wurde.
  s = s & Chr$(10) & "17\f = " & 17\f     REM 2, weil f zu 6 aufgerundet wurde.
  s = s & Chr$(10) & "17\(11.9/2) = " & 17\(11.9/2) REM 3, weil 11.9/2 zu 5
                                                    REM beschnitten wurde.
```

```
    MsgBox s
End Sub
```

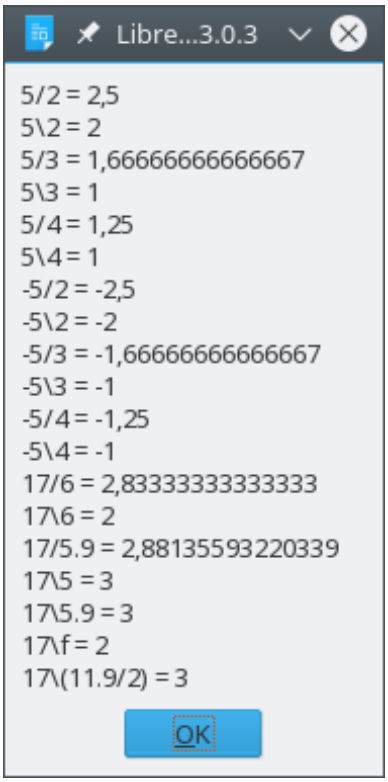


Bild 30. *Ganzzahlige Division.*

Addition (+), Subtraktion (-) und String-Verkettung (& und +)

Addition und Subtraktion haben dieselbe Priorität, die höher ist als der Operator zur String-Verkettung. Beim Addieren numerischer Werte ist Vorsicht geboten, denn der Operator + kann auch zur String-Verkettung genutzt werden. Wenn der erste Operand des Operators + eine Zahl ist und der zweite ein String, wird der String zu einer Zahl konvertiert. Wenn der erste Operand des Operators + ein String ist und der zweite eine Zahl, wird die Zahl in einen String konvertiert.

```
Print 123 + "3"       REM 126  (Numerisch)
Print "123" + 3       REM 1233 (String)
```

Der Verkettungsoperator versucht, die Operanden zu Strings zu konvertieren, wenn wenigstens ein Operator ein String ist.

```
Print 123 & "3"      REM 1233 (String)
Print "123" & 3      REM 1233 (String)
Print 123 & 3        REM Um zu funktionieren, muss wenigsten ein String dabei sein!
```

String-Manipulationen und numerische Operationen miteinander zu vermengen, kann zu seltsamen Ergebnissen führen, vor allem, weil die String-Verkettung mit dem Operator & geringere Priorität hat als der Operator +.

```
Print 123 + "3" & 4  '1264  Erst die Addition, dann die Stringkonvertierung
Print 123 & "3" + 4  '12334 Erst die Addition, aber der erste Operand ist ein String
Print 123 & 3 + "4"  '1237  Erst die Addition, aber der erste Operand ist ein Integer
```

3.8.2. Logische und bitweise Operatoren

Jeder logische Operator stellt eine einfache Frage und gibt eine Antwort zurück, die True (wahr) oder False (falsch) ist. Ist es zum Beispiel wahr, dass (Sie Geld haben) And (Sie mein Buch kaufen wollen)? Derartige Operationen sind einfach und werden häufig in Basic benötigt. Erheblich seltener genutzt, und hier nur der Vollständigkeit halber zur Beglückung der Computerprofis erwähnt, sind die

bitweisen Operatoren. Bitweise arbeitende Operatoren sind nicht schwierig, aber wenn Sie sie nicht verstehen, so wird Ihr Umgang mit Basic höchstwahrscheinlich gar nicht beeinträchtigt.

Ein logischer Operator wird im allgemeinen mit den Werten True und False in Verbindung gebracht. In Basic führen logische Operatoren aber auch bitweise Operationen auf Integer-Werte aus. Das bedeutet, dass jedes Bit des ersten Operanden mit dem entsprechenden Bit des zweiten Operanden verglichen wird, um dann das entsprechende Bit im Ergebnis zu setzen. Zum Beispiel wird bei den binären Operanden 01 und 10 von der 01 die 0 und von der 10 die 1 für das erste Bit des Resultats verwendet.

Ungewöhnlich bei den logischen und bitweisen binären Operatoren in Basic ist, dass die Priorität dieselbe ist. In anderen Sprachen hat And normalerweise größere Priorität als Or.

Tabelle 17 listet die logischen und bitweisen Operatoren auf, die von OOo unterstützt werden. True und False stehen für logische Werte, 0 und 1 für Bitwerte.

Tabelle 17. *Wahrheitstabelle für logische und bitweise Operatoren.*

x	y	x And y	x Or y	x Xor y	x Eqv y	x Imp y
True	True	True	True	False	True	True
True	False	False	True	True	False	False
False	True	False	True	True	False	True
False	False	False	False	False	True	True
1100	1010	1000	1110	0110	1001	1011

Intern werden die Operanden der logischen Operatoren zum Typ Long konvertiert. Dabei kommt es bei einem Fließkomma-Operanden zu dem unerwarteten Nebeneffekt, dass auch er zu Long konvertiert wird, was einen numerischen Überlauf zur Folge haben kann. Die Konvertierung von einer Fließkommazahl zu einer Ganzzahl vom Typ Long geht mit Runden, nicht mit Beschneiden des Wertes einher. Das funktioniert, weil der Wert für True -1 ist und der für False 0. Doch mit zwei Boolean-Operanden ist der Rückgabewert manchmal noch vom Typ Long.

Listing 39. *Logische Operanden sind ganzzahlig vom Typ Long.*

```
Sub LogicalOperandsAreLongs
  Dim v, b1 As Boolean, b2 As Boolean
  b1 = True : b2 = False
  v = (b1 Or b2)
  Print TypeName(v)    REM Long, weil Operanden zu Long konvertiert werden.
  Print v              REM -1,   weil der Rückgabetyp Long ist.
  Print (b2 Or "-1")   REM -1,   weil "-1" zu Long konvertiert wird.
End Sub
```

Bei manchen logischen Ausdrücken müssten eigentlich nicht alle Operanden ausgewertet werden. Zum Beispiel wird der Ausdruck (False And True) schon beim Operator And nach dem ersten Operanden als False erkannt. So etwas kennt man als Kurzschluss-Auswertung. Leider leider gibt es das nicht in Basic, es werden alle Operanden ausgewertet.

Tipp Basic unterstützt keine Kurzschluss-Auswertung, somit bewirkt (x <> 0 And y/x > 3) den Fehler „Division durch null“, falls x gleich null ist.

Die bitweisen Operatoren werden alle auf dieselbe Art veranschaulicht. Zwei Arrays werden mit booleschen Werten gefüllt, dazu kommen zwei Integer-Variablen mit ganzzahligen Werten.

```
xi% = 12 : yi% = 10
x() = Array(True, True, False, False)
y() = Array(True, False, True, False)
```

Die dezimale Zahl 12 wird auf der Basis 2 als 1100 repräsentiert, in Übereinstimmung mit den Werten in x(). Die dezimale Zahl 10 wird auf der Basis 2 als 1010 repräsentiert, in Übereinstimmung mit den Werten in y(). Der Operator wird dann nacheinander angewendet auf „x(0) op y(0)“, „x(1) op y(1)“, „x(2) op y(2)“, „x(3) op y(3)“ und „xi op yi“. Das Resultat wird in einer Meldung ausgegeben. Die Ganzzahlen werden auf der Basis 2 dargestellt, um die bitweise Operation zu verdeutlichen. Listing 40 demonstriert, wie ein Integer-Wert in eine Reihe von Bits konvertiert wird. Hierbei werden viele Techniken angewendet, die später in diesem Kapitel behandelt werden.

Listing 40. *Konvertierung eines Integer-Wertes in eine Binärzahl.*

```
Sub TestIntToBinary
  Dim s$
  Dim n%
  Dim x%
  x = InputBox("Bitte geben Sie eine ganze Zahl ein")
  If x <> 0 Then
    n = Log(Abs(x)) / Log(2) + 1
    If (x < 0) Then
      n = n + 4
    End If
  Else
    n = 1
  End If
  Print "s = " & IntToBinaryString(x, n)
End Sub

REM Konvertierung eines Integer-Wertes in einen String der Bits
REM x ist die zu konvertierende Ganzzahl
REM n ist die Anzahl der zu konvertierenden Bits
REM Es wäre leichter, wenn ich das niedrigste Bit herausschieben (shift)
REM und gleichzeitig das Vorzeichenbit der Zahl erhalten könnte, aber das geht nicht.
REM Ich bilde es dadurch nach, dass ich durch zwei dividiere, was aber bei negativen
REM Zahlen fehlschlägt. Um dieses Problem zu vermeiden, drehe ich alle Bits um,
REM wenn die Zahl negativ ist, und mache dadurch aus ihr eine positive Zahl.
REM Dann baue ich ein invertiertes Resultat.
Function IntToBinaryString(ByVal x%, ByVal n%) As String
  Dim b1$                 'Bit 0 (ganz rechts) für positive Zahlen
  Dim b0$                 'Bit 1 für positive Zahlen
  If (x >= 0) Then        'Nicht negativ, also funktioniert der Algorithmus
    b1 = "1" : b0 = "0"   'mit den Standard-Bitwerten.
  Else                    'Eine negative Zahl, also
    x = Not x             'werden alle Bits umgedreht,
    b1 = "0" : b0 = "1"   'auch die Standard-Bitwerte.
  End If
  Dim s$                  'Die Bits werden im String s$ aneinandergereiht
  Do While n > 0          'n ist die zurückzugebende Anzahl an Bits
    If (x And 1) = 1 Then 'And mit 1, um Bit 0 (ganz rechts) zu finden
      s = b1$ & s         'Bit 1 ist gesetzt, es wird 1 hinzugefügt (0 bei negativem x)
    Else
      s = b0$ & s         'Bit 1 ist frei, also wird 0 hinzugefügt (1 bei negativem x)
    End If
    x = x\2               'Ganzzahldivision durch 2
    n = n - 1             'Vermindere n um 1 für jedes erledigte Bit.
  Loop                    'Zurück zum Start der While-Schleife
```

```
  IntToBinaryString = s    'Zuweisung des Rückgabewertes
End Function
```

And

Operiert als logisches And auf booleschen Werten und als bitweises And auf numerischen Werten. Nehmen Sie den Satz „Sie können ins Kino gehen, (wenn Sie Geld haben) And (wenn Sie über ein Fahrzeug verfügen)“. Beide Bedingungen müssen wahr sein, bevor Sie ins Kino gehen können. Wenn beide Operanden True sind, ist das Ergebnis True, andernfalls ist das Ergebnis False.

Listing 41. *Operator And.*

```
Sub ExampleOpAnd
  Dim s$, x(), y(), i%, xi%, yi%
  xi% = 12 : yi% = 10
  x() = Array(True, True, False, False)
  y() = Array(True, False, True, False)
  For i = LBound(x()) To UBound(x())
    s = s & x(i) & " And " & y(i) & " = " & CBool(x(i) And y(i)) & Chr$(10)
  Next
  s = s & IntToBinaryString(xi, 4) & " And " & IntToBinaryString(yi, 4) &_
     " = " & IntToBinaryString(xi And yi, 4) & Chr$(10)
  MsgBox s, 0, "Beispiel für Operator And"
End Sub
```

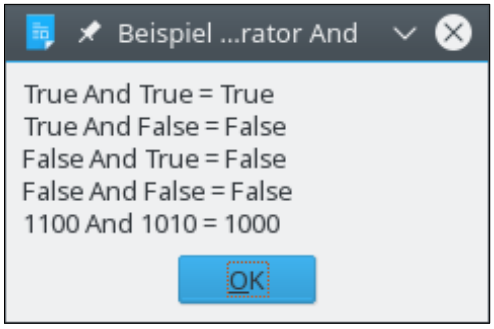


Bild 31. *Der Gebrauch des Operators And.*

Or

Operiert als logisches Or auf booleschen Werten und als bitweises Or auf numerischen Werten. Nehmen Sie den Satz „Sie können den Kauf tätigen, (wenn Sie das Geld haben) Or (wenn Ihr Freund das Geld hat)“. Es spielt keine Rolle, wer das Geld hat. Wenn einer von beiden Operanden True ist, ist das Ergebnis True, andernfalls ist das Ergebnis False.

Listing 42. *Operator Or.*

```
Sub ExampleOpOR
  Dim s$, x(), y(), i%, xi%, yi%
  xi% = 12 : yi% = 10
  x() = Array(True, True, False, False)
  y() = Array(True, False, True, False)
  For i = LBound(x()) To UBound(x())
    s = s & x(i) & " Or " & y(i) & " = " & CBool(x(i) Or y(i)) & Chr$(10)
  Next
  s = s & IntToBinaryString(xi, 4) & " Or " & IntToBinaryString(yi, 4) &_
     " = " & IntToBinaryString(xi Or yi, 4) & Chr$(10)
  MsgBox s, 0, "Beispiel für Operator Or"
End Sub
```

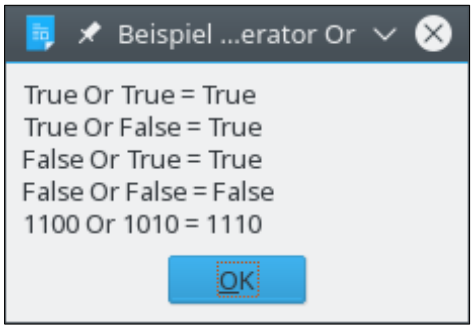


Bild 32. *Der Gebrauch des Operators Or.*

Xor

Der Operator Xor wird auch „exklusives Oder“ genannt. Er fragt nach Antivalenz oder Ungleichwertigkeit. Das Ergebnis ist True, wenn die Operanden unterschiedliche Werte haben. Das Ergebnis ist False, wenn beide Operanden denselben Wert haben. Das logische Xor operiert auf booleschen Werten und das bitweise Xor auf numerischen Werten.

Listing 43. *Operator Xor.*

```
Sub ExampleOpXOR
  Dim s$, x(), y(), i%, xi%, yi%
  xi% = 12 : yi% = 10
  x() = Array(True, True, False, False)
  y() = Array(True, False, True, False)
  For i = LBound(x()) To UBound(x())
    s = s & x(i) & " Xor " & y(i) & " = " & CBool(x(i) Xor y(i)) & Chr$(10)
  Next
  s = s & IntToBinaryString(xi, 4) & " Xor " & IntToBinaryString(yi, 4) &_
     " = " & IntToBinaryString(xi Xor yi, 4) & Chr$(10)
  MsgBox s, 0, "Beispiel für Operator Xor"
End Sub
```

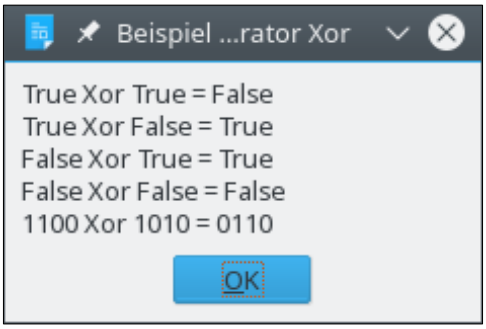


Bild 33. *Der Gebrauch des Operators Xor.*

Eqv

Der Operator Eqv fragt nach Äquivalenz oder Gleichwertigkeit. Sind beide Operanden gleich? Das logische Eqv operiert auf booleschen Werten und das bitweise Eqv auf numerischen Werten. Wenn beide Operanden denselben Wert haben, ist das Ergebnis True. Wenn die Operanden nicht denselben Wert haben, ist das Ergebnis False.

Listing 44. *Operator Eqv.*

```
Sub ExampleOpEqv
  Dim s$, x(), y(), i%, xi%, yi%
  xi% = 12 : yi% = 10
  x() = Array(True, True, False, False)
  y() = Array(True, False, True, False)
```

```
  For i = LBound(x()) To UBound(x())
    s = s & x(i) & " Eqv " & y(i) & " = " & CBool(x(i) Eqv y(i)) & Chr$(10)
  Next
  s = s & IntToBinaryString(xi, 4) & " Eqv " & IntToBinaryString(yi, 4) &_
     " = " & IntToBinaryString(xi Eqv yi, 4) & Chr$(10)
  MsgBox s, 0, "Beispiel für Operator Eqv"
End Sub
```

Bild 34. *Der Gebrauch des Operators Eqv.*

Imp

Der Operator Imp führt eine logische Schlussfolgerung durch, eine Implikation. Das logische Imp operiert auf booleschen Werten und das bitweise Imp auf numerischen Werten. Wie der Name schon aussagt, fragt „x Imp y“, ob die Aussage„x impliziert y“ wahr ist. Zum Verständnis einer logischen Implikation definieren Sie x und y folgendermaßen:

```
x = Der Himmel ist wolkig
y = Die Sonne ist nicht zu sehen
If x Then y
```

Wenn sowohl x als auch y wahr sind – der Himmel ist wolkig, und die Sonne ist nicht zu sehen –, kann die Aussage als wahr betrachtet werden. Die Aussage sagt nichts über y aus, wenn x nicht wahr ist. Mit anderen Worten, wenn der Himmel nicht wolkig ist, impliziert die Aussage nicht, dass die Sonne zu sehen oder nicht zu sehen ist. Es könnte zum Beispiel eine klare Nacht sein, oder Sie könnten (wie ein guter Computer-Fex) sich in einem Zimmer ohne Fenster befinden. Daraus folgt, dass die gesamte Aussage immer als wahr gewertet wird, wenn x falsch ist. Wenn schließlich x wahr ist und y nicht, wird die gesamte Aussage als falsch gewertet. Wenn der Himmel wolkig ist und die Sonne zu sehen ist, kann die Aussage unmöglich korrekt sein. Ein wolkiger Tag könnte nicht implizieren, dass die Sonne sichtbar ist.

Listing 45. *Operator Imp.*

```
Sub ExampleOpImp
  Dim s$, x(), y(), i%, xi%, yi%
  xi% = 12 : yi% = 10
  x() = Array(True, True, False, False)
  y() = Array(True, False, True, False)
  For i = LBound(x()) To UBound(x())
    s = s & x(i) & " Imp " & y(i) & " = " & CBool(x(i) Imp y(i)) & Chr$(10)
  Next
  s = s & IntToBinaryString(xi, 4) & " Imp " & IntToBinaryString(yi, 4) &_
     " = " & IntToBinaryString(xi Imp yi, 4) & Chr$(10)
  MsgBox s, 0, "Beispiel für Operator Imp"
End Sub
```

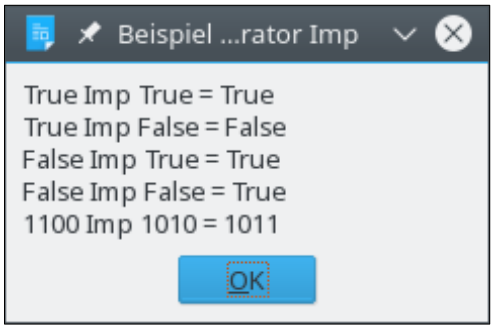


Bild 35. *Der Gebrauch des Operators Imp.*

Not

Das logische Not operiert auf booleschen Werten und das bitweise Not auf numerischen Werten. Das heißt, dass „Not True“ False ist und „Not False“ True ist. Bei bitweisen Operationen wird eine 1 zu 0 und eine 0 zu 1.

```
Print Not True    REM 0, das False ist
Print Not False   REM -1, das True ist
Print Not 2       REM -3, die Bits 0010 wurden zu 1101
```

Shift-Operationen

Einfache Operationen zum Verschieben der Bits eines Bytes nach links oder nach rechts, wie man sie aus anderen Programmiersprachen kennt, sind in Basic nicht direkt möglich, denn es fehlen die nötigen Operatoren, zum Beispiel die aus den Programmiersprachen C und Python bekannten Shift-Operatoren „<<“ und „>>“. Zum Beispiel ein Linksshift um 1 Stelle „00110011 << 1“ ergibt 01100110

Ein Linksshift um n Stellen ist gleichbedeutend mit der Multiplikation einer Binärzahl mit 2^n, ein Rechtsshift um n Stellen gleichbedeutend mit der Division einer Binärzahl durch 2^n. Die links oder rechts frei werdenden Bits werden mit 0 besetzt (aber Achtung bei negativen Zahlwerten). In jedem Fall kann es sein, dass links oder rechts gesetzte Bits (Wert = 1) aus dem Byte „herausgeschoben“ werden.

Die Grundlagen der binären Zahlendarstellung sind am Anfang dieses Abschnitts 3.8.2 beschrieben, mit einem ausführlichen Beispiel im Listing 40, in dem auch die in der Praxis zu beachtenden Probleme mit negativen Zahlen behandelt sind.

Beispiel für einen Shift nach links als Multiplikation:

```
Dim i%, s%, r%
i = 1
s = 8
r = i * 2^s       ' 256 = 2 hoch 8, also Linksshift um 8 Stellen, r = 256
```

Es gibt noch weitere Möglichkeiten, einen Shift zu erreichen. Das folgende Beispiel zeigt einen Weg über Maskieren und Divisionsrest für einen Shift um 8 Stellen nach rechts:

```
Dim i As Integer
Dim r As Integer
i = 256             '&H100
i = i And &HFF00    'linkes Byte = 1, rechtes Byte = 0
r = i Mod &FF       'Rechtsshift um 8 Stellen, r = 1
```

3.8.3. Vergleichsoperatoren

Die Vergleichsoperatoren funktionieren mit numerischen Datentypen, sowie mit den Typen Date, Boolean und String.

```
Print 2  = 8/4 And 0 <  1/3 And 2  > 1    '-1=True
Print 4 <= 4.0 And 1 >=   0 And 1 <> 0    '-1=True
```

Stringvergleiche basieren darauf, dass Strings intern als Zahlen vorliegen, und unterscheiden Groß- und Kleinschreibung. Der Buchstabe „A“ ist kleiner als der Buchstabe „B“. Alle Großbuchstaben sind kleiner als die Kleinbuchstaben.

```
Sub OKCompare
  Dim a$, b$, c$
  a$ = "A" : b$ = "B" : c$ = "B"
  Print a$  < b$   'True
  Print b$  = c$   'True
  Print c$ <= a$   'False
End Sub
```

Es kann zu seltsamen Problemen führen, wenn alle Operanden String-Konstanten sind. Wenn wenigstens ein Operand eine Variable ist, gibt es das erwartete Ergebnis. Das hat wahrscheinlich mit der Art und Weise zu tun, wie Operanden erkannt und zum Einsatz konvertiert werden.

```
Sub BadCompare
  Print "A" < "B"     '0=False, das ist nicht korrekt
  Print "B" < "A"     '-1=True, das ist nicht korrekt
  Print  3  = "3"     'False, aber das ändert sich bei der Verwendung einer Variablen
End Sub
```

Werden Variablen an Stelle von String-Konstanten verwendet, werden die numerischen Wert für den Vergleich zum Typ String konvertiert.

```
Sub OKCompare_2
  Dim a$, i%, t$
  a$ = "A" : t$ = "3" : i% = 3
  Print  a$ < "B"     'True,  String-Vergleich
  Print "B" <  a$     'False, String-Vergleich
  Print  i% = "3"     'True,  String-Vergleich
  Print  i% = "+3"    'False, String-Vergleich
  Print  3  =  t$     'True,  String-Vergleich
  Print  i% < "2"     'False, String-Vergleich
  Print  i% > "22"    'True,  String-Vergleich
End Sub
```

Tipp Wenn man Operanden unterschiedlichen Typs vergleicht, vor allem in der Konstellation numerischer Typ und String, ist eine explizite Tpykonvertierung sicherer. Konvertieren Sie entweder den String zu einer Zahl oder die Zahl zum String. Die dafür vorgesehenen Funktionen sehen Sie an späterer Stelle.

OOo erkennt die Visual-Basic-Anweisung Option Compare {Binary | Text}, aber die Anweisung bewirkt nichts, jedenfalls in OOo 3.20. Das aktuelle Verhalten ist der binäre Vergleich von Strings.

3.9. Ablaufsteuerung

Ablaufsteuerung handelt davon, welche Codezeile als nächste ausgeführt wird. Der Aufruf einer Subroutine oder einer Funktion ist eine einfache Form nicht bedingter Ablaufsteuerung. Zu komplexeren Ablaufsteuerungen gehören Verzweigungen und Schleifen. Ablaufsteuerung ermöglicht verwickelte Abläufe in Makros, die sich je nach aktueller Datenlage ändern.

Verzweigungsanweisungen bewirken eine Änderung des Programmflusses. Der Aufruf einer Subroutine oder einer Funktion ist eine nicht bedingte Verzweigung. Basic unterstützt bedingte Verzweigungsanweisungen wie „wenn x, dann tu y“. Schleifenanweisungen bewirken, dass das Programm Codebereiche wiederholt. Mit Schleifenanweisungen kann ein Bereich bestimmte Male wiederholt werden oder bis eine spezifische „exit“-Bedingung erreicht wird.

3.9.1. Definition eines Labels als Sprungmarke

Einige Anweisungen zur Ablaufsteuerung wie GoSub, GoTo und On Error benötigen eine markierte Stelle im Code, ein Label. Label-Namen gehorchen denselben Regeln wie Namen von Variablen. Auf einen Label-Namen folgt direkt ein Doppelpunkt. Sie erinnern sich, dass ein Doppelpunkt auch als Trenner von Anweisungen dient, um mehrere Anweisungen auf einer Zeile zu ermöglichen. Ein Leerzeichen zwischen Label und Doppelpunkt würde den Doppelpunkt als Anweisungstrenner ausweisen, wodurch das Label nicht definiert wäre. Die folgenden Zeilen sind gültiger Basic-Code.

```
  <Anweisungen>
  i% = 5 : z = q + 4.77

MyCoolLabel:
  <weitere Anweisungen>

JumpTarget: <weitere Anweisungen> REM Zwischen Label und Doppelpunkt ist kein Leerraum
```

Achtung Ein zwischen Label und Doppelpunkt eingefügtes Leerzeichen macht aus dem Doppelpunkt einen Anweisungstrenner, und das Label ist nicht definiert.

3.9.2. GoSub, GoTo, On GoSub und OnGoTo

Diese Anweisungen sind hartnäckige Überbleibsel von alten BASIC-Dialekten, beibehalten aus Gründen der Kompatibilität. Deren Verwendung von GoSub wird dringend abgeraten, weil es dazu verführt, nicht lesbaren Code zu produzieren. Schreiben Sie stattdessen eine Subroutine oder eine Funktion. Überdies unterstützt Visual Basic .NET das Schlüsselwort GoSub nicht mehr.

Denken Sie auch an Ihre Reputation: die Anweisungen sind verpönt. Sie werden hier nur aus Gründen der Vollständigkeit erläutert.

GoSub

Die Anweisung GoSub bewirkt, dass das Programm an einem definierten Label innerhalb der aktuellen Routine fortgesetzt wird. Ein Sprung an eine Stelle außerhalb der Routine ist nicht möglich. Beim Erreichen der Anweisung Return wird das Programm an der Stelle der GoSub-Anweisung fortgesetzt. Eine Return-Anweisung ohne vorheriges GoSub produziert einen Laufzeitfehler. Mit anderen Worten, Return ist kein Ersatz für Exit Sub oder Exit Function.

Listing 46. *Beispiel für GoSub.*

```
Sub ExampleGoSub
  Dim i As Integer
  GoSub Line2       REM Springt zu Line2 und kehrt zurück, i ist dann 1.
  GoSub [Line 1]    REM Springt zu [Line 1] und kehrt zurück, i ist dann 2.
  MsgBox "i = " + i, 0, "Beispiel für GoSub"  REM i ist nun 2
  Exit Sub          REM Beendet die aktuelle Subroutine.
[Line 1]:           REM Dieses Label enthält ein Leerzeichen.
  i = i + 1         REM Addiert 1 zu i.
  Return            REM Kehrt zur aufrufenden Stelle zurück.
Line2:              REM Dies ist ein typischeres Label, kein Leerzeichen.
  i = 1             REM Setzt i zu 1.
  Return            REM Kehrt zur aufrufenden Stelle zurück.
End Sub
```

GoTo

Die Anweisung GoTo bewirkt, dass das Programm an einem definierten Label innerhalb der aktuellen Routine fortgesetzt wird. Ein Sprung an eine Stelle außerhalb der Routine ist nicht möglich. Im Gegensatz zur Anweisung GoSub weiß GoTo nicht, von wo es aufgerufen wird.

Listing 47. *Beispiel für GoTo.*

```
Sub ExampleGoTo
  Dim i As Integer
  GoTo Line2          REM Okay, das sieht leicht genug aus,
Line1:                REM aber jetzt gerate ich doch ein wenig durcheinander.
  i = i + 1           REM Ich wünschte, ich hätte GoTo nicht verwendet.
  GoTo TheEnd         REM Das ist doch verrückt, das sieht irgendwie nach Spaghetti aus,
Line2:                REM ein Gewirr von losen Fäden: Spaghetti-Code.
  i = 1               REM Falls Sie es so machen mussten, dann
  GoTo Line1          REM haben Sie wohl etwas schlecht gemacht.
TheEnd:               REM Verwenden Sie kein GoTo.
  MsgBox "i = " + i, 0, "Beispiel für GoTo"
End Sub
```

On GoTo und On GoSub

Mit diesen Anweisungen wird das Programm abhängig von einem numerischen Ausdruck N zu einem Label verzweigt. Wenn N null ist, wird nicht verzweigt. Der numerische Ausdruck N muss im Bereich von 0 bis 255 liegen. Manchmal wird das als „berechnetes GoTo“ bezeichnet, weil eine Berechnung als Grundlage für den Programmfluss dient. Ein Sprung an eine Stelle außerhalb der Routine ist nicht möglich.

```
Syntax:  On N GoSub Label1[, Label2[, Label3[,...]]]
Syntax:  On N GoTo  Label1[, Label2[, Label3[,...]]]
```

In der Aufzählung wird die Arbeitsweise deutlich: wenn N = 1 ist, dann verzweige zum Label 1, wenn N = 2 ist, dann verzweige zu Label 2 ... Wenn N kleiner als 1 oder größer als die Anzahl der Labels ist, dann wird gar nicht verzweigt, die Anweisung wird einfach ignoriert.

Listing 48. *Beispiel für On GoTo.*

```
Sub ExampleOnGoTo
  Dim i As Integer
  Dim s As String
  i = 1
  On i+1 GoSub Sub1, Sub2
  s = s & Chr(13)
  On i GoTo Line1, Line2
  REM Mit Exit beenden wir die Subroutine, wenn wir den Rest nicht ausführen wollen.
  Exit Sub
Sub1:
  s = s & "In Sub 1" : Return
Sub2:
  s = s & "In Sub 2" : Return
Line1:
  s = s & "Am Label 1" : GoTo TheEnd
Line2:
  s = s & "Am Label 2"
TheEnd:
  MsgBox s, 0, "Beispiel für On GoTo"
End Sub
```

3.9.3. If Then Else

Mit der If-Konstruktion wird abhängig von einem Ausdruck ein Codeblock ausgeführt. Auch wenn man mit GoTo oder GoSub aus einem If-Block heraus springen kann, so kann man doch nicht in einen If-Block hinein springen. Die einfachste If-Anweisung hat die folgende Form:

```
If Bedingung Then Anweisung
```

Wenn mehrere Anweisungen benötigt werden, muss die Form aus mehreren Zeilen bestehen und mit der Zeile „End If“ (oder „EndIf“) abgeschlossen werden. Diese Form ist zur besseren Lesbarkeit natürlich auch mit nur einer Anweisung möglich:

```
If Bedingung Then
  Anweisung1
  [Anweisung2]
End If
```

Die Bedingung kann aus jedem Ausdruck bestehen, der entweder True oder False ergibt – oder in diese Werte konvertierbar ist. Mit einer etwas komplexeren Version können Sie mehr als eine einzige Bedingung abfragen.

```
If Bedingung Then
  Anweisungsblock
[ElseIf Bedingung Then]
  Anweisungsblock
[Else]
  Anweisungsblock
End If
```

Wenn die erste Bedingung True ergibt, läuft der erste Block. Die Anweisung ElseIf erlaubt mehrere If-Anweisungen für eine Abfolge von Tests. Der Anweisungsblock für die erste wahre Bedingung läuft. Wenn keine andere Bedingung True ergibt, läuft der Else-Block.

Listing 49. *Beispiel für If.*

```
Sub ExampleIf
  Dim i%
  i% = 4
  If i = 4 Then Print "i ist vier"
  If i <> 3 Then
    Print "i ist nicht drei"
  End If
  If i < 1 Then
    Print "i ist kleiner als 1"
  ElseIf i = 1 Then
    Print "i ist 1"
  ElseIf i = 2 Then
    Print "i ist 2"
  Else
    Print "i ist größer als 2"
  End If
End Sub
```

If-Anweisungen können verschachtelt sein.

```
If i <> 3 THEN
  If k = 4 Then Print "k ist vier"
  If j = 7 Then
     Print "j ist sieben"
  End If
End If
```

3.9.4. IIf

Die Funktion IIf („Immediate If“ = unmittelbares If) gibt abhängig von einer Bedingung einen von zwei Werten zurück.

```
Syntax:  object = IIf (Bedingung, AusdruckWennWahr, AusdruckWennFalsch)
```

Die Funktion benötigt drei Argumente. Das erste ist ein boolescher Wert, der bestimmt, welches der beiden weiteren Argumente zurückgegeben wird. Listing 50 zeigt, wie Sie sich eine eigene IIf-Funktion schreiben könnten, wenn es IIf nicht gäbe.

Listing 50. *Die Funktion IIf, wenn Sie sie selbst schrieben.*

```
Function myIIf(conditional, true_arg, false_arg) As Variant
  If CBool(conditional) Then
    myIIf = true_arg
  Else
    myIIf = false_arg
  End If
End Function
```

IIf ist de facto eine wunderbare einzeilige If-Then-Else-Anweisung.

```
max_alter = IIf(johns_alter > bills_alter, johns_alter, bills_alter)
```

Als Argumente können Sie einfache Datentypen, aber auch Funktionen verwenden, deren Rückgabewerte dann von IIf zur Rückgabe herangezogen werden.

Alle Argumente einer Routine werden vor der Ausführung ausgewertet. Weil auch IIf eine Funktion ist, werden zuerst alle ihre Argumente ausgewertet, das heißt, dass auch alle als Argumente verwendeten Funktionen ausgeführt werden, im Gegensatz zu einer If-Anweisung, bei der der von der Bedingung abhängige Code erst ausgeführt wird, wenn die Bedingung erfüllt ist, s. Listing 51.

Listing 51. *Wenn der Nenner gleich null ist, wird nicht dividiert.*

```
If denominator <> 0 Then              ' Der Nenner eines Bruches
  result = numerator / denominator    ' Zähler / Nenner
Else
  result = 0
End If
```

Wenn im Listing 51 der Nenner gleich null ist, wird nicht dividiert, sondern stattdessen null zurückgegeben. Sollten Ihre Mathematikkenntnisse ein wenig eingerostet sein: Es ist nicht erlaubt, eine Zahl durch null zu dividieren, ein Laufzeitfehler ist die Folge. Andererseits wissen Sie auch – vorausgesetzt, Ihre Mathematikkenntnisse sind doch nicht eingerostet –, dass es nicht wirklich korrekt ist, null als Ergebnis zu liefern, wenn der Nenner gleich null ist. Das Makro im Listing 52 zeigt, dass beide Funktionen f1 und f2 aufgerufen werden, obwohl nur der Wert von f2 zurückgegeben wird. Mit anderen Worten, IIF(x <> 0, 1/x, 0) verursacht einen Division-durch-Null-Fehler, wenn x null ist.

Listing 52. *Alle IIF-Argumente werden ausgewertet.*

```
Sub ExampleIIF
  REM Demonstration, dass ALLE Argumente ausgewertet werden.
  REM Die Ausgabe ist:
  REM "Ich befinde mich in der Funktion f1"
  REM " Ich befinde mich in der Funktion f2"
  REM "f2"
  Print IIF(1 > 2, f1(), f2())
End Sub

Function f1$
  Print "Ich befinde mich in der Funktion f1"
  f1 = "f1"
End Function

Function f2$
  Print "Ich befinde mich in der Funktion f2"
```

```
  f2 = "f2"
End Function
```

3.9.5. Choose

Die Funktion Choose ähnelt der Funktion IIF darin, dass sie abhängig vom Wert des ersten Arguments eines der weiteren Argumente zurückgibt. Der Unterschied besteht jedoch darin, dass sie einerseits mehr als zwei Rückgabeargumente haben kann und andererseits das erste Argument kein boolescher Wert ist, sondern ein Index, eine Ganzzahl, die angibt, welches der möglicherweise vielen Argumente zurückzugeben ist.

```
Syntax: obj = Choose(Ausdruck, Select_1[, Select_2, ... [,Select_n]])
```

Choose gibt null zurück, wenn der Index-Ausdruck kleiner ist als 1 oder größer ist als die Anzahl der Auswahlargumente der Liste. Choose gibt „Select_1“ zurück, wenn der Ausdruck 1 ergibt, und „Select_2“, wenn der Ausdruck 2 ergibt. Genauso kann man auch die Listenwerte in einem Array mit der Bereichsuntergrenze 1 speichern und per Index darauf zugreifen. Ein Beispiel:

Listing 53. *Beispiel für die Anweisung Choose.*

```
Sub ExampleChoose
  Dim i%, v
  i% = CStr(InputBox("Eine Zahl von 1 bis 4 (eine negative Zahl ist ein Fehler)"))
  v = Choose(i, "eins", "zwei", "drei", "vier")
  If IsNull(v) Then
    Print "V ist null"
  Else
    Print CStr(v)
  End If
End Sub
```

Auch für Choose gilt, dass jedes Argument ausgewertet wird, unabhängig davon, welches zurückgegeben wird

Listing 54. *Choose: Division-durch-null-Fehler,*
weil vor der Rückgabe [1/(i-2)] alle Argumente ausgewertet werden.

```
i% = 3
Print Choose(i%, 1/(i+1), 1/(i-1), 1/(i-2), 1/(i-3))
```

Auswahlargumente können Ausdrücke sein, auch mit Funktionsaufrufen. Bei der Anweisung Choose wird jede Funktion der Argumentenliste aufgerufen und jeder Ausdruck ausgewertet. Der Code in Listing 54 bewirkt einen Division-durch-null-Fehler, weil jedes Argument, nicht nur das zurückzugebende ausgewertet wird. Listing 55 ruft die Funktionen Choose1, Choose2, und Choose3 auf.

Listing 55. *Beispiel für die Anweisung Choose mit Funktionsaufrufen.*

```
Sub ExampleChooseWithFunctions
  Print Choose(2, "Eins", "Zwei", "Drei")           'Zwei
  Print Choose(2, Choose1(), Choose2(), Choose3())  'Zwei
End Sub

Function Choose1$()
  Print "Ich bin in Choose1"
  Choose1 = "Eins"
End Function

Function Choose2$()
  Print "Ich bin in Choose2"
  Choose2 = "Zwei"
End Function

Function Choose3$()
  Print "Ich bin in Choose3"
```

```
  Choose3 = "Drei"
End Function
```

> **Tipp** In der Anweisung Choose werden alle Argumente ausgewertet. Werden innerhalb der Argumente Funktionen genutzt, werden sie alle ausgeführt.

3.9.6. Select Case

Die Anweisung Select Case ähnelt einer If-Anweisung mit mehreren Else-If-Blöcken. Es wird aber nur ein Bedingungsausdruck definiert, der dann mit mehreren Werten auf Gleichheit geprüft wird:

```
Select Case bedingungs_ausdruck
  Case case_ausdruck1
    Anweisungsblock1
  Case case_ausdruck2
    Anweisungsblock2
  Case Else
    Anweisungsblock3
End Select
```

Der Ausdruck bedingungs_ausdruck wird in jedem Case-Ausdruck verglichen. Der auf den ersten Treffer folgende Block von Anweisungen wird ausgeführt. Der optionale Block Case Else wird ausgeführt, wenn keine Bedingung zutrifft. Es ist kein Fehler, wenn nichts zutrifft und kein Case Else definiert ist. Dann gibt es einfach nichts zu tun.

Case-Ausdrücke

Der Bedingungsausdruck wird nur einmal ausgewertet und dann der Reihe nach mit jedem Case-Ausdruck verglichen, bis ein Treffer erfolgt. Ein Case-Ausdruck ist häufig nur eine Konstante wie „Case 4“ oder „Case "Hallo"“.

```
Select Case 2
  Case 1
    Print "Eins"
  Case 3
    Print "Drei"
End Select
```

Sie können mehrere Werte gleichzeitig angeben, wenn Sie sie mit Kommas trennen: „`Case 3, 5, 7`“. Das Schlüsselwort `To` kontrolliert einen Wertebereich – zum Beispiel „`Case 5 To 10`“. Bereiche mit einem offenen Ende werden als „`Case < 10`“ oder als „`Case Is < 10`“ kontrolliert.

> **Tipp** Die Anweisung Case Is hat nichts mit dem Operator Is zu tun, der prüft, ob zwei Objekte gleich sind.

Jede Case-Anweisung in der Form „Case op Ausdruck“ ist eine Kurzform für „Case Is op Ausdruck“. Die Form „Case Ausdruck“ ist die Kurzform von „`Case Is = Ausdruck`“. Zum Beispiel ist „`Case >= 5`“ äquivalent zu „`Case Is >= 5`“, und „`Case 1+3`“ ist äquivalent zu „`Case Is = 1+3`“.

```
Select Case i
  Case 1, 3, 5
    Print "i ist eins , drei oder fünf"
  Case 6 To 10
    Print "i ist ein Wert von 6 bis 10"
  Case < -10
    Print "i ist kleiner als -10"
  Case Is > 10
```

```
    Print "i ist größer als 10"
  Case Else
    Print "Keine Ahnung, was i ist"
End Select
```

Eine Case-Anweisung kann, durch Kommas getrennt, eine Liste von Ausdrücken enthalten. Jeder Ausdruck kann einen zu einer Seite offenen Bereich einschließen. Jeder Ausdruck kann die Anweisung Case Is nutzen (s. Listing 56).

Listing 56. *Select Case: das Schlüsselwort Is ist optional.*

```
Select Case i%
  Case 6, Is = 7, Is = 8, Is > 15, Is < 0
    Print "" & i & " passt"
  Case Else
    Print "" & i & " ist außerhalb des Bereichs"
End Select
```

Wenn Case-Anweisungen so einfach sind, warum sind sie so oft fehlerhaft?

Ich sehe häufig fehlerhafte Beispiele von Case-Anweisungen. Es ist sehr lehrreich, sich anzusehen, was immer wieder falsch gemacht wird. Schauen Sie sich die Beispiele in der Tabelle 18 an. Die korrekte Form der fehlerhaften Beispiele finden Sie in der Tabelle 20.

Tabelle 18. *Korrekte und falsche Case-Anweisungen.*

Beispiel	Gültig	Beschreibung
`Select Case i` `  Case 2`	Korrekt	Der Case-Ausdruck 2 wird zur Zahl 2 ausgewertet und mit i verglichen.
`Select Case i` `  Case Is = 2`	Korrekt	Der Case-Ausdruck 2 wird zur Zahl 2 ausgewertet und mit i verglichen.
`Select Case i` `  Case Is > 7`	Korrekt	Der Case-Ausdruck 7 wird zur Zahl 7 ausgewertet und mit i verglichen.
`Select Case i` `  Case 4, 7, 9`	Korrekt	Der Bedingungsausdruck i wird nacheinander mit 4, 7 und 9 verglichen.
`Select Case x` `  Case 1.3 To 5.7`	Korrekt	Sie können einen Bereich definieren und Fließkommazahlen verwenden.
`Select Case i` `  Case i = 2`	Falsch	Der Case-Ausdruck (i=2) wird als True oder False ausgewertet. Dieser boolesche Wert wird mit i verglichen. Kurzform für „Is = (i=2)“.
`Select Case i` `  Case i<2 Or i>9`	Falsch	Der Case-Ausdruck (i<2 Or 9<i) wird als True oder False ausgewertet. Dieser boolesche Wert wird mit i verglichen. Kurzform für „Is = (i<2 Or 9<i)“.
`Select Case i%` `  Case i%>2 And i%<10`	Falsch	Der Case-Ausdruck (i%>2 And i% < 10) wird als True oder False ausgewertet. Dieser boolesche Wert wird mit i verglichen. Kurzform für „Is = (i%>2 And i% < 10)“.
`Select Case i%` `  Case Is>8 And i<11`	Falsch	Wiederum wird i% mit True und False verglichen. Es ist die Kurzform für „Is > (8 And i<11)“. Die Prioritätsregeln machen daraus „Is > (8 And (i<11))“. Es ist nicht anzunehmen, dass dies gewünscht ist.
`Select Case i%` `  Case Is>8 And Is<11`	Falsch	Kompilierungsfehler. Das Schlüsselwort Is muss direkt auf Case folgen.

Ich habe Beispiele mit falschen Definitionen wie „`Case i > 2 And i < 10`“ gesehen. Das geht schief. Glauben Sie so etwas nicht, auch wenn Sie es gedruckt lesen. Wenn Sie verstehen, warum es falsch ist, haben Sie die Case-Anweisungen gemeistert.

Das vorletzte unkorrekte Beispiel in der Tabelle 18 ist repräsentativ für die meisten Fehler in Case-Ausdrücken, die ich gesehen habe. Listing 57 behandelt den Fall, dass i kleiner als 11 ist, und den Fall, dass i größer als oder gleich 11 ist. Schlicht und einfach bewirkt `Is > 8 And i < 11`, dass der Case-Ausdruck den Wert von i mit dem Ergebnis eines booleschen Ausdrucks vergleicht, der nur 0

oder -1 sein kann. Das große Problem mit Case-Anweisungen besteht darin, dass sie wie If-Anweisungen aussehen, die nach True oder False fragen, aber Case-Anweisungen suchen einen bestimmten Wert, der mit dem Bedingungswert verglichen wird. Dafür sind 0 oder -1 nicht gerade hilfreich.

Nehmen wir den zweiten Fall im Listing 57, `i >= 11`. Der Operator < hat eine höhere Priorität als der Operator `And`, wird also zuerst ausgeführt. Der Ausdruck `i<11` ergibt False (wegen der Annahme, dass `i>=11`). False wird intern als 0 repräsentiert. Da bei null keine Bits gesetzt sind, ergibt `8 And 0` den Wert null. Für die Fälle, dass i größer als oder gleich 11 ist, wird der gesamte Ausdruck gleichbedeutend mit „`Is > 0`“. Mit anderen Worten, die Case-Anweisung wird für `i = 45` unerwünschterweise als Treffer angenommen.

Eine ähnliche Argumentation für Werte von i kleiner als 11 – dem Leser zur Übung überlassen – zeigt, dass die Case-Anweisung gleichbedeutend ist mit „`Case Is > 8`“. Daher werden Werte von i kleiner als 11 korrekt ausgewertet, nicht aber Werte von i größer als oder gleich 11.

Listing 57. *„Case Is > 8 And i < 11“ wird in unerwarteter Weise reduziert.*

```
Is > (8 And i < 11) => Is > (8 And -1) => Is > 8 'Korrekt, wenn i < 11
Is > (8 And i < 11) => Is > (8 And  0) => Is > 0 'Falsch, wenn i >= 11
```

Wie man fehlerfreie Case-Ausdrücke schreibt

Nachdem Sie ein paar einfache Beispiele kennengelernt haben, wird es Ihnen leichtfallen, fehlerfreie Case-Ausdrücke zu schreiben. Tabelle 19 listet die Varianten theoretisch auf, und Listing 61 zeigt sie in konkreten Beispielen.

Tabelle 19. *Einfache Case-Varianten.*

Beispiel	Beschreibung
`Case Is Operator Ausdruck`	Dieser Fall ist zugleich der einfachste wie auch der schwierigste. Wenn der Ausdruck aus einer Konstanten besteht, ist der Fall einfach. Wenn der Ausdruck aber komplexer ist, besteht der schwierige Teil darin, den Ausdruck zu definieren.
`Case Ausdruck`	Dies ist eine Kurzform von „Case Is Operator Ausdruck“, wenn der Operator auf Gleichheit prüft.
`Case Ausdruck To Ausdruck`	Prüft einen begrenzten Bereich. Wird gewöhnlich richtig gemacht.
`Case Ausdruck, Ausdruck, ...`	Jeder Ausdruck wird verglichen. Wird gewöhnlich richtig gemacht.

In den schwierigen Fällen reicht es, einen Ausdruck zu finden, der zu dem Bedingungswert ausgewertet wird, wenn es ein Treffer sein soll, und irgendetwas anderes ergibt, wenn es kein Treffer sein soll. Mit anderen Worten, im Falle von Select Case 4 muss der Ausdruck 4 ergeben, wenn der Anweisungsblock ausgeführt werden soll.

Listing 58. *Select Case x (String): Test mit booleschem Ausdruck.*

```
Select Case x
  Case IIF(boolescher Ausdruck, x, x & "1") ' Für den Fall, dass x ein String ist
```

In Listing 58 wird x zurückgegeben, wenn der boolesche Ausdruck True ergibt. Der Ausdruck x=x ist True, also ist die Case-Anweisung ein Treffer. Wenn der boolesche Ausdruck False ergibt, wird x&"1" zurückgegeben. Dieser String ist nicht mehr derselbe wie x, also ist die Case-Anweisung kein Treffer. Ähnlich macht man das auch mit numerischen Werten.

Listing 59. *Select Case x (numerisch): Test mit booleschem Ausdruck.*

```
Select Case x
  Case IIF(boolescher Ausdruck, x, x + 1) ' Für den Fall, dass x numerisch ist
```

In Listing 59 wird x zurückgegeben, wenn der boolesche Ausdruck True ergibt. Der Ausdruck x=x ist True, also ist die Case-Anweisung ein Treffer. Wenn der boolesche Ausdruck False ergibt, wird x+1 zurückgegeben. Für numerische Werte ergibt x=x+1 nicht True, also ist die Case-Anweisung kein Treffer. Im allgemeinen funktioniert das, doch es besteht das Risiko des numerischen Überlaufs. Von Bernard Marcelly, einem Mitglied des Projekts Französischsprachiges OOo, kommt eine brillante und elegantere Lösung für numerische Werte.

```
Case x Xor Not (boolescher Ausdruck)
```

Dies setzt voraus, dass Case ein Treffer sein soll, wenn der boolesche Ausdruck True (-1) ergibt, und kein Treffer, wenn False (0) herauskommt.

Listing 60. *Xor und Not in einer Case-Anweisung.*

```
x Xor Not(True)  = x Xor Not(-1) = x Xor  0 =  x
x Xor Not(False) = x Xor Not( 0) = x Xor -1 <> x
```

Zuerst war ich ziemlich verwirrt, bemerkte aber dann, wie brillant das tatsächlich ist. Es entsteht kein Überlaufproblem und funktioniert mit allen Integer-Werten von x. Vereinfachen Sie es nicht zur fehlerhaften Verkürzung „x And (boolescher Ausdruck)“, denn das versagt, wenn x null ist.

Listing 61. *Beispiel für Select Case.*

```
Sub ExampleSelectCase
  Dim i%
  i = Int((20 * Rnd) - 2) 'Rnd liefert eine Zufallszahl zwischen null und eins
  Select Case i%
    Case 0
      Print "" & i & " ist null"
    Case 1 To 5
      Print "" & i & " ist eine Zahl von 1 bis 5"
    Case 6, 7, 8
      Print "" & i & " ist die Zahl 6, 7 oder 8"
    Case IIf(i > 8 And i < 11, i, i + 1)
      Print "" & i & " ist größer als 8 und kleiner als 11"
    Case i% Xor Not(i% > 10 And i% < 16)
      Print "" & i & " ist größer als 10 und kleiner als 16"
    Case Else
      Print "" & i & " ist außerhalb des Bereichs 0 bis 15"
  End Select
End Sub
```

ExampleSelectCase in Listing 61 liefert bei jedem Durchlauf eine Zufallszahl als Ganzzahl von -2 bis 18. Starten Sie das Makro mehrfach hintereinander, so dass Sie jede Case-Anweisung ausgeführt bekommen. Jeder dieser Fälle könnte das IIf-Konstrukt verwendet haben.

Nachdem ich nun die verschiedenen Methoden des Umgangs mit Bereichen erläutert habe, wird es Zeit, die fehlerhaften Fälle in Tabelle 18 zu überarbeiten. Die Lösungen in Tabelle 20 sind nicht die einzig möglichen Lösungen, nutzen aber einige der präsentierten Muster.

Tabelle 20. *Die fehlerhaften Case-Anweisungen der Tabelle 18 - nun korrigiert.*

Falsch	Richtig	Beschreibung
`Select Case i` `  Case i = 2`	`Select Case i` `  Case 2`	Die Variable i wird mit 2 verglichen.
`Select Case i` `  Case i = 2`	`Select Case i` `  Case Is = 2`	Die Variable i wird mit 2 verglichen.
`Select Case i` `  Case i<2 Or i>9`	`Select Case i` `  Case IIf(i<2 Or i>9, i, i+1)`	Funktioniert auch, wenn i keine Ganzzahl ist.
`Select Case i%` `  Case i%>2 And i%<10`	`Select Case i%` `  Case 3 To 9`	i% ist eine Ganzzahl, also geht der Bereich von 3 bis 9.

Falsch	Richtig	Beschreibung
`Select Case i%` `  Case Is>8 And i<11`	`Select Case i%` `  Case i Xor Not(i>8 And i< 11)`	Funktioniert, weil i% eine Ganzzahl ist.

3.9.7. While ... Wend

Mit der Anweisung While ... Wend wiederholen Sie einen Anweisungsblock, solange eine Bedingung wahr ist. Dieses Konstrukt hat gewisse Nachteile gegenüber der Schleifenanweisung Do While ... Loop, bietet aber auch keine Vorteile. While ... Wend unterstützt keine Exit-Anweisung, und Sie können While ... Wend nicht mit GoTo verlassen.

```
While Bedingung
  Anweisungsblock
Wend
```

Visual Basic .NET kennt das Schlüsselwort Wend nicht, ein weiterer Grund, stattdessen Do While ... Loop zu verwenden.

3.9.8. Do ... Loop

Der Loop-Mechanismus kennt verschiedene Formen und wird eingesetzt, wenn ein Anweisungsblock wiederholt ausgeführt werden soll, solange oder bis eine Bedingung wahr ist. In der gebräuchlichsten Form wird die Bedingung geprüft, bevor die Schleife startet, und der Anweisungsblock wird so lange wiederholt, wie die Bedingung wahr bleibt. Wenn die Eingangsbedingung falsch ist, wird die Schleife gar nicht ausgeführt.

```
Do While Bedingung
  Block
  [Exit Do]
  Block
Loop
```

In ähnlicher, aber nicht so häufig vorkommender Form wird der Block so lange wiederholt, wie die Bedingung falsch ist. Mit anderen Worten, der Code wird ausgeführt, bis die Bedingung wahr wird. Wenn sich die Bedingung schon zu Beginn als wahr zeigt, wird die Schleife gar nicht ausgeführt.

```
Do Until Bedingung
  Block
  [Exit Do]
  Block
Loop
```

Sie können die Prüfung der Bedingung an das Schleifenende setzen, so dass der Anweisungsblock wenigstens einmal ausgeführt wird. In der folgenden Form läuft die Schleife wenigstens einmal durch und wird danach so lange wiederholt, wie die Bedingung wahr ist:

```
Do
  Block
  [Exit Do]
  Block
Loop While Bedingung
```

In der folgenden Form führen Sie die Schleife wenigstens einmal aus und danach immer wieder, solange die Bedingung falsch ist:

```
Do
  Block
  [Exit Do]
```

```
    Block
Loop Until Bedingung
```

Aussteigen aus der Do-Schleife

Die Anweisung Exit Do bewirkt das sofortige Schleifenende. Sie ist nur zwischen Do und Loop zulässig. Das Programm wird mit der Anweisung fortgesetzt, die der innersten Loop-Anweisung folgt. Die Subroutine ExampleDo im Listing 62 zeigt eine Do-While-Schleife, mit der eine Zahl in einem Array gesucht wird.

Listing 62. *Beispiel für Do Loop.*

```
Sub ExampleDo
  Dim a(), i%, x%
  a() = Array(2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30)
  x = Int(32 * Rnd)                  REM Zufällige Ganzzahl von 0 bis 31
  i = LBound(a())                    REM i ist die Bereichsuntergrenze des Arrays.
  Do While a(i) <> x                 REM Solange a(i) ungleich x ist,
   i = i + 1                         REM wird i um 1 hochgezählt.
   If i > UBound(a()) Then Exit Do   REM Ausstieg, sobald i zu groß ist.
  Loop                               REM Die Schleife kehrt zu Do While zurück.
  If i <= UBound(a()) Then           REM Wenn i nicht zu groß ist, wurde x gefunden.
    MsgBox "Habe " & x & " gefunden an der Position " & i, 0, "Beispiel für Do"
  Else
    MsgBox "Konnte " & x & " nicht im Array finden", 0, " Beispiel für Do"
  End If
End Sub
```

Welche Do-Loop-Form ist zu wählen?

Basic unterstützt vier Varianten der Do-Loop-Konstruktion. Der Einsatzzweck und Zeitpunkt bestimmt die jeweilige Variante. Das häufigste Problem liegt darin, dass die Schleife einmal zu viel oder einmal zu wenig durchlaufen wird, weil der Bedingungsausdruck am falschen Ende platziert wurde.

Zur Entscheidung, wo der Bedingungsausdruck einer Do-Loop-Schleife stehen soll, stellen Sie sich folgende Frage: „Muss die Schleife mindestens einmal durchlaufen werden?“ Bei der Antwort Nein muss der Bedingungsausdruck nach oben. Das verhindert, dass die Anweisungen innerhalb der Schleife ausgeführt werden, wenn der Vergleich fehlschlägt. Stellen Sie sich vor, Sie wollen alle Elemente eines Arrays unbekannter Größe ausdrucken. Das Array könnte leer sein und gar keine Elemente enthalten. In diesem Fall sollte der Code innerhalb der Schleife gar nicht ausgeführt werden (s. Tabelle 21).

Tabelle 21. *Schleifen mit While (solange) und Until (bis) sind sehr ähnlich.*

Do While	Do Until

`i% = LBound(a())`
`Do While i% <= UBound(a())`
`  Print a(i%)`
`  i% = i% + 1`
`Loop`
	`i% = LBound(a())`
	`Do Until i% > UBound(a())`
	`  Print a(i%)`
	`  i% = i% + 1`
	`Loop`

In beiden Fällen der Tabelle 21 wird die Schleife gar nicht durchlaufen, wenn das Array leer ist. Vor der Auswertung der Bedingung wird i% auf den Wert der Bereichsuntergrenze gesetzt. In beiden Fällen läuft die Schleife, solange i% nicht größer ist als die Bereichsobergrenze.

Den Unterschied zwischen einer While-Schleife und einer Until-Schleife kann man an einem einfachen Beispiel sehen. Solange (While) Sie im Auto Gas geben, können Sie es fahren. Bis (Until) Sie im Auto kein Gas mehr geben, können Sie es fahren. Der Hauptunterschied zwischen While und Until ist das Wort Not. Die vorige Aussage würde in Basic eher lauten „Until Not (es wird Gas gege-

ben)“. Die Wahl zwischen While und Until richtet sich danach, welches Sie ohne Not schreiben können.

Wenn die Schleife mindestens einmal durchlaufen werden soll, stellen Sie den Bedingungsausdruck an das Ende von Do Loop. Gesetzt den Fall, Sie fordern eine Nutzereingabe, bis ein gültiger Wert geliefert wird. Der Bedingungsausdruck steht dabei natürlicherweise am Ende.

```
Dim s$, x As Double
Do
  s$ = InputBox("Geben Sie bitte eine Zahl zwischen 1 und 5 ein")
  x = CDbl(s$)      'Konvertierung des Strings zu Double
Loop Until x >= 1 And x <= 5
```

Die Schleife braucht wenigstens einen Durchlauf, so dass wenigstens eine Zahl eingegeben werden kann. Die Schleife wird so lange wiederholt, bis eine gültige Zahl eingegeben wird. Zur Übung können Sie diese Schleife einmal mit While schreiben.

3.9.9. For ... Next

Die Anweisung For ... Next wiederholt einen Anweisungsblock für eine bestimme Anzahl von Durchläufen.

```
For zähler = startwert To endwert [Step schrittWert]
  Anweisungsblock1
  [Exit For]
  Anweisungsblock2
Next [zähler]
```

Die numerische Variable „zähler“ wird auf den Wert von „startwert“ initialisiert. Wenn das Programm zur Anweisung Next gelangt, wird zum „zähler“ der Step „schrittwert“ addiert. Ist kein Step-Wert angegeben, wird 1 addiert. Wenn „zähler“ kleiner oder gleich dem Wert von „endwert“ ist, wird der Anweisungsblock durchlaufen. Im folgenden eine äquivalente Do-While-Schleife:

```
zähler = startwert
step = schrittwert
Do While zähler <= endwert
  Anweisungsblock1
  [Exit Do]
  Anweisungsblock2
  zähler = zähler + step
Loop
```

Die Angabe von „zähler“ hinter der Next-Anweisung ist optional. Next bezieht sich automatisch auf die zuletzt vorgekommene For-Anweisung.

```
For i = 1 To 4 Step 2
  Print i ' Gibt erst 1, dann 3 aus.
Next i    ' In dieser Anweisung ist das i optional.
```

Mit der Anweisung Exit For verlassen Sie unmittelbar den For-Block, und zwar den nach der zuletzt vorgekommenen For-Anweisung. Listing 63 zeigt das mit einer Sortierroutine. Ein Array wird mit zufälligen Ganzzahlen gefüllt und dann mit Hilfe von zwei verschachtelten Schleifen sortiert. Diese Technik heißt „modifiziertes Bubblesort (Blasensortierung)“.

Zuerst wird iOuter auf die letzte Zahl des Arrays gesetzt. Die innere Schleife nutzt iInner und vergleicht jede Zahl mit der nachfolgenden. Wenn die erste Zahl größer als die zweite ist, werden die zwei Zahlen vertauscht. Die zweite Zahl wird nun mit der dritten verglichen. Nach dem Ende der inneren Schleife ist sichergestellt, dass die größte Zahl im Array die letzte Position einnimmt.

Danach wird iOuter um 1 heruntergezählt. Die innere Schleife ignoriert diesmal die letzte Zahl des Arrays und vergleicht wiederum jede Zahl mit der ihr folgenden. Nach dem Ende der inneren Schleife ist die zweitgrößte Zahl an der zweitletzten Stelle im Array.

Mit jedem Wiederholungsschritt wird eine weitere Zahl an die richtige Stelle geschoben. Wenn keine Zahlen vertauscht werden, ist die Liste sortiert.

Listing 63. *Modifiziertes Bubblesort.*

```
Sub ExampleForNextSort
  Dim iEntry(10) As Integer
  Dim iOuter As Integer, iInner As Integer, iTemp As Integer
  Dim bSomethingChanged As Boolean  'True, wenn eine Änderung erfolgte

  ' Das Array wird mit Ganzzahlen zwischen -10 und 10 gefüllt.
  For iOuter = LBound(iEntry()) To Ubound(iEntry())
    iEntry(iOuter) = Int((20 * Rnd) - 10)
  Next iOuter

  ' iOuter erhält nacheinander den Arrayindex von hinten nach vorne.
  For iOuter = UBound(iEntry()) To LBound(iEntry()) Step -1
    'Setzt voraus, dass das Array schon sortiert ist.
    'Prüft, ob es wirklich zutrifft.
    bSomethingChanged  = False
    For iInner = LBound(iEntry()) To iOuter - 1
      If iEntry(iInner) > iEntry(iInner + 1) Then
        iTemp = iEntry(iInner)
        iEntry(iInner) = iEntry(iInner + 1)
        iEntry(iInner + 1) = iTemp
        bSomethingChanged  = True
      End If
    Next iInner
    'Wenn das Array schon sortiert ist, wird die Schleife verlassen!
    If Not bSomethingChanged Then Exit For
  Next iOuter
  Dim s$
  For iOuter = LBound(iEntry()) To Ubound(iEntry())
    s = s & iOuter & " : " & iEntry(iOuter) & Chr$(10)
  Next iOuter
  MsgBox s, 0, "Sortiertes Array"
End Sub
```

3.9.10. Exit Sub und Exit Function

Mit der Anweisung Exit Sub wird eine Subroutine abrupt beendet, gleichermaßen eine Funktion durch Exit Function. Das Makro fährt mit der Anweisung fort, die der Anweisung folgt, mit der die Routine aufgerufen wurde. Die Exit-Anweisungen beenden nur die aktuell laufende Routine und gelten nur für die entsprechenden Typen, zum Beispiel können Sie Exit Sub nicht in einer Funktion verwenden.

3.10. Fehlerbehandlung mit On Error

Fehler fallen gewöhnlich in drei Kategorien – beim Kompilieren, zur Laufzeit sowie logische Fehler. Fehler beim Kompilieren sind üblicherweise Syntaxfehler wie zum Beispiel nicht vorhandene Anführungszeichen, die das Kompilieren Ihres Makros verhindern. Mit Fehlern zur Kompilierungszeit ist am einfachsten umzugehen, weil sie sofort gefunden werden, denn die IDE zeigt Ihnen, welche Zeile das Problem verursacht hat. Laufzeitfehler werden ordentlich kompiliert, treten aber auf, wenn das Makro läuft. Wenn zum Beispiel durch eine Variable dividiert wird, die an einem bestimmten Punkt den Wert null annimmt, wird ein Laufzeitfehler produziert. Der dritte Typ, logische Fehler,

sind Irrtümer in der konzeptionellen Logik des Programms. Sie werden kompiliert, laufen fehlerlos, liefern aber die falschen Antworten. Das sind die schlimmsten Fehler, denn Sie müssen sie selbst finden – der Rechner kann Ihnen dabei nicht helfen. Dieser Abschnitt handelt von Laufzeitfehlern: wie man mit ihnen umgeht und wie man sie behebt.

Eine Fehlerbehandlungsroutine ist ein Programmteil, der beim Auftritt eines Fehlers gestartet wird. Die Standard-Fehlerbehandlung zeigt eine Fehlermeldung und bricht das Makro ab. Basic bietet einen Mechanismus, dieses Verhalten zu beeinflussen (s. Tabelle 22). Das erste Format, On Error Resume Next, veranlasst OOo, alle Fehler zu ignorieren: Was auch immer passiert, mach weiter und tu so, als wäre alles in Ordnung. Das zweite Format, On Error GoTo 0, deaktiviert die aktuelle Fehlerbehandlung. Ungeachtet der an späterer Stelle erläuterten Bereichsaspekte der Fehlerroutine betrachten Sie On Error GoTo 0 als Mittel, die Standardmethode der Fehlerbehandlung wiederherzustellen: Brich das Makro ab und zeige eine Fehlermeldung. Das letzte Format, On Error GoTo LabelName, erlaubt Ihnen, Code zu schreiben, der Fehler nach Ihren eigenen Wünschen behandelt. Sie erstellen einen „Error-Handler“.

Tabelle 22. *Unterstützte On Error ...-Formate.*

Format	Einsatzbereich
`On Error Resume Next`	Ignoriert Fehler und setzt das Makro mit der folgenden Zeile fort.
`On Error GoTo 0`	Bricht die aktuelle Fehlerbehandlung ab.
`On Error GoTo LabelName`	Verzweigt zur genannten Sprungmarke.

Wenn ein Fehler auftritt, wird der aktuell ausgeführte Code abgebrochen und die Kontrolle dem aktuellen Error-Handler übertragen. Den Error-Handlers stehen die Funktionen in Tabelle 23 zur Verfügung, um die Ursache und die Position des Fehlers herauszufinden. Visual Basic setzt ein Error-Objekt ein und unterstützt die Funktionen der Tabelle 23 nicht.

Tabelle 23. *Variablen und Funktionen im Zusammenhang mit Fehlerroutinen.*

Funktion	Verwendung
CVErr	Konvertiert einen Ausdruck in ein Error-Objekt.
Erl	Nummer der Zeile, in der der letzte Fehler auftrat.
Err	Nummer des zuletzt aufgetretenen Fehlers.
Error Error(Fehlernummer)	Fehlermeldung des zuletzt aufgetretenen Fehlers, beziehungsweise des Fehlers mit der angegebenen Nummer.

Alle Error-Handlers müssen in einer Routine definiert werden und gehören somit zu eben dieser Subroutine oder Funktion. Wenn ein Fehler auftritt, arbeitet sich Basic rückwärts durch den Aufrufstapel, bis ein Error-Handler gefunden ist. Falls keiner zu finden ist, wird der Standard-Handler eingesetzt, der eine Fehlermeldung ausgibt und das Programm abbricht. Die Fehlerinformation Erl zeigt die Nummer der Zeile in der aktuellen Routine an, die den Fehler verursacht hat. Wenn zum Beispiel die aktuelle Routine in der Zeile 34 die Funktion b() aufruft und ein Fehler innerhalb von b() auftritt, wird ein Fehler gemeldet, der in der Zeile 34 aufgetreten ist. Listing 64 enthält dafür ein Beispiel, und Bild 36 zeigt den Aufrufstapel. Ein weiteres Beispiel finden Sie in Listing 69.

Listing 64. *Mit Erl wird die Zeilennummer ausgegeben.*

```
  x = x + 1                 'Angenommen, dies ist Zeile 33
  Call b()                  'Zeile 34:
                            'Fehler in b() oder in einer von b() aufgerufenen Routine
  Exit Sub                  'Abbruch der Subroutine
ErrorHandler:               'Kein weiterer Error-Handler zwischen hier und dem Fehler
  Print "Fehler in der Zeile " & Erl  'Gibt die Zeilennummer 34 aus
```

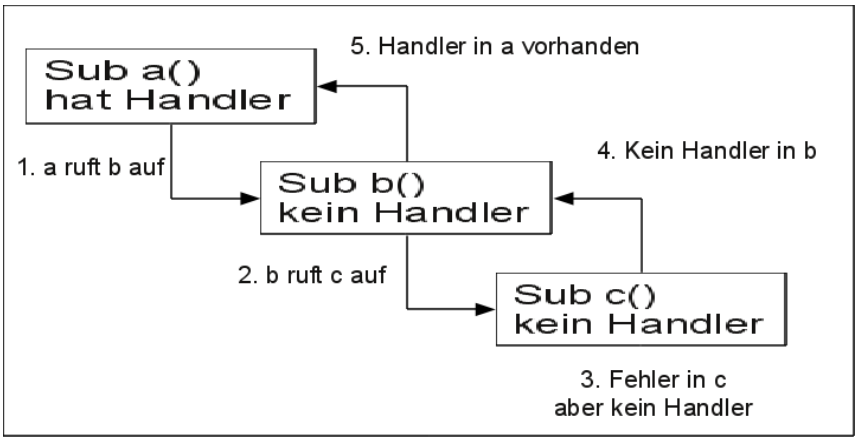


Bild 36. Der Weg durch den Aufrufstapel auf der Suche nach einem Error-Handler.

Sie können keine Fehler abfangen, die in einer DLL auftreten. Prüfen Sie stattdessen den Rückgabecode der aufgerufenen DLL.

Der Gültigkeitsbereich der Anweisung „On Error ...“ ist global. Sie bleibt bis zur nächsten „On Error ...“-Anweisung aktiv. Optional kann aber der Parameter „Local“ hinzugefügt werden. Die Anweisung „On Local Error ...“ beschränkt den Gültigkeitsbereich auf die Routine, in der sie steht. Ein lokaler Error-Handler setzt jeden global definierten Error-Handler vorübergehend außer Kraft. Wenn die aktuelle Routine beendet ist, wird auch der lokale Error-Handler gelöscht und der globale Error-Handler wieder aktiviert.

3.10.1. CVErr

Mit CVErr (in Tabelle 23 erwähnt) wird ein OOo-spezifischer interner Datentyp erzeugt, der einen Fehler verkörpert. Ich habe es noch in keinem Makro gesehen, aber es kann für eine sehr robuste Funktion hilfreich sein.

CVErr gibt einen internen OOo-Typ zurück, den man einer Variant-Variablen zuweisen sollte.

- Mit VarType wird der Typ des zurückgegebenen Werts ermittelt. Der VarType 10 heißt, dass ein Error-Objekt zurückgegeben wurde (s. Tabelle 81 im Abschnitt 9.8. Inspizierung und Erkennung von Variablen).
- CVErr akzeptiert ein Integer-Argument als interne Fehlernummer für den aufgetretenen Error. Durch die Konvertierung des internen Error-Objekts zu Integer wird die interne Fehlernummer zurückgegeben.

Das folgende Beispiel demonstriert den Gebrauch von CVErr.

***Listing 65.** Der Gebrauch von CVErr.*

```
Sub CallNotZero
  Dim xVar As Variant
  Dim xInt As Integer
  Dim i As Integer
  Dim s As String
  For i = -1 To 1
    xInt = NotZero(i)
    xVar = NotZero(i)
    s = s & "NotZero(" & i & ") = [" & xInt & "] als Zuweisung zu einem Integer"
    s = s & Chr$(10)
    s = s & "NotZero(" & i & ") = [" & xVar & "] VarType=" & VarType(xVar)
    s = s & Chr$(10)
  Next
```

```
  MsgBox s
End Sub

Function NotZero(x As Integer)
  If x <> 0 Then
    NotZero = x
  Else
    ' 7 ist eine willkürlich gewählte Zahl und bedeutet gar nichts.
    NotZero = CVErr(7)
  End If
End Function
```

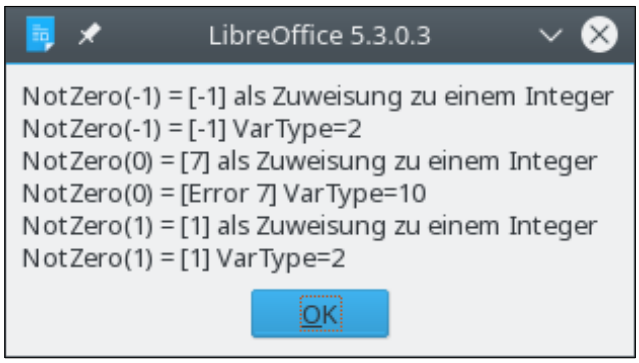


Bild 37. *Rückgabewerte mit CVErr.*

3.10.2. Fehler ignorieren mit On Error Resume Next

Fehlerbehandlung bedeutet in manchen Fällen Fehlerignorierung. Die Anweisung On Error Resume Next zwingt Basic im Falle eines Standardfehlers, den Fehler zu ignorieren und mit der nächsten Zeile im Makro weiterzumachen (s. Listing 66). Die Fehlerinformation wird gelöscht, so dass es nach der verursachenden Anweisung nicht mehr möglich ist zu prüfen, ob ein Fehler aufgetreten ist.

Listing 66. *Mit der Anweisung Resume Next ist der Fehler gelöscht.*

```
Private zero%
Sub ExampleErrorResumeNext
  On Error Resume Next
  Print 1 / zero%
  If Err <> 0 Then Print Error$ & " in der Zeile " & Erl 'Err wurde gelöscht
End Sub
```

3.10.3. Mit On Error GoTo 0 einen Error-Handler ausschalten

Verwenden Sie On Error GoTo 0, um einen definierten Error-Handler auszuschalten, üblicherweise innerhalb eines Error-Handlers oder nach dem Code, für den er wirken soll. Wenn ein Fehler innerhalb eines Error-Handlers auftritt, wird er nicht abgefangen. Das Makro bricht einfach ab.

Listing 67. *Der Error-Handler wird mit der Anweisung On Error GoTo 0 ausgeschaltet.*

```
Private zero%
Sub ExampleErrorResumeNext
  On Error Resume Next
  Print 1 / zero%
  On Error GoTo 0
  ...
End Sub
```

In einigen Versionen von Visual Basic wird auch On Error GoTo -1 als äquivalent zu On Error GoTo 0 akzeptiert.

3.10.4. Fehlermeldungen sichern

Das Makro im Listing 68 prüft einen Fehler, bevor der Error-Handler zurückgesetzt ist. Dann werden Fehlernummer und Zeilennummer gesichert. Obwohl das Makro den Text der Fehlermeldung nicht explizit speichert, kann er später von der Funktion Error ausgegeben werden, wenn die Fehlernummer als optionales Argument angegeben wird, s. auch Bild 38.

Listing 68. *Beispiel für die Sicherung einer Fehlermeldung.*

```
Sub ExampleError
  On Error Goto BadError        'Definition des Error-Handlers
  Print 1/ CInt(0.2)            'Division durch null
BadError:                       'Hier beginnt der Error-Handler
  Dim s As String               'Kumulierter Meldungstext
  Dim oldError As Integer       'Sicherung der Fehlernummer
  Dim lineNum As Integer        'Sicherung der Zeilennummer
  If Err <> 0 Then              'Falls ein Fehler auftritt,
    oldError = Err              'wird die Nummer gesichert
    lineNum = Erl               'und die Zeile gesichert.
    s = "Vor der Zurücksetzung des Error-Handlers:" & Chr$(10) &_
        "Ein Fehler mit der Nummer " & Err & " trat in der Zeile " & Erl & " auf." &_
        Chr$(10) & "Fehlermeldung: " & Error() & Chr$(10)
  End If
  On Error Goto 0               'Der Error-Handler wird zurückgesetzt

  REM Jetzt gibt es keine Informationen mehr
  s = s & Chr$(10) & "Nach der Zurücksetzung des Error-Handlers:" & Chr$(10) &_
      "Ein Fehler mit der Nummer " & Err & " trat in der Zeile " & Erl & " auf." &_
      Chr$(10)

  REM Verwendet die gesicherten Informationen
  s = s & Chr$(10) & "Die Fehlerinformation wurde so gesichert:" & Chr$(10) &_
      "Ein Fehler mit der Nummer " & oldError & " trat in der Zeile " &_
      lineNum & " auf." & Chr$(10) & "Fehlermeldung: " & Error(oldError)
  MsgBox s, 0, "Fehlerbehandlung"
End Sub
```

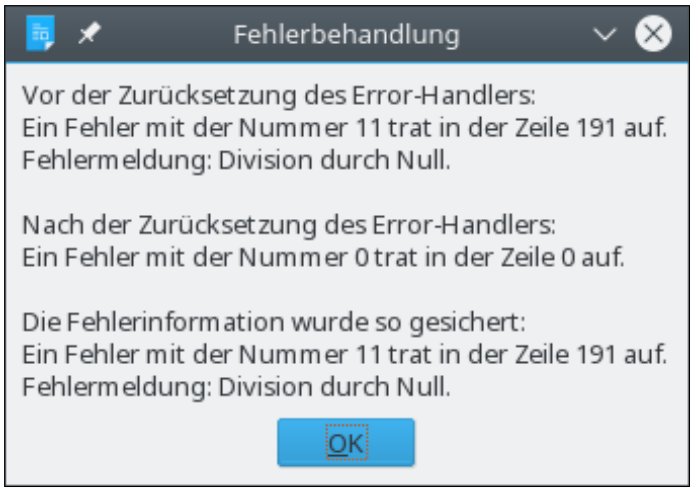


Bild 38. *Fehlerinformationen gehen verloren, wenn sie vor der Zurücksetzung des Error-Handlers nicht gesichert werden.*

3.10.5. Mit On Error GoTo Label einen eigenen Error-Handler definieren

Ihren eigenen Error-Handler richten Sie mit On Error GoTo Label ein. In Basic definieren Sie ein Label, indem Sie in einer gesonderten Zeile Text eingeben, dem ein Doppelpunkt folgt. Zeilenlabel

dürfen im Makro unter demselben Namen mehrfach vorkommen, innerhalb einer Routine müssen sie aber eindeutig sein. Somit können Fehlerbehandlungsroutinen immer gleich benannt werden, statt für jeden Error-Handler einen ganz eigenen Namen finden zu müssen (s. Listing 69 und Bild 39). Im Falle eines Fehlers wird der Programmfluss zu dem Label verzweigt.

Listing 69. *Fehlerbehandlung.*

```
Private zero%
Private error_s$
Sub ExampleJumpErrorHandler
  On Error GoTo ExErrorHandler
  JumpError1
  JumpError2
  Print 1 / zero%
  MsgBox error_s, 0, "Sprung zum Error-Handler"
  Exit Sub
ExErrorHandler:
  error_s = error_s & "Fehler in MainJumpErrorHandler in der Zeile " & Erl() &_
        " : " & Error() & Chr$(10)
  Resume Next
End Sub

Sub JumpError1
  REM Bewirkt einen Sprung zum Handler in ExampleJumpErrorHandler.
  REM Der Haupt-Error-Handler zeigt an, dass die Fehlerposition
  REM beim Aufruf von JumpError1 liegt und nicht innerhalb von JumpError1.
  Print 1 / zero%
  error_s = error_s & "Hey, ich bin in JumpError1" & Chr$(10)
End Sub

Sub JumpError2
  On Error GoTo ExErrorHandler
  Print 1 / zero%
  Exit Sub
ExErrorHandler:
  error_s = error_s & "Fehler in JumpError2 in der Zeile " & Erl() &_
        " : " & Error() & Chr$(10)
  Resume Next
End Sub
```

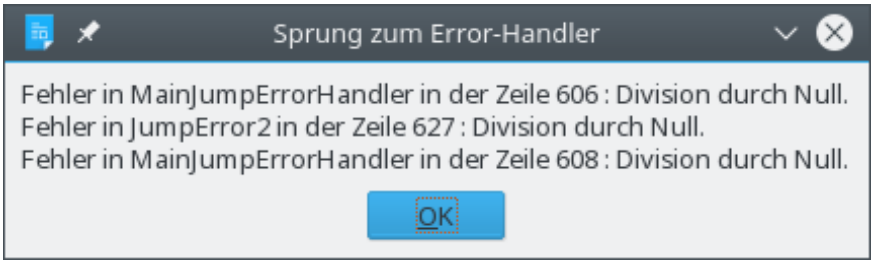


Bild 39. *Der zuletzt eingerichtete Error-Handler wird genutzt.*

Eine Routine kann mehrmals die Anweisung On Error enthalten. Jede dieser Anweisungen kann Fehler auf andere Weise behandeln. Beide Error-Handlers in Listing 69 benutzen Resume Next, ignorieren damit den Fehler und führen das Programm mit der auf den Fehler folgenden Zeile fort. Mit multiplen Error-Handlers ist es möglich, im Falle eines Fehlers Codebereiche zu überspringen (s. Listing 70).

Tipp Im Hilfetext von OOo 3.20 wird immer noch fälschlich behauptet, dass die Fehlerbehandlung am Anfang einer Routine eingefügt werden müsse.

Listing 70. *Überspringen von Codebereichen, wenn ein Fehler auftritt.*

```
  On Error GoTo PropertiesDone  'Ignoriert alle Fehler in diesem Bereich.
  a() = getProperties()         'Wenn getProperties() nicht funktioniert, dann wird ein
  DisplayStuff(a(), "Eigenschaften")  'Fehler verhindern, dass diese Zeile erreicht
                                      ' wird.
PropertiesDone:
  On Error GoTo MethodsDone        'Ignoriert alle Fehler in diesem Bereich.
  a() = getMethods()
  DisplayStuff(a(), "Methoden")
MethodsDone:
  On Error Goto 0                  'Schaltet alle aktuellen Error-Handler ab.
```

Wenn Sie einen Error-Handler schreiben, müssen Sie eine Entscheidung treffen, wie Sie die Fehler behandeln wollen. Mit Hilfe der Funktionen in der Tabelle 23 werden Fehler diagnostiziert und Fehlermeldungen ausgegeben oder in Logdateien geschrieben. Sie müssen aber auch den Programmablauf kontrollieren. Die Grundzüge der Fehlerbehandlung sind:

- Verlassen Sie die Subroutine oder Funktion mit Exit Sub oder Exit Function.
- Lassen Sie das Makro weiterlaufen und ignorieren Sie den Fehler (s. Listing 70).
- Setzen Sie mit Resume Next das Makro mit der dem Fehler folgenden Zeile fort (s. Listing 71 und Bild 40).
- Wiederholen Sie mit Resume die Ausführung der den Fehler erzeugenden Anweisung. Wenn das Problem aber nicht behoben ist, wird der Fehler wieder auftreten, mit der Folge einer Endlosschleife.
- Setzen Sie mit Resume LabelName das Programm an einer benannten Position fort.

Listing 71. *Error-Handler mit Resume Next.*

```
Sub ExampleResumeHandler
  Dim s$, z%
  On Error GoTo Handler1              'Gibt eine Meldung aus, macht bei Spot1 weiter.
  s = "(0) 1/z = " & 1/z & Chr$(10)   'Division durch null, also Sprung zu Handler1.
Spot1:                                'Von Handler1 hierher gekommen.
  On Error GoTo Handler2              'Handler2 verwendet Resume.
  s = s & "(1) 1/z = " & 1/z & Chr$(10) 'Zuerst Error, beim zweiten Mal korrekt.
  On Error GoTo Handler3              'Handler3 macht mit der nächsten Zeile weiter.
  z = 0                               'Ermöglicht eine weitere Division durch null.
  s = s & "(2) 1/z = " &1 /z & Chr$(10) 'Fehler und Sprung zu Handler3
 MsgBox s, 0, "Error-Handler mit Resume"
 Exit Sub
Handler1:
  s = s & "Handler1 aufgerufen von der Zeile " & Erl() & Chr$(10)
  Resume Spot1
Handler2:
  s = s & "Handler2 aufgerufen von der Zeile " & Erl() & Chr$(10)
  z = 1   'Behebt den Fehler und führt die Zeile noch einmal aus.
  Resume
Handler3:
  s = s & "Handler3 aufgerufen von der Zeile " & Erl() & Chr$(10)
  Resume Next
End Sub
```

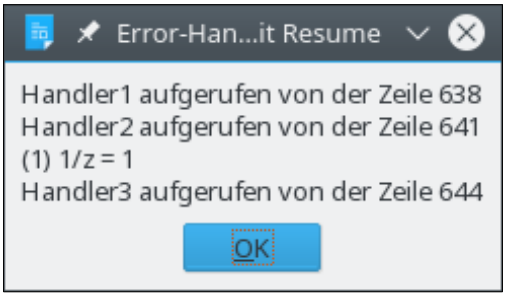


Bild 40. *Der jeweils zuletzt deklarierte Error-Handler wird genutzt.*

Tipp	In einem Error-Handler auftretende Fehler werden nicht behandelt. Das Makro bricht einfach ab.

3.10.6. Error-Handler – wozu?

Wenn ich ein Makro ausführe und es abstürzt, verstehe ich für gewöhnlich die manchmal kryptischen Fehlermeldungen und erkenne, wie ich damit umgehen muss. Wenn andere meine Makros ausführen und Fehler auftreten, informieren sie mich normalerweise darüber, weil sie nicht wissen, wie sie sie behandeln sollen. Das ist dann ein guter Indikator dafür, dass ich keine angemessene Fehlerbehandlung eingebaut habe.

Sie brauchen nicht für jede Routine einen Error-Handler. Wenn die aktuelle Routine keinen hat, aber die sie aufrufende Routine, wird deren Error-Handler aktiviert. Gesetzt den Fall, Sub1 hat einen Error-Handler und ruft Sub2 auf, das keinen hat. Wenn in Sub2 ein Fehler auftritt, wird der Error-Handler in Sub1 aktiviert.

Wenn Sie einen Error-Handler einsetzen, bestimmen Sie, wie und wann der Nutzer über einen Fehler informiert wird. Es gibt auch andere Gründe, einen Error-Handler zu nutzen, als die Kontrolle über die Fehlermeldungen auszuüben. Mit Bedacht eingesetzt können Error-Handlers den Umfang Ihres Makrocodes reduzieren. Nehmen Sie einmal die mathematischen Operationen. Es ist lästig, jede mathematische Operation vor ihrem Gebrauch zu testen.

```
If x <> 0 Then y = z / x
If x > 0 Then y = Log(x)
If i% < 32767 Then i% = i% + 1
```

Trotz meines paranoiden Codes, die Argumente zu testen, könnte ich einen numerischen Überlauf produzieren. Schlimmer noch, nichts geschieht im Falle eines Fehlers, das Makro wird normal weiter ausgeführt. Manchmal können Sie auch gar nichts testen, um den Fehler zu vermeiden. Vor OOo 2.0 gab zum Beispiel die Funktion DimArray ein ungültiges leeres Array zurück. Die Funktionen LBound und UBound generieren Ausnahmefehler bei diesen ungültigen leeren Arrays. Mit einer passenden Fehlerbehandlung werden LBound und UBound auch im Falle eines Fehlers sicher eingesetzt. Betrachten Sie folgende Fälle:

- Das Argument ist kein Array.
- In einem leeren Array ist zum Beispiel UBound < LBound: -1 und 0.
- Es gibt kein Problem, wenn das Array kein ungültiges leeres Array ist.
- Sollte die optionale Dimensionierung berücksichtigt werden?

Der Code in Listing 72 zeigt einen einfachen Error-Handler, der schlicht Fehler ignorieren kann. Die Funktion gibt True zurück, wenn die untere Dimensionsgrenze kleiner als oder gleich groß wie die obere ist – mit anderen Worten, ob das Array Daten enthält. Wenn ein Fehler auftritt, sei es, dass das Argument kein Array oder ein ungültiges leeres Array ist, wird die Zeile nicht bis zum Ende ausgeführt, so dass es gar keine Wertzuweisung gibt. Wenn die Zuweisung nicht erfolgt, wird der originale

Standardwert False zurückgegeben. Das ist dann nämlich die korrekte Antwort. Dem Leser bleibt es als Übung, eine sichere Routine für die obere und untere Grenze zu schreiben – allerdings sind die sicheren Versionen seit OOo 2.0 nicht mehr nötig, weil die Funktionen UBound und LBound gefixt sind.

Listing 72. *Ermittelt, ob ein Array Daten enthält.*

```
Sub ExampleArrayHasStuff
  Dim a(), b(3), v
  Print ArrayHasStuff(a())         'False, weil leer
  Print ArrayHasStuff(v)           'False, kein Array, der Error-Handler greift ein.
  Print ArrayHasStuff(DimArray()) 'False, ungültiges Array, der Error-Handler greift
                                   'ein.
  Print ArrayHasStuff(DimArray(3))'True,  nicht leer
  Print ArrayHasStuff(b())         'True,  nicht leer
End Sub

Function ArrayHasStuff(v) As Boolean
  REM Der Standardwert für Boolean ist False, also ist die Standardantwort False.
  REM Wenn ein Fehler auftritt, wird die Anweisung nicht beendet!
  REM Dies ist ein guter Fall für On Error Resume Next
  On Error Resume Next
  ArrayHasStuff = CBool(LBound(v) <= UBound(v))
End Function
```

Ein Error-Handler kann auch interaktiv sein. Der Code in Listing 73 versucht, eine nicht existierende Datei an eine nicht existierende Adresse zu kopieren. Keine Frage, es wird ein Fehler generiert. Eine Fehlermeldung wird ausgegeben mit der Frage an den Nutzer, ob die Kopie noch einmal versucht werden soll. Der Nutzer hat dadurch die Gelegenheit, Fehler zu korrigieren und weiterzumachen.

Listing 73. *Kopiert eine Datei.*

```
Sub ExampleCopyAFile()
  CopyAFile("/ich/bin/nicht/vorhanden.txt", "/ich/auch/nicht.txt")
End Sub

Sub CopyAFile(Src$, Dest$)
  On Error GoTo BadCopy:      'Richtet den Error-Handler ein.

  FileCopy(Src$, Dest$)       'Generiert den Fehler.

AllDone:                      'Ohne Fehler geht es hier weiter.
  Exit Sub                    'Rückkehr vom Handler zum Label AllDone.

BadCopy:                      'Ausgabe eines Fehlerdialogs.
  Dim rc%                     'Antwort auf die Frage nach einem neuen Versuch.
  rc% = MsgBox("Missglückte Kopie von " & Src$ & " zu " & Dest$ & Chr$(10) &_
         "Ursache: "&Chr$(10) & Error() & " In der Zeile " & Erl & Chr$(10) &_
         "Neuer Versuch?", (4 Or 32), "Fehler beim Kopieren")
  If rc% = 6 Then             'Ja, neuer Versuch
    Resume
  End If
  If rc% = 7 Then             'Nein, Sprung zum Label AllDone.
    Resume AllDone
  End If
End Sub
```

3.11. Fazit

Zum Aufbau eines nennenswerten OOo-Makroprogramms ist das Verständnis der Basic-Syntax unerlässlich. Dieses Kapitel deckte die wesentlichen Elemente ab:

- Die Syntax des Makros bestimmt den gültigen oder ungültigen Aufbau.
- Die Logik eines Makros bestimmt, was das Makro tut.
- Die Ablaufsteuerung lenkt die Abfolge der Anweisungen während der Laufzeit.
- Die Fehlerbehandlung lenkt das Makro, wenn es etwas Unvorhergesehenes tut.

Ein vollständiges, gut gebautes Makro, das eine wesentliche Funktion erbringt, wird höchstwahrscheinlich alle diese Merkmale der OOo-Programmierung erfüllen. Welche spezifischen Elemente aus dem OOo-Fundus für den Aufbau eines spezifischen Programms genutzt werden, hängt von der Anwendung ab, vom angestrebten logischen Verhaltens des Programms und vom besten Ermessen des Programmierers. Ein wesentlicher Teil des erfolgreichen Programmierens besteht darin, Erfahrung zu gewinnen, um die Ideen dieses Kapitels möglichst wirkungsvoll anzuwenden.

4. Numerische Routinen

In diesem Kapitel werden die Subroutinen und Funktionen vorgestellt, die Basic im Zusammenhang mit Zahlen bereitstellt – eingeschlossen mathematische Funktionen, Konvertierungsroutinen, Formatierung von Zahlen als String sowie Zufallszahlen. Außerdem werden in diesem Kapitel auch alternative Zahlensysteme behandelt.

Numerische Subroutinen und Funktionen sind Routinen, die mathematische Operationen durchführen. Wenn Sie schon einmal eine Tabellenkalkulation genutzt haben, sind Sie möglicherweise schon vertraut mit mathematischen Funktionen wie „SUMME“, mit der Gruppen von Zahlen addiert werden, oder vielleicht sogar mit „IKV“, womit der interne Zinsfuß einer Investition berechnet wird. Die von Basic bereitgestellten numerischen Routinen (s. Tabelle 24) sind in ihrer Form einfacher und arbeiten normalerweise mit nur einem oder zwei Argumenten und nicht mit ganzen Zahlengruppen.

Tabelle 24. *Mit Zahlen und numerischen Operationen verbundene Subroutinen und Funktionen.*

Funktion	Beschreibung
Abs(Zahl)	Absolutwert der Zahl.
Atn(Zahl)	Der Winkel, in rad, dessen Tangens die angegebene Zahl ist, im Bereich -Pi/2 bis Pi/2.
CByte(Ausdruck)	Rundet den numerischen oder String-Ausdruck zu einem Byte.
CCur(Ausdruck)	Konvertiert den numerischen oder String-Ausdruck zum Typ Currency.
CDbl(Ausdruck)	Konvertiert den numerischen oder String-Ausdruck zum Typ Double.
CDec(Ausdruck)	Konvertiert den numerischen oder String-Ausdruck zum Typ Decimal (nur unter Windows).
CInt(Ausdruck)	Rundet den numerischen oder String-Ausdruck zur nächsten Ganzzahl.
CLng(Ausdruck)	Rundet den numerischen oder String-Ausdruck zum nächsten Long-Wert.
Cos(Zahl)	Kosinus des angegebenen Winkels (in rad).
CSng(Ausdruck)	Konvertiert den numerischen oder String-Ausdruck zum Typ Single.
Exp(Zahl)	Potenziert die Basis des natürlichen Logarithmus um die angegebene Zahl.
Fix(Zahl)	Schneidet den Dezimalteil der Zahl ab.
Format(Zahl, Format)	Benutzerdefinierte Zahlenformatierung, s. Kapitel 7. String-Routinen.
Hex(Zahl)	Gibt die hexadezimale Darstellung der Zahl als String zurück.
Int(Zahl)	Rundet die Zahl in Richtung minus Unendlich. Gibt einen Double-Wert zurück.
Log(Zahl)	Der natürliche Logarithmus der Zahl. In Visual Basic .NET kann diese Methode überladen werden, um entweder den natürlichen Logarithmus (Basis e) oder den Logarithmus zu einer bestimmten Basis zurückzugeben.
Oct(Zahl)	Gibt die oktale Darstellung der Zahl als String zurück.
Randomize(Zahl)	Initialisiert den Zufallsgenerator. Wird die Zahl weggelassen, wird der aktuelle Wert des Systemzeitgebers verwendet.
Rnd	Zufallszahl als Double zwischen 0 und 1.
Round(Zahl, Nachkommastellen	Gibt eine auf eine angegebene Anzahl von Nachkommastellen gerundete Zahl zurück. Benötigt CompatibilityMode(True).
Sgn(Zahl)	Ganzzahl, die dem Vorzeichen der Zahl entspricht (-1, 0, 1).
Sin(Zahl)	Sinus des angegebenen Winkels (in rad).
Sqr(Zahl)	Quadratwurzel der Zahl.
Str(Zahl)	Konvertiert die Zahl zu einem String (ohne Lokalisierung).
Tan(Zahl)	Tangens des angegebenen Winkels (in rad).
Val(Text)	Konvertiert den String zu einer Zahl vom Typ Double. Sehr tolerant bei nicht-numerischem Text.

Die mathematischen Funktionen dieses Kapitels sind gut bekannt und vertraut bei Mathematikern, Ingenieuren und anderen, die jede Gelegenheit nutzen, im täglichen Leben Rechenformeln einzusetzen. Wenn Sie nicht dazugehören – wenn Sie den Rechenschieber vielleicht *nicht* für die coolste Erfindung seit dem Schnittbrot halten –, dann geraten Sie aber bitte nicht in Panik, wenn die Beschreibung naturgemäß ins Mathematische übergeht. Ich bemühe mich, die Information verständlich zu halten und gleichzeitig jenen tiefer gehende Informationen zu bieten, die sich dafür interessieren. Die Routinen sind thematisch in Unterkapitel gruppiert, so dass Sie die Unterkapitel überschlagen können, von denen Sie wissen, dass Sie sie nicht brauchen.

Die numerischen Routinen arbeiten mit numerischen Daten. Basic versucht vor der eigentlichen Berechnung, die Argumente in den passenden Typ zu konvertieren. Es ist aber sicherer, mit den in diesem Kapitel vorgestellten Konvertierungsfunktionen die Datentypen explizit zu konvertieren, als sich auf das Standardverfahren zu verlassen. Wenn ein ganzzahliges Argument erwartet wird und eine Fließkommazahl vorliegt, wird die Zahl standardmäßig gerundet. Im Beispiel „16.8 Mod 7“ wird 16.8 vor der Berechnung zu 17 aufgerundet. Bei dem Operator zur ganzzahligen Division werden die Operanden jedoch beschnitten. Im Beispiel „Print 4 \ 0.999“ wird 0.999 zu 0 beschnitten mit der Folge eines Division-durch-null-Fehlers.

Tipp Tabelle 24 enthält Subroutinen und Funktionen, keine Operatoren wie Mod, + oder \. Operatoren finden Sie im Kapitel 3. Sprachstrukturen.

4.1. Trigonometrische Funktionen

Trigonometrie ist die Lehre von den Eigenschaften der Dreiecke und trigonometrischen Funktionen und ihren Anwendungen. Die Behandlung trigonometrischer Funktionen bezieht sich normalerweise auf Dreiecke mit einem 90-Grad-Winkel, so genannte rechtwinklige Dreiecke (s. Bild 41). Es gibt einen Satz fester Beziehungen zwischen den trigonometrischen Funktionen, den Seitenlängen eines rechtwinkligen Dreiecks und den entsprechenden Winkeln. Wenn Sie diese Beziehungen kennen, können Sie die trigonometrischen Funktionen benutzen, um trigonometrische Aufgaben zu lösen.

In der Praxis benötigen Sie Trigonometrie, wenn Sie den Abstand zu einem Stab oder Turm bekannter Höhe schätzen wollen. Sie messen den Beobachtungswinkel vom Erdboden zur Spitze des Stabs, und da die Höhe des Stabs bekannt ist, errechnet sich Ihr Abstand zum Stab aus der Stabhöhe dividiert durch den Tangens des gemessenen Winkels. Dieses Verfahren eignet sich für Golf, Segeln oder Wandern, immer da, wo Sie den Abstand zu einem Fixpunkt bestimmen wollen (die Golffahne zum Beispiel oder ein Funkturm).

Die grundlegenden trigonometrischen Funktionen sind Sinus, Kosinus und Tangens. Jede ist definiert als das Verhältnis zweier Seiten eines rechtwinkligen Dreiecks. Die Werte dieser Funktionen für jede Größe des Winkels x entsprechen den Quotienten der Seiten eines rechtwinkligen Dreiecks mit diesem Winkel x. Für einen gegebenen Winkel bestimmen die trigonometrischen Funktionen die Seitenlängen eines rechtwinkligen Dreiecks. Gleichermaßen kann man, wenn die Längen von zwei beliebigen Seiten eines rechtwinkligen Dreiecks bekannt sind, mit Hilfe der inversen trigonometrischen Funktionen die Größe des Winkels berechnen.

Basic benutzt das Bogenmaß rad (Radiant) als Maßeinheit für Winkel. Die meisten wissenschaftlichen Laien denken jedoch in der Einheit Grad. Ein Winkel von 90 Grad, so die Ecke eines Quadrats, hat Pi/2 rad.

Tipp Die eingebaute Konstante Pi hat den Wert von etwa 3,1415926535897932385. Pi ist eine fundamentale Konstante, die in wissenschaftlichen Berechnungen weit verbreitet ist. Sie ist definiert als Verhältnis des Umfangs eines Kreises zu seinem Durchmesser. Die Winkelsumme im Dreieck – natürlich auch im rechtwinkligen Dreieck – ist 180 Grad oder Pi rad. Eine gewaltige Menge eleganter und praktischer mathematischer Methoden beruht auf dieser Beziehung zwischen Dreieck und Kreis. Alle Beschreibungen periodischer Bewegung fußen auf dieser Grundlage. Es gilt:Trigonometrie ist eine fundamentale und äußerst nützliche Gruppe mathematischer Hilfsmittel.

Mit Hilfe der Beziehung zwischen Grad (engl. degree) und rad (engl. radian) ist es einfach, zwischen beiden Winkelmaßeinheiten zu konvertieren.

```
degrees = (radians * 180) / Pi
radians = (degrees * Pi) / 180
```

Um den Sinus eines Winkels von 45 Grad zu berechnen, müssen Sie erst den Winkel in rad umrechnen. Das geht so:

```
radians = (45° * Pi) / 180 = Pi / 4 = 3,141592654 / 4 = 0,785398163398
```

Sie können diesen Wert direkt in der trigonometrischen Funktion Sin verwenden.

```
Print Sin(0.785398163398)    ' 0,707106781188
```

Zur Ermittlung des Winkels mit dem Tangens 0,577350269189 dient die Funktion Arkustangens. Der Rückgabewert ist in der Einheit rad, muss also in Grad zurückgerechnet werden.

```
Print Atn(0.577350269189) * 180 / Pi  '29,9999999999731
```

Tipp Rundungsfehler beeinträchtigen diese Beispiele, wie wir später noch sehen. Mit unendlicher Genauigkeit würde das vorige Beispiel eine Antwort von 30 Grad ergeben anstatt 29,9999999999731.

Die Antwort ist nahe 30 Grad. Das Dreieck im Bild 41 hilft uns, die trigonometrischen Funktionen zu erläutern.

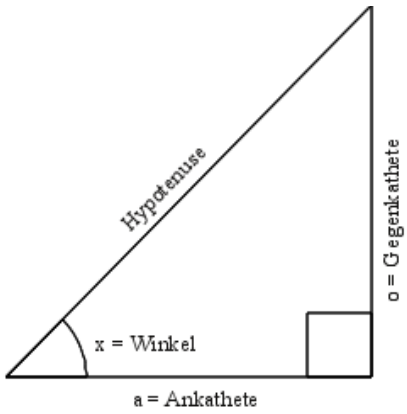


Bild 41. Ein rechtwinkliges Dreieck hat einen 90-Grad-Winkel.

Tabelle 25. *Trigonometrische Funktionen in Basic.*

StarBasic	VB	VB .NET	Rückgabewert
Atn	Atn	Math.Atan	Der Winkel, in rad, dessen Tangens die angegebene Zahl ist, im Bereich -Pi/2 bis Pi/2.
Cos	Cos	Math.Cos	Kosinus des angegebenen Winkels (in Radiant).
Sin	Sin	Math.Sin	Sinus des angegebenen Winkels (in Radiant).
Tan	Tan	Math.Tan	Tangens des angegebenen Winkels (in Radiant).

Tabelle 25 zeigt die von Basic unterstützten trigonometrischen Funktionen, bildlich dargestellt in dem rechtwinkligen Dreieck im Bild 41. Sie erwarten einen Ausdruck als einziges Argument, dessen Wert vor der Berechnung in eine Zahl doppelter Genauigkeit (Typ Double) konvertiert wird.

- Cos(x) = Ankathete / Hypotenuse
- Sin(x) = Gegenkathete / Hypotenuse
- Tan(x) = Gegenkathete / Ankathete = Sin(x) / Cos(x)
- Atn(Gegenkathete / Ankathete) = x

Der Code in Listing 74 löst eine Reihe von geometrischen Aufgaben mit Hilfe der trigonometrischen Funktionen. Der Code geht von einem rechtwinkligen Dreieck aus (s. Bild 41) mit der Länge 3 für die Gegenkathete und der Länge 4 für die Ankathete. Der Tangens wird leicht mit 3/4 errechnet, und die Funktion Atn berechnet die Größe des Winkels. Dazu kommen noch ein paar andere Berechnungen, wie die Bestimmung der Länge der Hypotenuse mit Hilfe der Funktionen Sin einerseits und Cos andererseits. Siehe auch Bild 42.

Listing 74. *Trigonometrisches Beispiel.*

```
Sub ExampleTrigonometric
  Dim OppositeLeg As Double      REM Gegenkathete
  Dim AdjacentLeg As Double      REM Ankathete
  Dim Hypotenuse As Double
  Dim AngleInRadians As Double   REM Winkel in rad
  Dim AngleInDegrees As Double   REM Winkel in Grad
  Dim s As String
  OppositeLeg = 3
  AdjacentLeg = 4
  AngleInRadians = Atn(3 / 4)
  AngleInDegrees = AngleInRadians * 180 / Pi
  s = "Gegenkathete = " & OppositeLeg & Chr$(10) &_
      "Ankathete = " & AdjacentLeg & Chr$(10) &_
      "Winkel in Grad von ATN = " & AngleInDegrees & Chr$(10) &_
      "Hypotenuse von COS = " & AdjacentLeg/Cos(AngleInRadians) & Chr$(10) &_
      "Hypotenuse von SIN = " & OppositeLeg/Sin(AngleInRadians) & Chr$(10) &_
      "Gegenkathete von TAN = " & AdjacentLeg * Tan(AngleInRadians)
  MsgBox s, 0, "Trigonometrische Funktionen"
End Sub
```

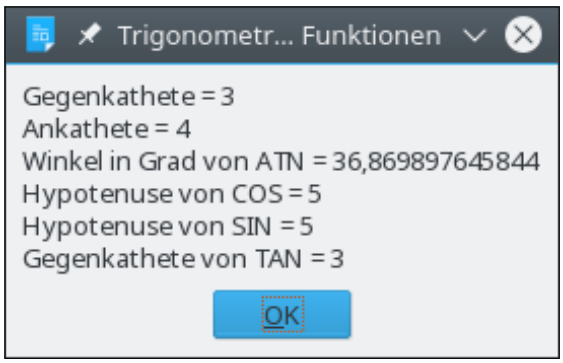


Bild 42. *Mit den trigonometrischen Funktionen werden Dreiecksaufgaben gelöst.*

4.2. Rundungsfehler und Genauigkeit

Ein Computer oder ein Taschenrechner führt numerische Rechnungen mit nur einer begrenzten Anzahl von Ziffern aus. So entstehen Rundungsfehler. Dieses Problem besteht nicht bei Ganzzahlen.

Die Zahl 1/3 wird in dezimaler Form zu 0,33333333, aber eigentlich gehörte eine unendliche Anzahl von Dreien hinter das Dezimalkomma. Mit vier Genauigkeitsstellen wird die Zahl als 0,3333 geschrieben. Das ist ungenau in der Darstellung und den Rechenergebnissen.

```
1/3 + 1/3 + 1/3 = 3/3 = 1          'Das korrekte Ergebnis ist 1
0,3333 + 0,3333 + 0,3333 = 0,9999  'Das Ergebnis mit begrenzter Genauigkeit,
                                   'ein wenig daneben
```

Das einfache Makro in Listing 75 zeigt das Problem. Der Wert 0,2 wird so oft zur Variablen *num* addiert, bis sie den Wert 5 erreicht. Wenn wir unendliche Genauigkeit hätten oder wenn der Rechner die Zahl 0,2 intern exakt repräsentierte, wäre die Schleife beendet, wenn die Variable *num* den Wert 5 erhält. Die Variable wird jedoch niemals genau gleich dem Wert 5 sein, daher wird die Schleife nie beendet. Der Wert 5 wird zwar ausgegeben, aber nur, weil die Anweisung Print den Wert 4,9999999 für die Ausgabe auf 5 rundet.

Listing 75. *Rundungsfehler und begrenzte Genauigkeit verhindern das Schleifenende.*

```
Dim num As Single
Do
  num = num + 0.2
  If num > 4.5 Then Print num  'Ausgabe: 4,6, 4,8, 5, 5,199999...
Loop Until num = 5.0
Print num
```

Im Rechner wirken komplexe Rundungsalgorithmen im Bestreben, die Auswirkungen der begrenzten Genauigkeit zu reduzieren – begrenzte Genauigkeit bedeutet, dass zur Darstellung einer Zahl nur eine begrenzte Menge an Ziffern und Speicherplatz zur Verfügung steht. Trotz dieser unterstützenden Maßnahmen zeigt Listing 75 deutlich, dass die internen komplexen Rundungsalgorithmen das Problem nicht beheben. Wenn Sie zwei Fließkommazahlen auf Gleichheit prüfen, ist es sicherer, sie mit einem Wertebereich zu vergleichen. Der Code in Listing 76 stoppt die Schleife, wenn die Variable größer als oder gleich 5 ist.

Listing 76. *Vermeidung von Rundungsfehlern durch >= (größer als oder gleich).*

```
Dim num As Single
Do
  num = num + 0.2
Loop Until num >= 5.0
Print num  '5,199999
```

Der Code in Listing 76 funktioniert irgendwie, aber Sie wollen möglicherweise, dass die Schleife endet, wenn die Variable *num* den Wert 4,9999999 hat, und nicht bei 5,1999999. Das kann man erreichen, indem man prüft, ob die zwei Zahlen nahe beieinander anstatt gleich sind. Die große Frage dabei ist, wie nahe die zwei Zahlen einander sein müssen, dass Sie als gleich gewertet werden. Aufgrund Ihrer Kenntnis der vorliegenden Aufgabe können Sie normalerweise eine einfache Schätzung vornehmen. Variablen vom Typ Single können etwa 8 genaue Stellen enthalten, die vom Typ Double etwa 16. Versuchen Sie nicht, von den Variablen eine höhere Genauigkeit zu erwarten, als sie leisten können. Der Code in Listing 76 verwendet Variablen einfacher Genauigkeit (Single), womit Sie eine Genauigkeit von ungefähr sieben Stellen erwarten können. Der Code in Listing 77 gibt den Unterschied zwischen 5 und *num* aus – beachten Sie, dass etwa sechs Stellen korrekt sind.

Listing 77. *Vergleich der Variablen mit einem Wertebereich.*

```
Dim num As Single
Do
  num = num + 0.2
Loop Until 4.99999 < num And num < 5.00001
Print 5 - num  '4,76837158203125E-07  = 0,000004768371582031 25
```

Die Funktion Abs gibt den Absolutwert einer Zahl zurück. Mit ihrer Hilfe vereinfachen Sie den Prüfvorgang, ob eine Zahl sehr nahe bei einer anderen ist.

```
If Abs(num - 5) < 0.00001 Then
```

Mit Abs und Subtraktion finden Sie heraus, wie nahe zwei Zahlen einander sind, aber das mag nicht ausreichend sein. Das Licht bewegt sich zum Beispiel 299.792.458 Meter pro Sekunde. Diese Zahl hat 9 Ziffern. Eine Zahl mit einfacher Genauigkeit ist nur bis zu etwa sieben Stellen genau. Siehe Listing 78.

Listing 78. *Variablen vom Typ Single sind nur auf sieben oder acht Stellen genau.*

```
Dim c1 As Single  'In der Wissenschaft wird der Buchstabe c als Zeichen
Dim c2 As Single  'für die Lichtgeschwindigkeit verwendet.
c1 = 299792458    'Lichtgeschwindigkeit in Metern pro Sekunde: 9 Stellen
c2 = c1 + 16      'Addition von 16 zur Lichtgeschwindigkeit
If c1 = c2 Then   'Beide sind gleich, weil nur die ersten sieben
  Print "Gleich"  'oder acht Stellen signifikant sind.
End If
```

Der Code in Listing 78 addiert zur Lichtgeschwindigkeit 16 hinzu, doch das ändert den Wert nicht, und zwar, weil nur die ersten sieben oder acht Ziffern signifikant sind. Der Code in Listing 79 verwendet eine Zahl, die zwar in der Größenordnung kleiner ist, aber dieselbe Anzahl an Ziffern hat. Die Addition von 1 würde eine signifikante Stelle ändern, aber die Addition einer viel kleineren Zahl führt wiederum dazu, dass die Zahlen gleich sind.

Listing 79. *Variablen vom Typ Single sind nur auf sieben oder acht Stellen genau.*

```
Dim c1 As Single   'In der Wissenschaft wird der Buchstabe c als Zeichen
Dim c2 As Single   'für die Lichtgeschwindigkeit verwendet.
c1 = 299.792458    'Dies sind neun Ziffern, aber es ist nicht die Lichtgeschwindigkeit
c2 = c1 + .0000016 'Man muss eine kleine Zahl addieren, damit beide gleich bleiben
If c1 = c2 Then    'Beide sind gleich, weil nur die ersten sieben
  Print "Gleich"   'oder acht Stellen signifikant sind.
End If
```

Fließkommazahlen können in ihrer Größe sehr unterschiedlich sein – das betrifft die reine Größe der Zahl auf der Zahlenskala, aber nur unwesentlich die Anzahl an relevanten Stellen. Beim Test, ob zwei Zahlen ungefähr gleich sind, können sich große Zahlen um einen größeren Betrag unterscheiden als kleine Zahlen. Die größte akzeptierte Differenz hängt von der reinen Zahlengröße ab. Mathematiker nennen das den „relativen Fehler“. Siehe Listing 80.

Listing 80. *Vergleich zweier Zahlen.*

```
REM Die Zahl n1 ist von primärem Interesse
REM n2 wird mit n1 relativ verglichen
REM rel_diff ist die gewünschte relative Differenz
REM rel_diff wird als nicht negativ angenommen
Function AreSameNumber(n1, n2, rel_diff) As Boolean
  AreSameNumber = False               'Ausgangsannahme: sie unterscheiden sich.
  If n1 <> 0 Then                     'Keine Division durch n1, wenn n1 den Wert 0 hat.
    If Abs((n1 - n2) / n1) <= rel_diff Then 'Division der Differenz durch n1
      AreSameNumber = True                  'für den relativen Vergleich.
    End If                             'Wenn n1, die Zahl, um die es geht, null ist,
  ElseIf Abs(n2) <= rel_diff Then       'dann wird rel_diff mit n2 verglichen.
    AreSameNumber = True
  End If
End Function
```

Der Code in Listing 80 dividiert den Unterschied zweier Zahlen durch eine der beiden. Der Code in Listing 81 prüft, ob Zahlen unterschiedlicher Größe dieselbe Zahl sind.

Listing 81. *Test, ob die Zahlen gleich sind.*

```
Sub CheckSameNumber
  Dim s1 As Single
  Dim s2 As Single
  Print AreSameNumber(299792458, 299790000, 1e-5)   'True: fünf gleiche Ziffern
  Print AreSameNumber(299792458, 299700000, 1e-5)   'False: vier gleiche Ziffern
  s1 = 299792458                                    's1 erhält einen anderen Wert
  s2 = 299792448                      'zugewiesen als s2, ist aber dieselbe Zahl.
  Print AreSameNumber(s1, s2, 0.0)   'True: dieselbe Zahl in einfacher Genauigkeit.
  Print AreSameNumber(299.792458, 299.790000, 1e-5)'True: fünf gleiche Ziffern
  Print AreSameNumber(2.99792458, 2.99700000, 1e-5)'False: vier gleiche Ziffern
End Sub
```

Es ist viel geschrieben und geforscht worden über die negativen Aspekte im Zusammenhang mit Fließkommazahlen. Eine vollständige Behandlung dieses Themas übersteigt daher bei weitem den Rahmen dieses Buches. Für den Normalgebrauch sind die Probleme kaum so störend, aber wenn sie auftauchen, können sie sehr verwirrend sein, wenn Sie sich der Problematik nicht bewusst sind.

In diesem Zusammenhang steht die Funktion Round, die eine auf eine angegebene Anzahl von Nachkommastellen gerundete Zahl als Typ Double zurückgibt. Der erste Parameter ist der zu rundende numerische Ausdruck. Der optionale zweite Parameter ist ein Integerwert, der festlegt, wie viele Stellen rechts der Dezimalstelle in die Rundung einbezogen werden sollen. Standard ist 0. Das folgende Beispiel stammt aus der deutschen Hilfe. Die Funktion benötigt CompatibilityMode(True).

```
Sub ExampleRound
  CompatibilityMode(True)
  Dim r
  r = Pi
  Print r           '3,14159265358979
  Print Round(r, 5) '3,14159
  r = exp(1)
  Print r           '2,71828182845904
  Print Round(r)    '3
End Sub
```

4.3. Mathematische Funktionen

Die mathematischen Basic-Funktionen erwarten ein numerisches Argument. Vor der Berechnung werden alle Standardtypen zu Double konvertiert. Strings dürfen aus Hexadezimal- und Oktalzahlen bestehen. Die Funktionen sind dieselben wie in Visual Basic (s. Tabelle 26).

Tabelle 26. *Mathematische Funktionen in Basic.*

StarBasic	VB	VB .NET	Rückgabewert
Abs	Abs	Math.Abs	Absolutwert der angegebenen Zahl.
Exp	Exp	Math.Exp	Potenziert die Basis des natürlichen Logarithmus um die angegebene Zahl.
Log	Log	Math.Log	Der natürliche Logarithmus der angegebenen Zahl. In Visual Basic .NET kann diese Methode überladen werden, um entweder den natürlichen Logarithmus (Basis e) oder den Logarithmus zu einer bestimmten Basis zurückzugeben.
Sgn	Sgn	Math.Sign	Ganzzahl, die dem Vorzeichen der angegebenen Zahl entspricht (-1, 0, 1).
Sqr	Sqr	Math.Sqrt	Quadratwurzel der angegebenen Zahl.

Mit der Funktion Abs bestimmen Sie den Absolutwert einer Zahl. Sie können sich das so vorstellen, dass einfach das Vorzeichen (+ oder -) von der Zahl abgetrennt wird. Die geometrische Definition von Abs(x) ist der Abstand von x zu 0 auf einer geraden Linie.

```
Abs(23.33) = 23.33
Abs(-3)    = 3
Abs("-1")  = 1  'Beachten Sie, dass der String "-1" zu Double konvertiert wird.
```

Die Funktion Sgn bestimmt das Vorzeichen einer Zahl. Es wird ein Integer-Wert -1, 0 oder 1 zurückgegeben, wenn die Zahl entsprechend negativ, null oder positiv ist.

```
Sgn(-37.4) = -1
Sgn(0)     =  0
Sgn("4")   =  1
```

Die Quadratwurzel von 9 ist 3, denn 3 mal 3 ergibt 9. Mit der Funktion Sqr erhalten Sie die Quadratwurzel einer Zahl. Die Funktion Sqr kann keine Quadratwurzel einer negativen Zahl errechnen – der Versuch endet in einem Laufzeitfehler.

```
Sqr(100) = 10
Sqr(16)  = 4
Sqr(2)   = 1.414213562371
```

Logarithmen wurden von John Napier erfunden, der von 1550 bis 1617 lebte. Napier entwickelte Logarithmen, um arithmetische Berechnungen dadurch zu vereinfachen, dass Addition und Subtraktion an die Stelle von Multiplikation und Division traten. Logarithmen haben folgende Eigenschaften:

```
Log(x*y) = Log(x) + Log(y)
Log(x/y) = Log(x) - Log(y)
Log(x^y) = y * Log(x)
```

Die Funktion Exp ist die Umkehrung der Funktion Log. Zum Beispiel ist Exp(Log(4)) = 4 und Log(Exp(2)) = 2. Konzeptionell wandeln Logarithmen Multiplikationsaufgaben in Additionsaufgaben. So kann man Logarithmen in ihrer ursprünglichen Gestaltung verwenden.

```
Print Exp(Log(12) + Log(3)) '36 = 12 * 3
Print Exp(Log(12) - Log(3)) ' 4 = 12 / 3
```

Logarithmen sind durch die Gleichung y=b^x definiert. Daher sagt man: der Logarithmus Basis b von y ist x. Zum Beispiel mit dem Logarithmus Basis 10: 10^2 = 100. Der Logarithmus Basis 10 von 100 ist also 2. Häufig wird der Logarithmus mit der Basis e (ungefähr e=2,71828182845904523536) verwendet, weil er ein paar nette mathematische Eigenschaften besitzt. Dies ist der „natürliche Logarithmus“ und wird in Basic genutzt. Visual Basic .NET ermöglicht Ihnen, Logarithmen auf anderer Basis zu berechnen. Das kann jedoch einfach mit der Formel erreicht werden, wonach der Logarithmus Basis b durch Log(x)/Log(b) bestimmt ist, ungeachtet der Basis des aktuell genutzten Logarithmus.

Logarithmen spielen heute als allgemeine Rechenvereinfachungen nicht mehr die Rolle, angesichts der Leistungsfähigkeit heutiger Rechner. Allerdings beschreiben Logarithmen das Verhalten vieler natürlicher Phänomene. Zum Beispiel wird das Bevölkerungswachstum oft mit Logarithmen dargestellt, denn geometrisches Wachstum zeigt sich in einer logarithmischen Kurvendarstellung als Gerade. Exponentielle und logarithmische Kurven werden auch ausgiebig im Ingenieurswesen bei Berechnungen genutzt, die das dynamische Verhalten elektrischer, mechanischer und chemischer Systeme beschreiben.

Das Makro in Listing 82 berechnet den Logarithmus der Zahl x (erstes Argument) zur angegebenen Basis b (zweites Argument). Zum Beispiel berechnen Sie mit LogBase(8, 2) den Logarithmus Basis 2 von 8 (Ergebnis 3).

Listing 82. *LogBase.*

```
Function LogBase(x, b) As Double
  LogBase = Log(x) / Log(b)
End Function
```

4.4. Numerische Konvertierungen

Basic versucht, Argumente vor der Operation zu einem passenden Typ zu konvertieren. Es ist jedoch sicherer, die Datentypen explizit mit Hilfe von Konvertierungsfunktionen – präsentiert in diesem Kapitel – umzuwandeln, als sich auf das Standardverhalten zu verlassen, denn es könnte nicht das Gewünschte erbringen. Wenn ein Integer-Argument benötigt wird und eine Fließkommazahl bereit steht, wird die Zahl standardmäßig gerundet. Bei „16.8 Mod 7“ zum Beispiel wird vor der Berechnung 16.8 zu 17 aufgerundet. Der Operator für die Ganzzahldivision schneidet die Operanden jedoch ab. Bei „Print 4\0.999“ zum Beispiel wird 0.999 zu 0 und verursacht einen Division-durch-null-Fehler.

Es gibt viele verschiedene Methoden und Funktionen zur Konvertierung zu numerischen Typen. Die wesentlichen Konvertierungsfunktionen wandeln Zahlen um, die als Strings gemäß dem lokalen Gebietsschema vorliegen. Die Konvertierungsfunktionen in Tabelle 27 wandeln jeden String oder numerischen Ausdruck in eine Zahl um. Hexadezimal- oder Oktalzahlen in Strings müssen in der Basic-Standardnotation vorliegen. Zum Beispiel muss die hexadezimale Zahl 2A als "&H2A" angegeben werden.

Tabelle 27. *Konvertierung zu einem numerischen Typ.*

Funktion	Typ	Beschreibung
CByte(Ausdruck)	Byte	Rundet den numerischen oder String-Ausdruck zu einem Byte.
CCur(Ausdruck)	Currency	Konvertiert den numerischen oder String-Ausdruck zum Typ Currency. Für Dezimaltrenner und Währungssymbole gilt das lokale Gebietsschema.
CDec(Ausdruck)	Decimal	Konvertiert den Ausdruck zum Typ Decimal (nur unter Windows).
CInt(Ausdruck)	Integer	Rundet den numerischen oder String-Ausdruck zum nächsten Integer-Wert.
CLng(Ausdruck)	Long	Rundet den numerischen oder String-Ausdruck zum nächsten Long-Wert.
CDbl(Ausdruck)	Double	Konvertiert den numerischen oder String-Ausdruck zum Typ Double.
CSng(Ausdruck)	Single	Konvertiert den numerischen oder String-Ausdruck zum Typ Single.

Die Funktionen, die eine Ganzzahl zurückgeben, verhalten sich alle ähnlich. Numerische Ausdrücke werden gerundet und nicht abgeschnitten. Ein String-Ausdruck, der keine Zahl enthält, wird mit 0 gewertet. Es wird nur der Teil des Strings ausgewertet, der eine Zahl enthält. Siehe Listing 83.

Listing 83. *CInt und CLng ignorieren nicht-numerische Werte.*

```
Sub CintIgnoreNonNum
  Print CInt(12.2)       ' 12
  Print CLng("12.5")     ' 13
  Print CInt("xxyy")     '  0
  Print CLng("12.1xx")   ' 12
  Print CInt(-12.2)      '-12
  Print CInt("-12.5")    '-13
  Print CLng("-12.5xx")  '-13
End Sub
```

Achtung Sollte die Ausgabe bei Ihnen immer nur 12 oder -12 sein, dann liegt das wahrscheinlich an einem Bug, der in der einen oder anderen Linux-Umgebung vorkommt (s. Bug 70236 - CDbl truncates values with locale setting passed by an environment variable). Eine Ausweichlösung ist der Einsatz der Funktion Val, zum Beispiel:

```
Print CInt(Val("12.5"))
```

Die Funktion Val kennt aber kein lokales Gebietsschema. Der String muss als Dezimaltrenner einen Punkt haben. Siehe auch Listing 86.

CLng und CInt verhalten sich ähnlich, aber nicht ganz gleich bei verschiedenen Arten von Überlaufbedingungen. Zu große dezimale Zahlen in Strings verursachen einen Laufzeitfehler. Es gibt zum

Beispiel bei CInt("40000") und CLng("999999999999") einen Laufzeitfehler, aber nicht bei CLng("40000"). CLng verursacht nie einen Überlauf bei zu großen Hexadezimal- oder Oktalzahlen, es wird in stillschweigender Duldung null zurückgegeben. CInt interpretiert jedoch Hexadezimal- und Oktalzahlen als Long und konvertiert sie dann zu Integer, mit dem Ergebnis, dass ein gültiger Long-Wert beim Konvertieren zu Integer einen Laufzeitfehler auslöst. Ein hexadezimaler Wert aber, der zu groß für einen gültigen Long-Wert ist, wird anstandslos zu null und dann zu einem Integer umgewandelt (s. Listing 84).

Listing 84. *CInt wertet die Zahl als Long und konvertiert dann zu Integer.*

```
Sub CIntFromHex
  Print CLng("&HFFFFFFFFFFE")   '0  Überlauf bei Long
  Print CInt("&HFFFFFFFFFFE")   '0  Überlauf bei Long,
                                '   danach Konvertierung zu Integer
  Print CLng("&HFFFFE")         '1048574
  Print CInt("&HFFFFE")         'Laufzeitfehler, konvertiert zu Long, danach Überlauf
End Sub
```

Der Code in Listing 85 konvertiert eine Reihe von hexadezimalen Zahlen mit CLng zu Long. Zur Erläuterung der Ausgaben von Listing 85 s. Tabelle 28.

Listing 85. *Beispiel für CLng mit Hexadezimalzahlen.*

```
Sub ExampleCLngWithHex
  On Error Resume Next
  Dim s$, i%
  Dim v()
  v() = Array("&HF",      "&HFF",      "&HFFF",      "&HFFFF",_
              "&HFFFFF", "&HFFFFFF", "&HFFFFFFF", "&HFFFFFFFF",_
              "&HFFFFFFFFF",_
              "&HE",      "&HFE",      "&HFFE",      "&HFFFE",_
              "&HFFFFE", "&HFFFFFE", "&HFFFFFFE", "&HFFFFFFFE",_
              "&HFFFFFFFFE")
  For i = LBound(v()) To UBound(v())
    s = s & i & " CLng(" & v(i) & ") = "
    s = s & CLng(v(i))
    s = s & Chr$(10)
  Next
  MsgBox s
End Sub
```

Tabelle 28. *CLong mit Hexadezimalzahlen: Ausgaben von Listing 85 mit Erläuterungen.*

Eingabe	CLng	Erläuterung
F	15	Korrekter hexadezimaler Wert.
FF	255	Korrekter hexadezimaler Wert.
FFF	4095	Korrekter hexadezimaler Wert.
FFFF	65535	Korrekter hexadezimaler Wert.
FFFFF	1048575	Korrekter hexadezimaler Wert.
FFFFFF	16777215	Korrekter hexadezimaler Wert.
FFFFFFF	268435455	Korrekter hexadezimaler Wert.
FFFFFFFF	-1	Korrekter hexadezimaler Wert. Erzeugte früher einen Laufzeitfehler.
FFFFFFFFF	0	Überlauf gibt null zurück, es sind neun hexadezimale Ziffern.
E	14	Korrekter hexadezimaler Wert.

Eingabe	CLng	Erläuterung
FE	254	Korrekter hexadezimaler Wert.
FFE	4094	Korrekter hexadezimaler Wert.
FFFE	65534	Korrekter hexadezimaler Wert.
FFFFE	1048574	Korrekter hexadezimaler Wert.
FFFFFE	16777214	Korrekter hexadezimaler Wert.
FFFFFFE	268435454	Korrekter hexadezimaler Wert.
FFFFFFFE	-2	Korrekter hexadezimaler Wert. Erzeugte früher einen Laufzeitfehler.
FFFFFFFFE	0	Überlauf gibt null zurück, es sind neun hexadezimale Ziffern.

Wenn Sie Zahlen schreiben, brauchen Sie keine führenden Nullen anzugeben. 3 und 003 sind zum Beispiel ein und dieselbe Zahl. Long Integer kann acht hexadezimale Ziffern enthalten. Wenn nur vier geschrieben werden, können Sie führende Nullen voraussetzen. Wenn die Hexadezimalzahl zu groß für Long ist, wird null zurückgegeben. Die negativen Zahlen sind genauso leicht zu erklären. Intern repräsentiert der Rechner eine negative Zahl durch ein gesetztes erstes Bit. Bei all den Hexadezimalziffern 8, 9, A, B, C, D, E und F ist in der binären Zahlendarstellung das höchste Bit gesetzt. Wenn die erste Hexadezimalziffer ein gesetztes höchstes Bit hat, ist der zurückgegebene Long-Wert negativ. Eine Hexadezimalzahl ist positiv, wenn sie weniger als acht hexadezimale Ziffern hat, und negativ, wenn sie aus acht hexadezimalen Ziffern besteht, deren erste Ziffer 8, 9, A, B, C, D, E oder F ist. Nun, so war es jedenfalls einmal.

Die Funktion CByte verhält sich wie CInt und CLng, wiewohl mit Besonderheiten. Der Rückgabetyp, Byte, wird als Zeichen behandelt, wenn nicht explizit in eine Zahl konvertiert wird. Ein Byte ist ein Kurz-Integer, das nur acht Bit einnimmt anstatt 16 Bit bei Integer.

```
Print CByte("65")          'A hat den ASCII-Wert 65
Print CInt(CByte("65xx")) '65 wird direkt zu einer Zahl konvertiert.
```

Tipp

Ein Integer ist in VB .NET äquivalent zu einem Long in StarBasic.

VB hat abweichende Rundungsregeln. Zahlen werden zur nächsten geraden Zahl hin gerundet, wenn der Dezimalteil exakt 0,5 ist. Das nennt man IEEE-Rundung.

Die Funktionen, die eine Fließkommazahl zurückgeben, verhalten sich alle ähnlich. Numerische Ausdrücke werden zu dem nächstliegenden Wert konvertiert. Strings mit nicht-numerischen Anteilen lösen einen Laufzeitfehler aus. Zum Beispiel ergibt CDbl("13.4e2xx") einen Laufzeitfehler. CDbl und CSng lösen einen Laufzeitfehler aus für zu große Hexadezimal- und Oktalzahlen.

Listing 86. *CSng und CDbl mit String-Argumenten.*

```
Sub CDblAndCSng
  On Error GoTo ErrorHandler
  Dim s$
  s = "CDbl(12.2) = "
  s = s & CDbl(12.2)       ' 12.2
  s = s & Chr$(10) & "CSng(""12,55e1"") = "
  s = s & CSng("12,55e1") ' 125.5
  s = s & Chr$(10) & "CDbl(""-12,2e-1"") = "
  s = s & CDbl("-12,2e-1")'-1.22
  s = s & Chr$(10) & "CSng(""-12,5"") = "
  s = s & CSng("-12,5")    '-12.5
  s = s & Chr$(10) & "CDbl(""xxyy"") = "
  s = s & CDbl("xxyy")     ' Laufzeitfehler
  s = s & Chr$(10) & "CSng(""12,1xx"") = "
  s = s & CSng("12,1xx")   ' Laufzeitfehler
  MsgBox s
```

```
  Exit Sub
  ErrorHandler:
    s = s & " Fehler: " & Error
    Resume Next
End Sub
```

Achtung

Anmerkung des Übersetzers: Sollte die Ausgabe bei Ihnen andere Werte zeigen, dann liegt das wahrscheinlich an einem Bug, der in der einen oder anderen Linux-Umgebung vorkommt (s. Bug 70236 - CDbl truncates values with locale setting passed by an environment variable). Eine Ausweichlösung ist der Einsatz der Funktion Val, zum Beispiel:

```
Print CInt(Val("-12.2e-1"))
```

Die Funktion Val kennt aber kein lokales Gebietsschema. Der String muss einen Punkt als Dezimaltrenner haben. Der folgende Code zeigt eine Hilfsfunktion für Gebietsschemata mit Komma als Dezimaltrenner (zum Verständnis der Wirkungsweise s. Abschnitt 5.4. Array zu String und wieder zurück auf der Seite 123 ff).

```
REM Ersetzt die Funktion Val für Zahlenstrings mit Komma als
REM Dezimaltrenner.
Function ValCSD(sCSDNum As String) As Double
  ValCSD = Val(Join(Split(Join(Split(sCSDNum, "."), ""), ","), "."))
End Function
```

Beispiel für ValCSD.

```
Sub ExampleValCSD
  Print CSng(ValCSD("12,55e1")) '125,5
  Print ValCSD("12,55e1")       '125,5
End Sub
```

Die Funktionen CDbl und CSng scheitern an Strings mit nicht-numerischen Daten, die Funktion Val jedoch nicht. Nehmen Sie die Funktion Val, um einen String, der noch andere Zeichen enthalten kann, zum Typ Double zu konvertieren. Die Funktion Val schaut sich jedes Zeichen in der Zeichenkette an, ignoriert Leerzeichen, Tabulatorschritte und Zeilenumbruchzeichen und stoppt erst, wenn sie auf ein Zeichen stößt, das nicht ein Teil einer Zahl ist. Symbole und Zeichen, die oft als Teil numerischer Werte gelten, wie Dollarzeichen und Kommas, werden nicht erkannt. Die Funktion erkennt allerdings Oktal- und Hexadezimalzahlen, die mit &O (oktal) und &H (hexadezimal) eingeleitet werden.

Die Funktion Val behandelt Leerzeichen anders als die anderen Funktionen. Zum Beispiel gibt Val(" 12 34") die Zahl 1234 zurück, CDbl und CSng produzieren einen Laufzeitfehler, und CInt gibt mit demselben Argument 12 zurück.

Listing 87. *Die Behandlung von Leerzeichen ist unterschiedlich.*

```
Sub NumsAreDifferent
  On Error GoTo ErrorHandler
  Dim s$
  s = "Val(""  12  34"") = "
  s = s & Val("  12    34")
  s = s & Chr$(10) & "CInt(""  12  34"") = "
  s = s & CInt("  12    34")
  s = s & Chr$(10) & "CLng(""  12  34"") = "
  s = s & CLng("  12    34")
  s = s & Chr$(10) & "CSng(""  12  34"") = "
  s = s & CSng("  12    34")
  s = s & Chr$(10) & "CDbl(""  12  34"") = "
  s = s & CDbl("  12    34")
  MsgBox s
  Exit Sub
```

```
  ErrorHandler:
    s = s & " Fehler: " & Error
    Resume Next
End Sub
```

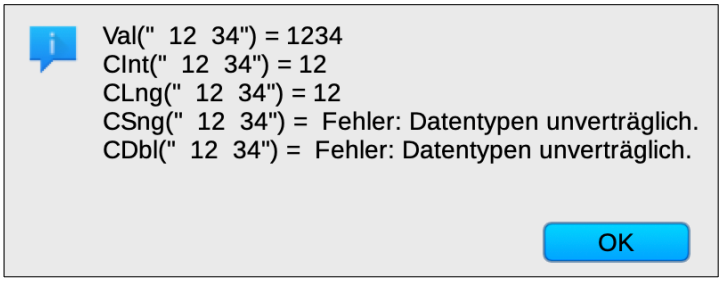


Bild 43. *Leerzeichen werden bei der numerischen Konvertierung nicht einheitlich behandelt.*

Listing 88. *Die Funktion Val ist die Umkehrung der Funktion Str.*

```
Sub ExampleVal
  Print Val(" 12 34")   '1234
  Print Val("12 + 34")  '12
  Print Val("-1.23e4")  '-12300
  Print Val(" &FF")     '0
  Print Val(" &HFF")    '255
  Print Val("&HFFFF")   '-1
  Print Val("&HFFFE")   '-2
  Print Val("&H3FFFE")  '-2, ja, es wird wirklich zu -2 konvertiert
  Print Val("&HFFFFFFFFFFFFFF") '-1
End Sub
```

Die Funktion Val verhält sich beim Erkennen von Hexadezimal- oder Oktalzahlen seltsam genug, so dass ich von einem Bug rede. Intern werden Hexadezimal- und Oktalzahlen zu einem 32 Bit großen Long Integer, und die niedrigstwertigen 16 Bits dann zu einem Integer konvertiert. Das erklärt , warum in Listing 88 die Zahl H3FFFE zu -2 konvertiert wird, weil nämlich nur die niedrigstwertigen 16 Bits erkannt werden – falls Sie es vergessen haben sollten, es sind die ganz rechts angeordneten vier Hexadezimalziffern. Listing 89 demonstriert dieses seltsame Verhalten. Die Ausgabe ist in Tabelle 29 erläutert.

Tipp

Die Funktion Val berücksichtigt bei der Zahlenkonvertierung das lokale Gebietsschema nicht. Als Dezimaltrenner gilt nur der Punkt, das Komma kann als Tausendertrenner dienen, aber niemals rechts vom Dezimaltrenner. Um lokalisierte Zahlen zu konvertieren, verwenden Sie CDbl oder CLng. Falls Sie es vergessen haben sollten: Das Gebietsschema ist eine weitere Methode, auf die Einstellungen zuzugreifen, die die Formatierungen abhängig von einem bestimmten Land beeinflussen. Siehe Listing 88.

Listing 89. *Beispiel für Val mit Hexadezimalzahlen.*

```
Sub ExampleValWithHex
  Dim s$, i%
  Dim l As Long
  Dim v()
  v() = Array("&HF",      "&HFF",      "&HFFF",      "&HFFFF",_
              "&HFFFFF", "&HFFFFFF", "&HFFFFFFF", "&HFFFFFFFF",_
              "&HFFFFFFFFF",_
              "&HE",      "&HFE",      "&HFFE",      "&HFFFE",_
              "&HFFFFE", "&HFFFFFE", "&HFFFFFFE", "&HFFFFFFFE",_
              "&HFFFFFFFFE", "&H111111111", "&H1111")
  For i = LBound(v()) To UBound(v())
    s = s & "Val(" & v(i) & ") = " & Val(v(i)) & Chr$(10)
  Next
```

```
    'Folgendes funktionierte in OOo 2.x,
    'misslingt aber in OOo 3.2.1
    'l = "&H" & Hex(-2)
    s = s & Chr$(10) & "Hex(-1) = " & Hex(-1) & Chr$(10)
    s = s & "Hex(-2) = " & Hex(-2) & Chr$(10)
    's = s & "l = &H" & Hex(-2) & " ==> " & l & Chr$(10)
    MsgBox s
End Sub
```

Tabelle 29. *Val mit Hexadezimalzahlen: Ausgabe vom Listing 89 mit Erläuterungen.*

Eingabe	Ausgabe	Erläuterung
F	15	Hexadezimales F ist 15.
FF	255	Hexadezimales FF ist 255.
FFF	4095	Hexadezimales FFF ist 4095.
FFFF	-1	Hexadezimales FFFF ist -1 für ein 16-Bit-Integer (zwei Bytes).
FFFFF	-1	Nur die zwei Bytes ganz rechts (vier Hex-Ziffern) werden erkannt.
E	14	Hexadezimales E ist 14.
FE	254	Hexadezimales FE ist 254.
FFE	4094	Hexadezimales FFE ist 4094.
FFFE	-2	Hexadezimales FFFE ist -2 für ein 16-Bit-Integer (zwei Bytes).
FFFFE	-2	Nur die zwei Bytes ganz rechts werden erkannt.
FFFFFE	-2	Nur die zwei Bytes ganz rechts werden erkannt.
FFFFFFE	-2	Nur die zwei Bytes ganz rechts werden erkannt.
FFFFFFFE	-2	Nur die zwei Bytes ganz rechts werden erkannt.
HFFFFFFFFE	-2	Nur die zwei Bytes ganz rechts werden erkannt.
111111111	4639	Korrekt, nur die zwei Bytes ganz rechts.
1111	4639	Korrekt.

Val konvertiert Hexadezimalzahlen, nutzt aber nur die zwei Bytes ganz rechts.

Die Funktionen CByte, CInt, CLng, CSng und CDbl konvertieren Zahlen, Strings oder Ausdrücke zu einem spezifischen numerischen Typ. Die Funktionen Int und Fix entfernen den dezimalen Anteil und geben den Typ Double zurück. Ein String-Ausdruck, der keine Zahl enthält, wird als 0 ausgewertet. Es wird nur der Teil berücksichtigt, der eine Zahl enthält, s. Tabelle 30.

Tabelle 30. *Entfernung des dezimalen Anteils einer Fließkommazahl.*

Funktion	Typ	Beschreibung
Int	Double	Rundet die Zahl in Richtung negativ unendlich.
Fix	Double	Schneidet den dezimalen Anteil ab.

Die Funktionen Int und Fix unterscheiden sich nur in der Behandlung negativer Zahlen. Fix entfernt immer den dezimalen Teil, was der Rundung Richtung null entspricht. Int rundet hingegen in Richtung negativ unendlich. Als Beispiel: „Int(12.3)“ wird 12 und „Int(-12.3)“ wird -13.

```
Sub IntAndFix
  Print Int(12.2)      ' 12
  Print Fix(12.2)      ' 12
  Print Int("12.5")    ' 12
  Print Fix("12.5")    ' 12
  Print Int("xxyy")    '  0
```

```
  Print Fix("xxyy")    '  0
  Print Int(-12.4)     '-13
  Print Fix(-12.4)     '-12
  Print Fix("-12.1xx")'-12
  Print Int("-12.1xx")'-13
End Sub
```

Die Funktion CCur konvertiert einen numerischen Ausdruck zu einem Währungsobjekt. Visual Basic .NET hat die Unterstützung für die Funktion CCur und auch für den Datentyp Currency eingestellt. StarBasic hält noch an dem Datentyp Currency fest.

4.5. Konvertierungen von Zahl zu String

Funktionen zur Stringkonvertierung, s. Tabelle 31, wandeln Daten, die keine Strings sind, in Strings um. In OOo wird Text gemäß Unicode 2.0 gespeichert, wodurch eine Vielzahl von Sprachen unterstützt wird. Jede String-Variable kann bis zu 65.535 Zeichen enthalten.

Tabelle 31. *Funktionen zur String-Konvertierung.*

Funktion	Beschreibung
Str	Konvertiert eine Zahl zu einem String ohne Lokalisierung.
CStr	Konvertiert jeden Typ zu einem String. Zahlen und Datumswerte werden gemäß dem lokalen Gebietsschema formatiert.
Hex	Gibt eine Zahl in hexadezimaler Darstellung als String zurück.
Oct	Gibt eine Zahl in oktaler Darstellung als String zurück.

4.6. Einfache Formatierung

Die Funktion CStr konvertiert ganz allgemein jeden Typ zu einem String. Der Rückgabewert hängt vom Datentyp des Arguments ab. Boolesche Werte resultieren in „True“ oder „False“. Datumswerte werden in das kurze Datumsformat des Systems umgesetzt. Zahlen werden zu einer Zeichenkette konvertiert. Siehe Listing 90.

Listing 90. *Die Ausgabe von CStr ist abhängig vom Gebietsschema, hier Deutsch (Deutschland).*

```
Sub TestCStr
  Dim n As Long, d As Double, b As Boolean
  n = 999999999 : d = EXP(1.0) : b = False
  Print "X" & CStr(b)   'XFalse
  Print "X" & CStr(n)   'X999999999
  Print "X" & CStr(d)   'X2,71828182845904
  Print "X" & CStr(Now)'X26.07.2020 14:55:19 (fast 17 Jahre nach der 1. Auflage
End Sub
```

Die Funktion CStr führt eine einfache Zahlenformatierung nach Kenntnis des lokalen Gebietsschemas durch. Mit Str gibt es eine einfache Konvertierung einer Zahl zu einem String. Obwohl die Funktion Str eigentlich speziell für numerische Werte gedacht ist, ähnelt die Ausgabe sehr der von CStr. Wenn die Funktion Str eine Zahl zu einem String konvertiert, wird immer für das Vorzeichen der Zahl ein Leerzeichen vorangestellt. Eine negative Zahl enthält das Minuszeichen, das Leerzeichen entfällt. Eine nicht-negative Zahl hingegen enthält ein führendes Leerzeichen. Str gibt eine Zahl nicht-lokalisiert zurück, als Dezimaltrenner steht immer der Punkt. Die Ausgabe eines Datums wird jedoch gemäß dem Gebietsschema formatiert. Siehe Listing 91.

Listing 91. *Die Ausgabe von Str ist unabhängig vom Gebietsschema (Ausnahme: Datum).*

```
Sub TestStr
  Dim n As Long, d As Double, b As Boolean
  n = 999999999 : d = EXP(1.0) : b = False
  Print "X" & Str(b)   'XFalse
  Print "X" & Str(n)   'X 999999999
```

```
  Print "X" & Str(d)   'X 2.71828182845904
  Print "X" & Str(Now)'X26.07.2020 14:55:19 (fast 17 Jahre nach der 1. Auflage
End Sub
```

Die Ausgabe vom Code in Listing 90 und Listing 91 sind gleich bis auf die führenden Leerzeichen vor den nicht-negativen Zahlen und dem Dezimaltrenner. Wenn Sie den Code mit einem anderen Gebietsschema ausführen, wie zum Beispiel Englisch (USA), ändert sich die Ausgabe von Listing 90.

Tipp	Es gibt kaum einen Grund, Str anstatt CStr zu nutzen. Str mag wohl ein wenig schneller sein, aber CStr kennt Ihr aktuelles Gebietsschema.

Zur Demonstration, dass CStr abhängig vom Gebietsschema arbeitet, habe ich mein Gebietsschema auf Englisch (USA) umgestellt und den Code in Listing 90 noch einmal ausgeführt. Listing 92 zeigt, dass der Dezimaltrenner nun der Punkt ist und dass das Datum die Form DD/MM/YYYY hat.

Listing 92. *Die Ausgabe von CStr ist abhängig vom Gebietsschema; hier Englisch (USA).*

```
Dim n As Long, d As Double, b As Boolean
n = 999999999 : d = Exp(1.0) : b = False
Print "X" & CStr(b)   'XFalse
Print "X" & CStr(n)   'X999999999
Print "X" & CStr(d)   'X2.71828182845904
Print "X" & CStr(Now)'X06/29/2011 20:35:42
```

4.7. Zahlen auf anderer Basis, hexadezimal, oktal und binär

Basic stellt die Funktionen Hex und Oct bereit zur Konvertierung einer Zahl in ihre hexadezimale und oktale Form. Allerdings fehlt originäre Unterstützung zur Konvertierung zu und von Binärzahlen. Und man kann die Ausgabe von Hex und Oct nicht direkt zur Rückkonvertierung zu einer Zahl verwenden, weil dem String das einleitende „&H“ und „&O“ fehlt.

```
Print Hex(447)              '1BF
Print CInt("&H" & Hex(747)) '747
Print Oct(877)              '1555
Print CInt("&O" & Oct(213)) '213
```

Der Quellcode für Kapitel 2 enthält die Funktion IntToBinaryString, die eine Ganzzahl zu einer Binärzahl konvertiert (Modul LanguageConstructs). Die Funktion ist sehr flexibel, aber nicht besonders schnell. In Listing 93 finden Sie eine schnellere Routine, die sich der Funktion Hex bedient.

Listing 93. *Konvertierung einer Ganzzahl zu einer Binärzahl (String).*

```
Function IntToBinaryString(ByVal x As Long) As String
  Dim sHex As String
  Dim sBin As String
  Dim i As Integer
  sHex = Hex(x)
  For i = 1 To Len(sHex)
    Select Case Mid(sHex, i, 1)
      Case "0"
        sBin = sBin & "0000"
      Case "1"
        sBin = sBin & "0001"
      Case "2"
        sBin = sBin & "0010"
      Case "3"
        sBin = sBin & "0011"
      Case "4"
```

```
        sBin = sBin & "0100"
      Case "5"
        sBin = sBin & "0101"
      Case "6"
        sBin = sBin & "0110"
      Case "7"
        sBin = sBin & "0111"
      Case "8"
        sBin = sBin & "1000"
      Case "9"
        sBin = sBin & "1001"
      Case "A"
        sBin = sBin & "1010"
      Case "B"
        sBin = sBin & "1011"
      Case "C"
        sBin = sBin & "1100"
      Case "D"
        sBin = sBin & "1101"
      Case "E"
        sBin = sBin & "1110"
      Case "F"
        sBin = sBin & "1111"
    End Select
  Next
  IntToBinaryString = sBin
End Function
```

Der Code in Listing 93 mag lang sein, ist aber sehr einfach. Es gibt eine Wechselbeziehung zwischen Hexadezimalziffern und Binärziffern: jede Hexadezimalziffer entspricht vier Binärziffern. Diese Beziehung besteht nicht mit Zahlen auf der Basis 10. Die Zahl wird mit der Funktion Hex zu einer Hexadezimalzahl konvertiert. Dann wird jede Hexadezimalziffer zu den entsprechenden Binärziffern konvertiert. Um eine Binärzahl in Stringform zu einem Integer zurück zu wandeln, nehmen Sie den Code in Listing 94.

Listing 94. *Konvertierung einer Binärzahl (String) zu einem Long-Integer.*

```
Function BinaryStringToLong(s$) As Long
  Dim sHex As String
  Dim sBin As String
  Dim i As Integer
  Dim nLeftOver As Integer 'Für den Rest nach Ganzzahldivision
  Dim n As Integer

  n = Len(s$)
  nLeftOver = n Mod 4
  If nLeftOver > 0 Then
    sHex = SmallBinToHex(Left(s$, nLeftOver))
  End If
  For i = nLeftOver + 1 To n Step 4
    sHex = sHex & SmallBinToHex(Mid(s$, i, 4))
  Next
  BinaryStringToLong = CLng("&H" & sHex)
End Function

Function SmallBinToHex(s$) As String
  If Len(s$) < 4 Then s$ = String(4 - Len(s$), "0") & s$
  Select Case s$
    Case "0000"
```

```
      SmallBinToHex = "0"
    Case "0001"
      SmallBinToHex = "1"
    Case "0010"
      SmallBinToHex = "2"
    Case "0011"
      SmallBinToHex = "3"
    Case "0100"
      SmallBinToHex = "4"
    Case "0101"
      SmallBinToHex = "5"
    Case "0110"
      SmallBinToHex = "6"
    Case "0111"
      SmallBinToHex = "7"
    Case "1000"
      SmallBinToHex = "8"
    Case "1001"
      SmallBinToHex = "9"
    Case "1010"
      SmallBinToHex = "A"
    Case "1011"
      SmallBinToHex = "B"
    Case "1100"
      SmallBinToHex = "C"
    Case "1101"
      SmallBinToHex = "D"
    Case "1110"
      SmallBinToHex = "E"
    Case "1111"
      SmallBinToHex = "F"
  End Select
End Function
```

Zur Konvertierung eines Binärzahlstrings zu einem Integer wird die Zahl zuerst zu einer Hexadezimalzahl konvertiert. Ein Satz von vier Binärziffern entspricht einer einzelnen Hexadezimalziffer. Die Zahl wird links mit Nullen aufgefüllt, so dass der String in Blöcke von je vier Binärziffern aufgeteilt werden kann. Jeder dieser Blöcke von vier Binärziffern wird zu einer Hexadezimalziffer umgewandelt. Dann wird mit Hilfe der Funktion CLng die Hexadezimalzahl in die dezimale Form gebracht. Die Routine in Listing 95 zeigt den Gebrauch dieser Funktionen, s. auch Bild 44.

Listing 95. *Beispiel für Konvertierungen einer Ganzzahl.*

```
Sub ExampleWholeNumberConversions
  Dim s As String
  Dim n As Long
  Dim nAsHex$, nAsOct$, nAsBin$
  s = InputBox("Zu konvertierende Zahl:", "Long in anderer Form", "1389")
  If IsNull(s) Then Exit Sub
  If Len(Trim(s)) = 0 Then Exit Sub
  n = CLng(Trim(s))  'Trim entfernt führende und nachgestellte Leerzeichen
  nAsHex = Hex(n)
  nAsOct = Oct(n)
  nAsBin = IntToBinaryString(n)
  s = "Originalzahl = " & CStr(n) & Chr$(10) &_
      "Hex(" & CStr(n) & ") = " & nAsHex & Chr$(10) &_
      "Oct(" & CStr(n) & ") = " & nAsOct & Chr$(10) &_
```

```
        "Binary(" & CStr(n) & ") = " & nAsBin &_
        " ==> " & BinaryStringToLong(nAsBin)
  MsgBox(s, 0, "Konvertierung einer ganzen Zahl")
End Sub
```

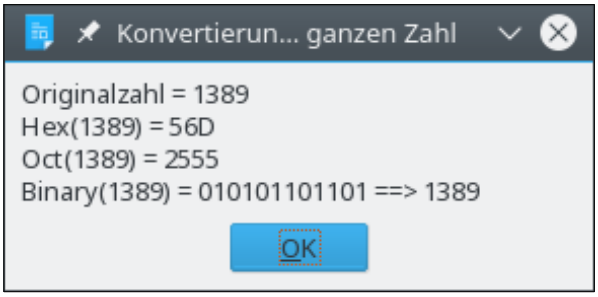


Bild 44. *Konvertierung einer ganzen Zahl in die hexadezimale, oktale und binäre Form.*

4.8. Zufallszahlen

Basic erzeugt Zufallszahlen als Fließkommazahlen im Bereich zwischen 0 und 1. Von Rechnern erzeugte Zufallszahlen sind im allgemeinen nicht zufällig. Diese „Zufallszahlen“ werden durch einen Algorithmus erzeugt, der auf einer vorherigen Zufallszahl aufsetzt. Die erste Zahl, auf der alle weiteren Zufallszahlen basieren, heißt „Seed“ (deutsch „Saat“). Mit der Funktion Randomize wird der Wert des Seed festgelegt. Wird kein Wert angegeben, verwendet die Funktion den aktuellen Wert des Systemzeitgebers. Durch die Angabe eines spezifischen Startwerts haben Sie die Möglichkeit, identische Zufallszahlensequenzen zu erzeugen, um Programme zu testen.

Die Funktion Rnd erzeugt eine Zufallszahl zwischen 0 (inklusive) und 1 (exklusive): 0 <= x < 1.

Die Basic-Hilfe führt für die Funktion Rnd ein optionales Argument auf, das allerdings ignoriert wird. Ich habe den Quellcode der Funktion Rnd untersucht, der sich übrigens von der Version 1.x bis 3.2.1 nicht verändert hat: Das Argument wird tatsächlich ignoriert.

```
Print Rnd()    'Eine Zahl zwischen 0 und 1
Print Rnd()    'Noch eine Zahl zwischen 0 und 1
```

Um einen anderen Zahlenbereich als 0 bis 1 zu erhalten, braucht man ein paar mathematische Rechenoperationen. Zum Beispiel erhält man durch die Multiplikation einer Zufallszahl mit 10 eine Fließkommazahl zwischen 0 und 10 (exklusive). Für einen Bereich, der nicht mit 0 beginnen soll, addieren Sie einen passenden Offset. Siehe Listing 96.

Listing 96. *Rückgabe einer Zufallszahl innerhalb eines bestimmten Bereichs.*

```
Function RndRange(lowerBound As Double, upperBound As Double) As Double
REM lowerBound = untere Grenze, upperBound = obere Grenze
  RndRange = lowerBound + Rnd() * (upperBound - lowerBound)
End Function
```

Mit einer entsprechenden Funktion, zum Beispiel CInt oder CLng, erhalten Sie eine ganze Zahl statt einer Fließkommazahl.

```
CLng(lowerBound + Rnd() * (upperBound - lowerBound))
```

Zur Ermittlung des GGT (des Größten Gemeinsamen Teilers) zweier Ganzzahlen hatte ich zwei Funktionen zur Auswahl. Ich wollte wissen, welche schneller ist. Ich erzeugte Zufallszahlen und rief jede Routine einige tausendmal auf. Bei Tests zur Laufzeitermittlung ist es wichtig, für jeden Durchlauf dieselben Daten zu benutzen. Ich konnte die Zufallszahlen aus dem Grund nehmen, weil die Anweisung Randomize mir die Möglichkeit gab, jedes Mal dieselben Zufallszahlen zu erzeugen.

```
Randomize(2)    'Der Zufallszahlengenerator wird auf einen definierten Stand gesetzt.
t1 = GetSystemTicks()
For i = 0 To 30000
  n1 = CLng(10000 * Rnd())
```

```
  n2 = CLng(10000 * Rnd())
  Call gcd1(n1, n2)
Next
total_time_1 = getSystemTicks() - t1

Randomize(2)    'Der Zufallszahlengenerator wird auf den definierten Stand
zurückgesetzt.
t1 = GetSystemTicks()
For i = 0 To 30000
  n1 = CLng(10000 * Rnd())
  n2 = CLng(10000 * Rnd())
  Call gcd2(n1, n2)
Next
total_time_2 = getSystemTicks() - t1
```

4.9. Fazit

Die mathematischen Standardfunktionen in OOo bieten wenig Überraschendes. Die Konvertierungsfunktionen arbeiten gut, trotz der einen oder anderen Eigenart bei der Konvertierung von Strings zu Zahlen. Achten Sie darauf, die Funktion zu wählen, die mit dem Format und dem verwendeten Wertebereich umgehen kann. Besondere Aufmerksamkeit muss auch auf das Runden gelegt werden. Obwohl das Rundungsverhalten dokumentiert und konsistent ist, bewirken verschiedene Funktionen und Operatoren doch unterschiedliche Rundungsergebnisse.

5. Array-Routinen

Dieses Kapitel beschreibt die Subroutinen und Funktionen, die Basic zur Handhabung von Arrays bereitstellt. Es sind Methoden, Arrays zu bearbeiten, Arrays mit oder ohne Daten zu erstellen und Array-Dimensionen zu ändern. Dieses Kapitel stellt auch Methoden vor, die Eigenschaften von Array-Variablen zu ermitteln.

Ein Array ist eine Datenstruktur, in der gleichartige Datenelemente in indexierter Form abgelegt sind – zum Beispiel eine Spalte mit Namen oder eine Zahlentabelle. Basic hat Subroutinen und Funktionen, die Array-Dimensionen ändern, bestehende Arrays überprüfen und zwischen Arrays und skalaren Datentypen (das sind solche, die keine Arrays sind) konvertieren.

Die Mehrzahl der in Tabelle 32 aufgeführten Routinen erwartet als erstes Argument eine Array-Variable. Als Argumente zu einer Routine können Array-Variablen mit angehängten runden Klammern geschrieben werden, die jetzt optional sind, aber früher erforderlich waren (s. Listing 97).

Tipp	Es gibt beim Lesen eines Codes keinen Weg herauszufinden, ob sich a() auf ein Array oder eine Funktion bezieht. Sie müssen die Stelle finden, wo das gute Stück deklariert wurde.

Listing 97. *Runde Klammern sind nicht immer erforderlich, aber immer erlaubt.*

```
Sub AreArrayParensRequired
  Dim a(1 To 2)       'a() wird mit spezifischen Dimensionen deklariert.
  Dim b()             'b() wird als Array ohne spezifische Dimensionierung deklariert.
  Dim c               'c ist ein Variant und kann auch ein Array referenzieren.
  c = Array(1, 2)     'c referenziert ein Variant-Array.
  Print IsArray(a()) 'True
  Print IsArray(b()) 'True
  Print IsArray(c()) 'True
  Print IsArray(a)   'True
  Print IsArray(b)   'True
  Print IsArray(c)   'True
End Sub
```

Tabelle 32. *Liste der Subroutinen und Funktionen im Zusammenhang mit Arrays.*

Funktion	Beschreibung
Array(Args)	Gibt ein Variant-Array zurück, mit den Argumenten als Elemente.
DimArray(Args)	Gibt ein leeres Variant-Array zurück, dimensioniert durch die Argumente.
IsArray(Variant)	Gibt True zurück, wenn die Variable ein Array ist, ansonsten False.
Join(Array) Join(Array, Trenner)	Gibt einen String zurück, der die einzelnen Array-Elemente in Folge enthält, jeweils getrennt durch den optionalen Trennstring. Standardtrenner ist das Leerzeichen.
LBound(Array) LBound(Array, Dimension)	Gibt die untere Bereichsgrenze des Arrays zurück. Die optionale Dimensionsangabe bestimmt die zu berücksichtigende Dimension. Die erste Dimension ist 1.
ReDim [Preserve] Variable(Args) [As Type]	Ändert die Array-Dimensionen mit derselben Syntax wie Dim. Mit dem Schlüsselwort Preserve bleiben die bestehenden Werte erhalten. „As Type“ ist optional.
Split(String) Split(String, Trenner) Split(String, Trenner, n)	Splittet den String in ein Array von Strings. Der Standardtrenner ist ein Leerzeichen. Das optionale Argument „n“ begrenzt die Anzahl der zurückzugebenden Stringelemente.
UBound(Array) UBound(Array, Dimension)	Gibt die obere Bereichsgrenze des Arrays zurück. Die optionale Dimensionsangabe bestimmt die zu berücksichtigende Dimension. Die erste Dimension ist 1.

Der Gebrauch des Wortes „Dimension“ bezogen auf Arrays ist vergleichbar mit dem Bezug auf räumliche Dimensionen. Ein Array mit nur einer Dimension ist zum Beispiel wie eine Linie, man

kann entlang der Linie Kästchen aufstellen, die für die Datenelemente stehen. Ein Array mit zwei Dimensionen ist wie ein Gitter mit den Datenelementen in Zeilen und Spalten.

```
Dim a(3) As Integer          'Eindimensionales Array
Dim b(3 To 5) As String      'Eindimensionales Array
Dim c(5, 4) As Integer       'Zweidimensionales Array
Dim d(1 To 5, 4) As Integer  'Zweidimensionales Array
```

Die maximale Größe eines Arrays hängt vom verfügbaren Systemspeicher ab und somit auch vom Betriebssystem. Gehen Sie mal von etwa 1 Milliarde Einträge aus.

Für Softwareentwickler: intern ist ein Array als C++std::vector implementiert. Sehen Sie mal nach.

5.1. Array() erstellt schnell ein eindimensionales Array mit Daten

Mit der Funktion Array erstellen Sie auf schnelle Art und Weise ein Variant-Array mit Datenelementen. Die Funktion gibt ein Array vom Typ Variant zurück, das die übergebenen Argumente als Daten enthält, s. Listing 98. Diese Methode ist sehr effizient, wenn es darum geht, mit vorher feststehenden Werten ein Variant-Array einzurichten. Für jedes Argument gibt es einen Array-Eintrag.

Im folgenden Code wird erst ein Array mit fünf Elementen definiert, von 0 bis 4, und dann wird umständlich jedes Element initialisiert.

```
Dim v(4)
v(0) = 2 : v(1) = "Hilfe": v(2) = Now : v(3) = True : v(4) = 3.5
```

Das kann man mit der Funktion Array() einfacher haben. Sie können Konstanten als Argumente nehmen. Sie sind aber nicht dazu gezwungen. Es können auch Variablen oder Funktionen sein.

```
Sub JoinHelpMe
  Dim v()
  Dim FirstName$ : FirstName = "Bob"
  v() = Array(0, "Hilfe", Now, True, 3.5)
  Print Join(Array("Hilfe", 3, "Joe", Firstname))
End Sub
```

Die Argumentenliste ist eine durch Komma getrennte Liste von Ausdrücken, deren Typ beliebig ist, weil jedes Element des zurückgegebenen Arrays vom Typ Variant ist.

Listing 98. *Die Funktion Array gibt ein Variant-Array zurück.*

```
Sub PrintEighteen
  Dim vFirstNames                'Eine Variant-Variable kann ein Array referenzieren
  Dim vAges()                    'Eine Variant-Variable kann ein Array referenzieren
  vFirstNames = Array("Tom", "Rob")  'Array mit Strings
  vAges = Array(18, "Ten")           'Array mit einer Zahl und einem String
  Print vAges(0)                     'Das erste Element hat den Wert 18
End Sub
```

Variant-Variablen können Daten jedes Typs enthalten, einschließlich Arrays. Ich verwende häufig Variant-Variablen zur Aufnahme von Werten, die von Methoden zurückgegeben werden und deren Typ ich nicht mit Sicherheit kenne. Ich prüfe dann den Typ und gehe mit dem Wert entsprechend um. Das ist eine praktische Anwendung des Variablentyps Variant. Weil also eine Variant-Variable jeden Typ enthalten kann – inklusive ein Array –, kann auch jedes Element in einem Variant-Array ein Array enthalten. Im Code in der Tabelle 33 wird gezeigt, wie ein Array in ein anderes Array eingefügt wird. Der Code beider Spalten ist praktisch gleichwertig.

Tipp	Obwohl benutzerdefinierte Strukturen keine Arrays enthalten können, so können sie doch Variant-Variablen haben, die Arrays aufnehmen können.

Variant-Arrays haben den Vorteil, dass man leicht Kollektionen mit Daten verschiedener Typen aufbauen kann. Zum Beispiel können Postenbeschreibung (String), Auftragsnummer (Integer) und Rechnungsbetrag (Double oder Currency) ohne weiteres als Reihen in einem Array angelegt werden, und für jeden Kunden eine Reihe mit jedem dieser Datentypen. Solche Array-Reihen verhalten sich wie Datenbankreihen. In älteren Programmiersprachen musste man für jeden Datentyp eigene Arrays deklarieren, mit der Folge von erweitertem Programmieraufwand, um den Einsatz multipler relationaler Datenarrays zu bewältigen.

Tabelle 33. *Variant kann ein Array enthalten. Zwei wirkungsgleiche Methoden.*

`Dim v(1)`
`v(0) = Array(1, 2, 3)`
`v(1) = Array("eins", "zwei", "drei")`
	`Dim v()`
	`v = Array(Array(1, 2, 3),_`
	`  Array("eins", "zwei", "drei"))`

Wenn Sie in OOo 1.x ein Array innerhalb eines Arrays adressieren wollten, mussten Sie erst das enthaltene Array extrahieren und dieses dann indexieren. Viele meiner alten und noch existenten Code-Beispiele basieren darauf.

Listing 99. *Umständliche Methode, ein Array in einem Array anzusprechen.*

```
v = Array(Array(1, 2, 3), Array("eins", "zwei", "drei"))
x = v(0)          'Dies ist sehr umständlich.
Print x(1)        'Gibt 2 aus.
```

Irgendwann zwischen den Versionen 2.x und 3.x wurde die naheliegende Lösung präsentiert: man kann nun das enthaltene Array direkt ansprechen. Das ist vor allem dann nützlich, wenn Sie Datenarrays mit Zellinhalten aus Calc verwenden.

Listing 100. *Ab OOo 3.x müssen Sie nicht das enthaltene Array vor der Nutzung extrahieren.*

```
Sub ArrayInArray
  Dim v() : v = Array(Array(1, 2, 3), Array("eins", "zwei", "drei"))
  Print v(0)(1)
End Sub
```

Obwohl es so einfach ist, ein Array innerhalb eines Arrays zu erstellen, ist es aber normalerweise einfacher, ein Array mit mehreren Dimensionen anzulegen, wie in Listing 101 gezeigt wird. Die Konstruktion „Array in Array“ ist manchmal nützlich, wenn es in offensichtlicher Beziehung zu der realen Datenorganisation steht. Im allgemeinen ist es am besten, die Daten auf eine Weise zu organisieren, dass ein möglichst direkter, natürlicher und einprägsamer Bezug dazu besteht, wie sie erstellt, verwendet und verarbeitet werden.

Listing 101. *Es ist leichter, mehrdimensionale Arrays zu verwenden als Arrays in Arrays.*

```
Dim v(0 To 1, 0 To 2)
v(0, 0) = 1       : v(0, 1) = 2       : v(0, 2) = 3
v(1, 0) = "eins" : v(1, 1) = "zwei" : v(1, 2) = "drei"
Print v(0, 1)    'Gibt 2 aus
```

Es ist möglich, ein Variant-Array jedem anderen Array zuzuweisen, ohne Rücksicht auf dessen deklarierten Typ. Die Zuweisung eines Arrays zu einem anderen bewirkt, dass das eine Array das andere referenziert: sie werden zu ein und demselben Array. Wie schon früher erwähnt, ist das keine gute Idee. Es wird als Bug verstanden und wohl in einer späteren Basic-Version nicht mehr erlaubt sein. Nutzen Sie die Funktion Array und erfreuen Sie sich an Ihrer Flexibilität, aber weisen Sie das zurückgegebene Array nur einer Variant-Variablen oder einem Variant-Array zu.

Listing 102. *Zuweisung eines String-Arrays zu einem Integer-Array.*

```
Sub BadArrayTypes
  Dim a(0 To 1) As Integer
  Dim b(0 To 1) As String
  b(0) = "null" : b(1) = "eins"
  a() = b()
```

```
  Print a(0)
End Sub
```

Tipp Es ist keine gute Idee, ein Variant-Array einer Variablen zuzuweisen, die mit einem Typ anders als Variant deklariert wurde. Nachdem zum Beispiel einem Integer-Array die Referenz auf ein Variant-Array zugewiesen wurde, ist es möglich, den Elementen im Array Werte zuzuweisen, die nicht vom Typ Integer sind. Zugriffe auf Array-Elemente könnten wegen der Unverträglichkeit der Typen Integer und Variant unerwartete Werte zurückgeben.

```
Dim a(0 To 4) As Integer  ' Integer-Array.
a(2) = "Tom"              ' Zuweisung eines Strings zu einem Integer.
Print a(2)                ' 0, weil der String zu 0 konvertiert wird.
a() = Array(4, "Bob", 7)  ' Array gibt immer ein Variant-Array zurück.
a(2) = "Tom"              ' a() ist nun ein Variant.
Print a(2)                ' Tom
```

5.2. DimArray erstellt leere mehrdimensionale Arrays

Die Funktion DimArray legt ein leeres, dimensioniertes Variant-Array zur Laufzeit an und gibt es zurück. Die Argumente spezifizieren die Array-Dimensionen; je Argument eine Dimension. Ohne die Angabe eines Arguments wird ein leeres Array ohne Dimensionen angelegt.

Wenn Sie die benötigte Größe des Arrays schon vorher kennen, können Sie die Dimensionen bei der Deklarierung der Variablen definieren. Wenn Sie die Größe aber nicht kennen und ein Variant-Array akzeptabel ist, dann können Sie es als leeres, dimensioniertes Array zur Laufzeit erstellen.

Listing 103. *DimArray gibt ein dimensioniertes Variant-Array zurück, das keine Daten enthält.*

```
i% = 7
v = DimArray(3 * i%)    'Dasselbe wie Dim v(0 To 21)
v = DimArray(i%, 4)     'Dasselbe wie Dim v(0 To 7, 0 To 4)
```

Der Code in Listing 103 zeigt nicht, wie die Variable v deklariert wurde. Es funktioniert gleich gut, wenn v als Variant oder als Variant-Array deklariert wird. Die Argumentenliste besteht aus einer durch Kommas getrennten Liste von Ausdrücken. Jeder Ausdruck wird zu einem ganzzahligen Wert gerundet und als Bereich einer Dimension des zurückgegebenen Arrays gesetzt.

```
Sub DimArrayTests
  Dim a As Variant
  Dim v()
  Dim i As Integer
  i = 2
  a = DimArray(3)                          'Dasselbe wie Dim a(0 To 3)
  a = DimArray(1 + i, 2 * i)               'Dasselbe wie Dim a(0 To 3, 0 To 4)
  v()  = DimArray(1)                       'Dasselbe wie Dim v(0 To 1)
  v(0) = Array(1, 2, 3)                    'O nein, nicht das schon wieder!
  v(1) = Array("eins", "zwei", "drei")     'Sie können das tun, aber igitt!
  v = DimArray(1, 2)                       'Das ist nun sinnvoller!
  v(0, 0) = 1      : v(0, 1) = 2      : v(0, 2) = 3
  v(1, 0) = "eins" : v(1, 1) = "zwei" : v(1, 2) = "drei"
  Print v(0, 1)    'Gibt 2 aus
End Sub
```

Tipp Option Base 1 wirkt sich nicht auf die Array-Dimensionen aus, wie sie von der Funktion DimArray zurückgegeben werden. Für jede Dimension liegt die untere Bereichsgrenze bei null und die obere Bereichsgrenze bei dem gerundeten ganzzahligen Wert des betreffenden Ausdrucks.

5.3. Änderung der Array-Dimensionen

Mit ReDim werden die Dimensionen eines existierenden Arrays geändert. Die Syntax ist dieselbe wie bei der Anweisung Dim. Mit der Angabe des Schlüsselworts Preserve bleiben alle Daten erhalten, falls die Dimensionen erweitert werden. Falls sie aber eingeschränkt werden, gehen Daten durch Abtrennung verloren. Im Gegensatz zu anderen BASIC-Varianten erlaubt es StarBasic, dass alle Dimensionen eines Arrays unter Beibehaltung bestehender Daten geändert werden.

Der Hauptzweck der Anweisung ReDim ist die Änderung der Dimensionen eines bestehenden Arrays. Wenn Sie die benötigte Größe des Arrays schon vorher kennen, können Sie die Dimensionen bei der Deklarierung der Variablen definieren. Wenn Sie die Größe zu Beginn aber nicht kennen, dann können Sie das Array mit beliebiger Größe deklarieren, einschließlich als leeres Array, und die Dimensionen dann ändern, wenn Sie sie kennen.

```
Dim v() As Integer
Dim x(4) As Integer
i% = 7
ReDim v(3 * i%) As Integer      'Dasselbe wie Dim v(0 To 21) As Integer.
ReDim x(i%, 1 To 4) As Integer 'Dasselbe wie Dim x(0 To 7, 1 To 4).
```

Die Anweisung ReDim ändert die Dimension eines vorhandenen Arrays, auch eines leeren. ReDim definiert sowohl die Dimensionen als auch den Typ. Die Typangabe bei der Anweisung ReDim muss dem Typ entsprechen, mit dem die Variable deklariert wurde. Wenn die Typen voneinander abweichen, erhalten Sie einen Fehler zur Kompilierungszeit: „Variable bereits definiert“.

```
Dim a() As Integer             'Leeres Integer-Array.
Dim v(8)                       'Variant-Array mit neun Elementen.
ReDim v()                      'v() ist ein gültiges leeres Array.
ReDim a(2 To 4, 5) As Integer 'a() ist ein zweidimensionales Array.
```

Die Funktion DimArray legt ein dimensioniertes Variant-Array ohne enthaltene Daten an und gibt es zurück. Das können Sie jedoch nicht gebrauchen, wenn Sie ein Array eines bestimmten Typs benötigen oder wenn Sie einfach die Dimensionen eines vorhandenen Arrays mit Datenerhalt ändern wollen. Die Anweisung ReDim ändert die Dimensionen eines existierenden Arrays mit der Option, die bestehenden Daten zu erhalten. Sie können mit der Anweisung ReDim ein dimensioniertes Array leeren.

Die Subroutine in Listing 104 enthält viele Beispiele der Anweisung ReDim zusammen mit dem Schlüsselwort Preserve. Bild 45 zeigt die Ergebnisse dieser Befehle.

Listing 104. *Mit ReDim Preserve ändern Sie die Dimensionen und bewahren die Daten.*

```
Sub ExampleReDimPreserve
  Dim a(5) As Integer      'Ein Array im Bereich 0 bis 5
  Dim b()                  'Ein leeres Array vom Typ Variant
  Dim c() As Integer       'Ein leeres Array vom Typ Integer
  Dim s$                   'Der String zur Kumulierung des Ausgabetextes

  REM a ist dimensioniert von 0 bis 5 und erhält die Werte a(i) = i
  a(0) = 0 : a(1) = 1 : a(2) = 2 : a(3) = 3 : a(4) = 4 : a(5) = 5
  s$ = "a() zu Beginn = " & Join(a()) & Chr$(10)

  REM a wird dimensioniert von 1 bis 3 mit den Werten a(i) = i
  ReDim Preserve a(1 To 3) As Integer
  s$ = s$ & "ReDim Preserve a(1 To 3) = " &  Join(a()) & Chr$(10)

  ReDim a() As Integer
  s$ = s$ & "ReDim a() hat LBound = " & LBound(a()) _
          & " UBound = " & UBound(a()) & Chr$(10)

  REM Array() gibt Variant zurück
```

```
  REM b wird dimensioniert von 0 bis 9 mit den Werten b(i) = i + 1
  b = Array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
  s$ = s & Chr$(10) & "b() zu Beginn = " & Join(b()) & Chr$(10)

  REM b wird dimensioniert von 1 bis 3 mit den Werten b(i) = i+1
  Dim il%, iu%
  il = 1 : iu = 3
  ReDim Preserve b(il To iu)
  s$ = s$ & "ReDim Preserve b(1 To 3) = " &  Join(b()) & Chr$(10)

  ReDim b(-5 To 5)
  s$ = s$ & "ReDim b(-5 To 5) = " &  Join(b()) & Chr$(10)
  s$ = s$ & "ReDim b(-5 To 5) hat LBound = " & LBound(b()) _
          & " UBound = " & UBound(b()) & Chr$(10) & Chr$(10)

  ReDim b(-5 To 5, 2 To 4)
  s$ = s$ & "ReDim b(-5 To 5, 2 To 4) hat Dimension 1 LBound = " &_
       LBound(b()) & " UBound = " & UBound(b()) & Chr$(10)
  s$ = s$ & "ReDim b(-5 To 5, 2 To 4) hat Dimension 2 LBound = " &_
       LBound(b(), 2) & " UBound = " & UBound(b(), 2) & Chr$(10)

  MsgBox s$, 0, "ReDim-Beispiele"
End Sub
```

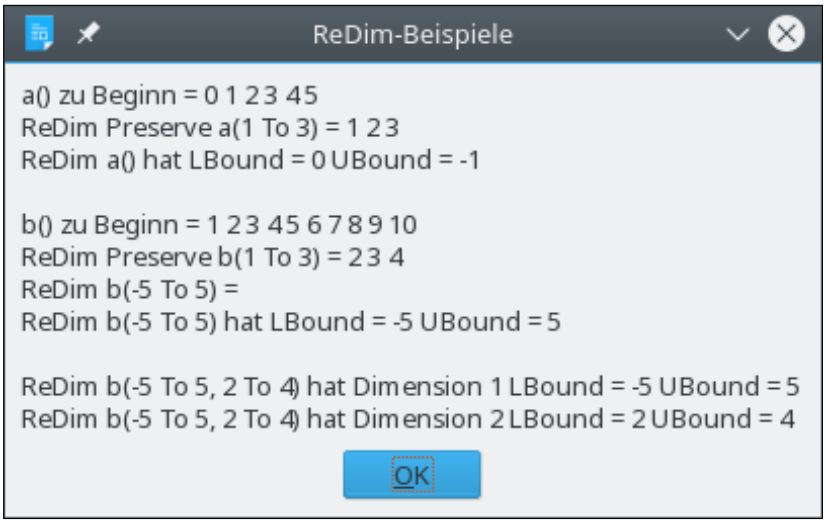


Bild 45. *Mit ReDim ändert man die Dimensionen eines Arrays.*

5.4. Array zu String und wieder zurück

Genauso wie es für die Bildschirm- oder Druckausgabe üblich ist, ein Array von Werten in einen einzigen String zu konvertieren, ist es auch üblich, einen String in Einzelteile zu zerlegen. Basic stellt für diese Aufgaben die Funktionen Join und Split zur Verfügung.

Das erste Argument für die Funktion Join ist ein eindimensionales Array. Bei jeder anderen Dimension gibt es einen Laufzeitfehler. Die Elemente des Arrays werden mit einem optionalen Trennstring zwischen den Elementen verkettet. Als Standardtrenner dient ein einzelnes Leerzeichen.

```
Join(Array(1, 2, 3))        '1 2 3  verwendet wird der Standardtrenner
Join(Array(1, 2, 3), "X")   '1X2X3  mit einem spezifischen Trenner
Join(Array(1, 2, 3), "")    '123    mit einem leeren Trenner
```

Die Funktion Split gibt ein Variant-Array von Strings zurück. Dazu wird ein String an spezifizierten Trennpositionen in Einzelstrings zerlegt. Mit anderen Worten, ein String wird mit einem einzigen Befehl in Einzelteile zergliedert. Der Trennstring markiert die Teilgrenzen. Zum Beispiel zerlegt der Trenner „XY“ den String „12XY11XY22“ in die Einzelstrings („12“, „11“, „22“). Der Standardtrenner ist ein Leerzeichen, es kann aber ein beliebiger Stringausdruck sein, der länger ist als null.

```
Split("1 2 3")              'gibt Array("1", "2", "3") zurück, Trenner: " "
Split("1, 2, 3", ", ")      'gibt Array("1", "2", "3") zurück, Trenner: ", "
```

Das optionale dritte Argument ist dazu da, die Größe des zurückgegebenen Arrays zu limitieren. Es dient einzig der Größenbegrenzung und spielt keine Rolle, wenn die zurückgegebene Größe kleiner als das Limit ist. Zum Beispiel hat die 4 in Split("1X2X3", "X", 4) keine Wirkung, denn das zurückgegebene Array hat nur drei Elemente. Wenn das Limit jedoch erreicht ist, enthält das letzte Element des Arrays den Rest des unzerteilten Strings.

```
Split("1, 2, 3", ", ", 2)  'gibt das Array ("1", "2, 3") zurück, Trenner: ", "
```

Tipp

Das zweite Argument der Funktion Split ist ein String, also wird es von Basic automatisch zu einem String konvertiert. Die Anweisung Split("0 1 2 3", 2) konvertiert die 2 zu einem String und verwendet ihn als Trenner. Das zurückgegebene Array enthält zwei Elemente, „0 1“ und „ 3“. Sie müssen den Trenner explizit spezifizieren, wenn Sie die Anzahl an zurückgegebenen Strings limitieren wollen. Die korrekte Form ist Split("0 1 2 3", " ", 2).

Die Funktion Split setzt vor und nach dem Trenner je einen String voraus, sogar wenn der String die Länge null hat.

```
Split("X1XX2X", "X") ergibt ("", "1", "", "2", "")
```

Der erste zurückgegebene String ist leer, weil das erste Argument mit einem Trenner beginnt. Zwei aufeinander folgende Trenner produzieren einen leeren String zwischen „1“ und „2“. Und schließlich ist der letzte String leer, weil ein Trenner am Ende steht.

Die Funktion Split ist beinahe die Umkehrung der Funktion Join. Die Funktion Join kann einen String mit der Länge null als Trenner verwenden, die Funktion Split aber nicht. Wenn der mit Join erzeugte String den Trenner enthält, wird durch ein folgendes Split ein anderer Satz Strings erzeugt. Zum Beispiel wird "a b c" erzeugt, wenn man "a b" und "c" mit einem Leerzeichen aneinanderfügt. Zerteilt man den String dann mit einem Leerzeichen, erhält man ("a", "b", "c"), was nicht dem ursprünglichen Array entspricht.

Ich habe viel Zeit damit verbracht, ein Makro zu schreiben und auszutesten, das in einem String den Text „Sbx“ an jeder Stelle entfernt. Mit Split und Join ist das erheblich kürzer und schneller:

```
Join(Split(s, "Sbx"), "")
```

5.5. Funktionen für Informationen über Arrays

Die fundamentalste Frage an ein Array ist, ob es wirklich ein Array ist oder nicht. Die Funktion IsArray gibt True zurück, wenn das Argument ein Array ist, ansonsten False. Mit den Funktionen LBound und UBound ermitteln Sie die untere und obere Dimensionsgrenze eines Arrays. Ein Array ist leer, wenn die Obergrenze kleiner als die Untergrenze ist.

Das erste Argument für LBound und UBound ist das betreffende Array. Das zweite, optionale, Argument ist ein Integer-Ausdruck zur Auswahl der zurückzugebenden Dimension. Der Standardwert ist 1, der die Unter- bzw. Obergrenze der ersten Dimension zurückgibt.

```
Dim a()
Dim b(2 To 3, -5 To 5)
Print LBound(a())     ' 0
Print UBound(a())     '-1, weil das Array leer ist
Print LBound(b())     ' 2, kein optionales zweites Argument, daher Standardwert 1
Print LBound(b(), 1) ' 2, das optionale zweite Argument zeigt auf die erste Dimension
Print UBound(b(), 2) ' 5
```

Wenn der Wert des zweiten Arguments unzulässig ist oder wenn er größer ist als die Anzahl an Dimensionen oder wenn er kleiner ist als 1, dann wird ein Laufzeitfehler ausgelöst.

Listing 105. *SafeUBound wird keinen Fehler auslösen.*

```
Function SafeUBound(v, Optional n) As Integer
  SafeUBound = -1               'Im Fall eines Fehlers ist die Rückgabe geregelt.
  On Error GoTo BadArrayFound   'Springe ans Ende im Falle eines Fehlers
  If IsMissing(n) Then          'Wurde das optionale Argument verwendet?
    SafeUBound = UBound(v)
  Else
    SafeUBound = UBound(v, n)   'Es gibt ein optionales Argument
  End If
BadArrayFound:                  'Sprungmarke im Falle eines Fehlers
  On Error GoTo 0               'Abschalten dieses Error-Handlers
End Function
```

Das Makro in Listing 105 gibt brav -1 zurück, wenn ein Fehler auftritt. Es ist der korrekte Wert für ungültige leere Arrays, aber -1 wird auch geliefert, wenn das erste Argument kein Array ist oder wenn das zweite Argument einfach zu groß ist. Die Funktion ArrayInfo in Listing 106 verwendet eine ähnliche Technik, um Array-Informationen über eine Variable zu ermitteln. Siehe auch Bild 46.

Listing 106. *Gibt Informationen über ein Array aus.*

```
REM Wenn das erste Argument ein Array ist, werden die Dimensionen bestimmt.
REM Besondere Beachtung gilt einem mit DimArray oder Array erstellten leeren Array.
REM a     : zu überprüfende Variable
REM sName : Name der Variablen für einen besser lesbaren Ausgabetext
Function ArrayInfo(a, sName$) As String
  REM Zu Anfang die Überprüfung, ob:
  REM die Variable nicht NULL ist - ein leeres Objekt
  REM die Variable nicht EMPTY ist - ein nicht initialisiertes Variant-Objekt
  REM die Variable ein Array ist.
  If IsNull(a) Then
    ArrayInfo = "Variable " & sName & " ist null"
    Exit Function
  End If
  If IsEmpty(a) Then
    ArrayInfo = "Variable " & sName & " ist Empty"
    Exit Function
  End If
  If Not IsArray(a) Then
    ArrayInfo = "Variable " & sName & " ist kein Array"
    Exit Function
  End If

  REM Die Variable ist ein Array, also ran an die Arbeit
  Dim s As String                 'Der Rückgabestring in s
  Dim iCurDim As Integer          'Aktuelle Dimension
  Dim i%, j%                      'Für die LBound- und UBound-Werte
  On Error GoTo BadDimension      'Der Error-Handler
  iCurDim = 1                     'Bereit zur Überprüfung der ersten Dimension

  REM Der Beginn des Rückgabestrings
  s = "Array dimensioniert als " & sName$ & "("

  Do While True                     'Endlosschleife
    i = LBound(a(), iCurDim)        'Fehler, wenn die Dimensionsangabe zu groß ist
```

```
    j = UBound(a(), iCurDim)          'oder wenn es ein ungültiges leeres Array ist

    If i > j Then Exit Do             'Ende der Schleife, wenn das Array leer ist

    If iCurDim > 1 Then s = s & ", " 'Trennt die Dimensionen mit einem Komma
    s = s & i & " To " & j            'Fügt die aktuellen Dimensionsgrenzen ein
    iCurDim = iCurDim + 1             'Weiter zur nächsten Dimension
  Loop

  REM Das Makro erreicht diese Stelle nur, wenn das Array gültig und leer ist.
  REM Ansonsten wird ein Fehler ausgelöst, wenn die Dimensionsangabe zu groß wird
  REM und ein Sprung zum Error-Handler erfolgt.
  REM Fügt den Typ ein, wie er von der Funktion TypeName zurückgegeben wird.
  REM Dem Typnamen ist "()" angehängt, das entfernt werden muss
  s = s & ") As " & Left(TypeName(a), Len(TypeName(a)) - 2)
  ArrayInfo = s
  Exit Function

BadDimension:
  REM Schaltet den Error-Handler aus
  On Error GoTo 0

  REM Fügt den Typ ein, wie er von der Funktion TypeName zurückgegeben wird.
  REM Dem Typnamen ist "()" angehängt, das entfernt werden muss
  s = s & ") As " & Left(TypeName(a), Len(TypeName(a)) - 2)

  REM Wenn der Fehler bei der ersten Dimension auftrat, dann muss
  REM das Array ungültig und leer sein.
  REM Dieser Fehler tritt spätestens ab OOo 3.2.1 nicht mehr auf.
  If iCurDim = 1 Then s = s & " *** UNGÜLTIGES leeres Array"
  ArrayInfo = s
End Function

Sub UseArrayInfo
  Dim i As Integer, v
  Dim ia(1 To 3) As Integer
  Dim sa() As Single
  Dim m(3, 4, -4 To -1)

  Dim s As String
  s = s & ArrayInfo(i, "i") & Chr$(10)           'Kein Array
  s = s & ArrayInfo(v, "v") & Chr$(10)           'Leere Variant-Variable
  s = s & ArrayInfo(sa(), "sa") & Chr$(10)       'Leeres Array
  s = s & ArrayInfo(Array(), "Array") & Chr$(10) 'Leeres Variant-Array, BÖSE bis
                                                 'irgendwann vor OOo 3.2.1
  s = s & ArrayInfo(ia(), "ia") & Chr$(10)
  s = s & arrayInfo(m(), "m") & Chr$(10)
  MsgBox s, 0, "Array-Info"
End Sub
```

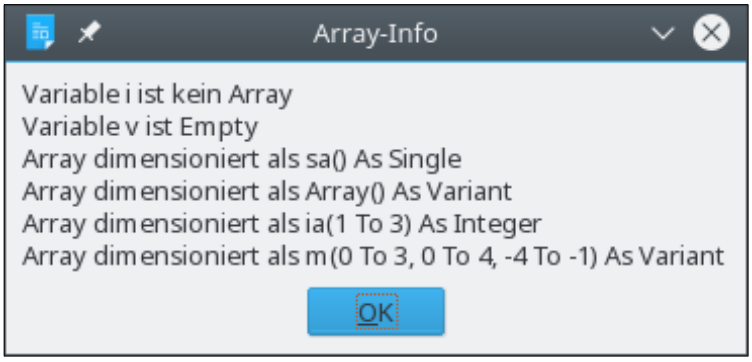


Bild 46. *Ermittlung der Dimensionen eines Arrays mit Hilfe eines passenden Error-Handlers.*

In einem Array mit nur einer Dimension kann die obere Bereichsgrenze niedriger sein als die untere. Das ist ein Hinweis darauf, dass für das Array noch keine Speicherplätze für Datenelemente bereitgehalten sind, im Unterschied zu einem Array, für das zwar Speicher bereitgehalten ist, in dem aber noch keine Daten gespeichert sind. Für die meisten Datentypen, wie zum Beispiel Integer, ist im bereitgehaltenen Speicher auch ein Wert gespeichert.

```
Dim a(3) As Integer 'Dieses Array hat vier Integer-Werte, alle sind null
Dim b(3)            'Dieses Array hat vier Variants, alle sind leer
Dim c()             'Dieses Array hat eine Dimension und keinen Speicherplatz:
                    'Ubound < Lbound
v = Array()         'Dieses Array hat keine Dimensionen.
```

5.6. Fazit

Die Behandlung von Arrays in Basic ist sehr flexibel. Sie können die Array-Eigenschaften abfragen und die Dimensionen ändern. Der Typ Variant bietet in Basic große Flexibilität zur Einrichtung von Kollektionen zusammengehöriger Daten unterschiedlichen Typs. Strings und Arrays sind miteinander verwandt. Die Verwendung der Funktionen Join und Split mit String-Arrays ermöglicht einen sehr kompakten und mächtigen Code zur String-Bearbeitung.

6. Datums- und Uhrzeit-Routinen

Gegenstand dieses Kapitels sind die Subroutinen und Funktionen, die Basic zu Datumsangaben bereitstellt – Funktionen, die das aktuelle Datum und die aktuelle Uhrzeit abrufen, Veränderungen an Datums- und Uhrzeitangaben durchführen und Zeitmessungen vornehmen. Behandelt wird auch ein möglicherweise unerwartetes Verhalten im Zusammenhang mit dem 4. Oktober 1582 und dem 30. Dezember 1899.

Variablen vom Typ Date enthalten sowohl ein Datum als auch eine Uhrzeit. Basic behandelt Dates intern als Fließkommazahlen vom Typ Double. Der Teil der Zahl links vom Dezimaltrenner enthält das Datum als Anzahl der Tage seit dem 30. Dezember 1899, der Bruchanteil rechts vom Dezimaltrenner enthält die Uhrzeit. Die Addition von 1 zu einem Datum erhöht zum Beispiel das Datum um einen Tag. Die Addition von 1/24 zu einem Datumswert erhöht das Datum um eine Stunde, schließlich hat ein Tag ja 24 Stunden. In der Tabelle 34 sind die Funktionen zu Datum und Uhrzeit in OOo zusammengefasst.

Tabelle 34. *Funktionen und Subroutinen im Zusammenhang mit Datum und Uhrzeit.*

Funktion	Typ	Beschreibung
CDate(Ausdruck)	Date	Konvertiert eine Zahl oder einen String zu einem Date-Wert.
CDateFromIso(String)	Date	Konvertiert einen ISO-8601-Datumsstring zu einem Date-Wert.
CDateToIso(Date)	String	Konvertiert einen Datumswert zu einem ISO-8601-Datumsstring.
Date()	String	Gibt das aktuelle Datum als String zurück.
DateSerial(Jahr, Monat, Tag)	Date	Erzeugt einen Date-Wert aus den Integer-Argumenten Jahr, Monat, Tag.
DateValue(Date)	Date	Extrahiert das Datum aus einem Datums-/Uhrzeitwert durch Abschneiden des Dezimalteils.
Day(Date)	Integer	Gibt den Tag des Monats eines Date-Werts als Integer (1..31) zurück..
GetSystemTicks()	Long	Gibt die Anzahl von Systemzeit-Perioden (Systemticks) als Long zurück.
Hour(Date)	Integer	Gibt die Stunde eines Date-Werts als Integer (0..23) zurück.
IsDate(Wert)	Boolean	Ist dies (Wert, in einen String konvertiert) ein Datum?
Minute(Date)	Integer	Gibt die Minute eines Date-Werts als Integer (0..59) zurück.
Month(Date)	Integer	Gibt den Monat eines Date-Werts als Integer (1..12) zurück.
Now()	Date	Gibt das aktuelle Datum und die aktuelle Uhrzeit als Date zurück.
Second(Date)	Integer	Gibt die Sekunde eines Date-Werts als Integer (0..59) zurück.
Time()	String	Gibt die aktuelle Uhrzeit als String im Format HH:MM:SS zurück.
Timer()	Date	Gibt die Anzahl der Sekunden seit Mitternacht zurück. Wird am besten direkt zu Long konvertiert.
MonthName(Integer)	String	Gibt den Monatsnamen zurück, der dem Argument (1..12) entspricht. Benötigt OptionCompatible.
WeekDayName(Integer)	String	Gibt den Wochentagsnamen zurück, der dem Argument (1..7) entspricht. Benötigt OptionCompatible.
DateAdd(Interv, Anz, Datum)	Date	Addiert ein Intervall zu einem Datum.
DateDiff(Interv, Dat1, Dat2)	Integer	Gibt die Anzahl der Intervalle zwischen zwei Datumsangaben zurück.
DatePart(Interv, Datum)	Variant	Ermittelt einen bestimmten Wert aus einem Datum.
FormatDateTime(Datum)	String	Formatiert Datum und Zeit als String. Benötigt OptionCompatible.
TimeSerial(Std, Min, Sek)	Date	Erzeugt einen Date-Wert aus den getrennten Komponenten Std, Min, Sek.
TimeValue(“HH:MM:SS”)	Date	Erzeugt einen Date-Wert ohne Datum, eine reine Uhrzeit, intern als Double im Wert zwischen 0 und 1.

Funktion	Typ	Beschreibung
WeekDay(Date)	Integer	Gibt den Wochentag eines Date-Werts als Ziffer (1..7 entsprechend Sonntag...Samstag) zurück.
Year(Date)	Integer	Gibt das Jahr als Integer von einem Date-Wert zurück.

6.1. Kompatibilitätsprobleme

OOo hat bei der Addition einer Zahl und eines Datums immer einen Datumswert zurückgegeben. In LibreOffice wurde dieses Verhalten geändert: es wurde eine Zahl zurückgegeben. Nachdem existierende Makros Fehler produzierten, änderte LibreOffice den Code für manche Kombinationen von Datumswert und Zahl dahin, dass ein Datumswert resultiert. Zur Sicherheit sollten Sie immer das Resultat gezielt in den Typ konvertieren, den Sie erwarten:

```
CDate(2 + Now)
```

6.2. Ermittlung des aktuellen Datums und der aktuellen Uhrzeit

Basic kennt Funktionen zur Ermittlung des aktuellen Datums und der aktuellen Uhrzeit.: Date, Time und Now, s. Tabelle 35. Die Funktionen Date und Time geben einen String mit dem aktuellen Datum, beziehungsweise mit der aktuellen Uhrzeit zurück. Die Strings sind gemäß dem lokalen Gebietsschema formatiert (**Extras | Optionen | Spracheinstellungen | Sprachen**, und dann **Gebietsschema**). Die Funktion Now gibt ein Date-Objekt zurück, das sowohl das aktuelle Datum als auch die aktuelle Uhrzeit enthält.

Tipp Now gibt ein Date-Objekt zurück, das intern als Double gespeichert wird. Die Funktionen Date und Time geben beide einen String zurück.

Tabelle 35. *Datums- und Uhrzeitfunktionen in Basic.*

Funktion	Beschreibung
Date	Gibt das aktuelle Datum als String zurück.
Now	Gibt das aktuelle Datum und die aktuelle Uhrzeit als Date zurück.
Time	Gibt die aktuelle Uhrzeit als String zurück.

Die Ausgabe des Datums und der Uhrzeit ist einfach.

```
Sub PrintDateTime
  Print Date
  Print Time
  Print Now
End Sub
```

6.3. Datumsangaben, Zahlen und Strings

Basic erkennt Datumsangaben in zwei verschiedenen Stringformaten. Das naheliegende Format ist durch das lokale Gebietsschema vorgegeben. Weniger naheliegend ist das Datumsformat nach ISO 8601. Strings werden immer im lokalen Format erwartet außer bei den Routinen, die speziell für das ISO-8601-Format gelten. Die den Funktionen Date und Time übergebenen Argumente werden nach Möglichkeit in den passenden Typ konvertiert. Somit akzeptieren die meisten der in Tabelle 36 aufgelisteten Funktionen ihre Argumente sowohl als String als auch numerisch oder als Date.

Tabelle 36. *Funktionen zur Datums- und String-Konvertierung.*

Funktion	Beschreibung
CDate(Ausdruck)	Konvertiert eine Zahl oder einen String zu einem Date-Wert.

Funktion	Beschreibung
DateValue(Date)	Konvertiert einen formatierten String vom 1. Dezember 1582 bis zum 31. Dezember 9999 zu einem Date-Wert ohne Uhrzeit.
CDateFromIso(String)	Konvertiert einen ISO-8601-Datumsstring zu einem Date-Wert.
CDateToIso(Date)	Konvertiert einen Datumswert zu einem ISO-8601-Datumsstring.
IsDate(Wert)	Ist dieser Wert (in einen String konvertiert) ein Datum?

Die Funktion IsDate testet, ob ein String ein gültiges Datum darstellt. Das Argument wird immer vor der Auswertung zu einem String konvertiert, so dass mit numerischen Argumenten False zurückgegeben wird. Die Funktion IsDate testet nicht allein die Syntax – sie prüft auch, ob der String eine gültige Datumsangabe ist. "29.02.2003" scheitert, weil der Februar im Jahr 2003 nur 28 Tage hatte. Eine solche Gültigkeitsprüfung wird nicht auf die Uhrzeitkomponente des Strings ausgedehnt (s. Listing 107 und Listing 108).

Listing 107. *IsDate überprüft, ob ein String ein gültiges Datum darstellt.*

```
Sub TestIsDate
  Dim s$
  s = "" & IsDate("1. Dezember 1582 2:13:42") & " = 1. Dezember 1582 2:13:42" 'True
  s = s & Chr(10) & IsDate("2:13:42") & " = 2:13:42"        'True
  s = s & Chr(10) & IsDate("1.12.1582") & " = 1.12.1582"    'True
  s = s & Chr(10) & IsDate(Now) & " = Now"                  'True
  'True: 26 Uhr, 61 Minuten und 112 Sekunden!!
  s = s & Chr(10) & IsDate("26:61:112") & " = 26:61:112"
  'False: Zuerst zu einem String konvertiert
  s = s & Chr(10) & IsDate(True) & " = True"
  'False: Zuerst zu einem String konvertiert
  s = s & Chr(10) & IsDate(32686.22332) & " = 32686.22332"
  'False: Nur 28 Tage im Februar 03
  s = s & Chr(10) & IsDate("29.02.2003") & " = 29.02.2003"
  MsgBox s
End Sub
```

Es besteht offenbar eine Unstimmigkeit in der Funktion IsDate, nämlich dass "29.02.2003" ungültig ist, "26:61:112" aber nicht. Wenn bei Uhrzeiten ein Zeitabschnitt zu groß ist, wird der Überschuss einfach zum nächsten Abschnitt addiert. Zum Beispiel sind 61 Minuten eine Stunde und eine Minute. Und mit 112 Sekunden wird eine Minute und 52 Sekunden dem errechneten Zeitwert zugeschlagen. In Listing 108 (mit Bild 47) wird der Zusammenhang deutlich. Beachten Sie in Zeile 2, dass 30 Stunden zu sechs Stunden werden, wobei der Tag um eins erhöht wird.

Listing 108. *Die Uhrzeit-Konvertierung ist gewöhnungsbedürftig.*

```
Sub ExampleTimeConversions
  On Error GoTo Oops
  Dim Dates()
  Dim i As Integer
  Dim s As String
  Dates() = Array("1.1.1 00:00:00 ", " 1.1.1 22:40:00  ", " 1.1.1 30:40:00  ",_
                  " 1.1.1 30:100:00 ", " 1.1.1 30:100:100")
  For i = LBound(Dates()) To UBound(Dates())
    s = s & CStr(i) & "   " & Dates(i) & " => "
    s = s & CDate(Dates(i))
    s = s & Chr$(10)
  Next
  MsgBox s, 0, "Seltsame Uhrzeiten"
  Exit Sub
  Oops:
  s = s & " Error"
```

```
  Resume Next
End Sub
```

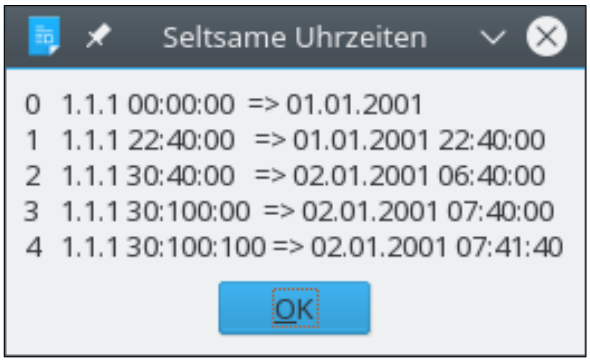


Bild 47. *Scheinbar ungültige Uhrzeiten sind gültig.*

Abgesehen vom seltsamen Verhalten bei Uhrzeiten werden Datum und Uhrzeit problemlos konvertiert. Allerdings gibt es ungültige Datumsangaben beim Sprung vom Julianischen zum Gregorianischen Kalender.

6.4. Lokal formatierte Datumsangaben

Die Funktion CDate konvertiert einen String oder eine Zahl zu einem Date-Wert. Sie geht vom Format des lokalen Gebietsschemas aus, inklusive der Uhrzeit. Die Funktion DateValue entfernt den Uhrzeitanteil durch Abschneiden des Dezimalteils des zugrundeliegenden Double-Werts, s. Listing 109. Das führt allerdings bei manchen Datumswerten zu unerwarteten Ergebnissen.

Listing 109. *CDate gibt Datum und Uhrzeit zurück, DateValue entfernt die Uhrzeit.*

```
Sub DateValueTest
  Print Now                  'Zum Beispiel 19.07.2011 16:05:53
  Print DateValue(Now)       'Zum Beispiel 19.07.2011
End Sub
```

Die Standardsprache auf meinen (das heißt hier: des Übersetzers) Rechnern ist „Deutsch (Deutschland)“. Zur Umstellung auf ein anderes Gebietsschema gehen Sie über das Menü **Extras | Optionen | Spracheinstellungen | Sprachen** und wählen dann das **Gebietsschema**.

Testen Sie den Code in Listing 110 mit verschiedenen Gebietsschemata.

Listing 110. *Gibt Datumsangaben auf der Grundlage des lokalen Gebietsschemas aus.*

```
Sub PrintLocaleSpecific
  Dim d As Date
  d = CDate("1.2.3")    'Abhängig vom Gebietsschema: 1.2.3 oder 1/2/3
  Print d               'Ausgabe gemäß dem Gebietsschema
  Print Year(d)
  Print Month(d)
  Print Day(d)
End Sub
```

Ich habe den Code in Listing 110 mit vier verschiedenen Gebietsschemata ausgeführt: Englisch (USA), Englisch (Großbritannien), Französisch (Frankreich) und Deutsch (Deutschland). Die Ergebnisse sehen Sie in der Tabelle 37. Das Format zur Ausgabe eines Datums ist spezifisch für das Gebietsschema, wie man in der Deutschland-Spalte sehen kann. Die Variable d mit CDate("1.2.3") (Deutschland) zu initialisieren anstatt mit CDate("1/2/3") (anderes Gebietsschema) ändert nichts an der Ausgabe in Tabelle 37.

Tabelle 37. *Das Gebietsschema beeinflusst das Datum.*

Code	USA	UK	Frankreich	Deutschland
`Print d`	"01/02/2003"	"01/02/2003"	"01/02/2003"	"01.02.2003"
`Print Year(d)`	2003	2003	2003	2003
`Print Month(d)`	1	2	2	2
`Print Day(d)`	2	1	1	1

6.5. Datumsangaben nach ISO 8601

Die Internationale Norm ISO 8601 legt die numerische Darstellung des Datums und der Uhrzeit fest. Diese genormte Schreibweise dient der Vermeidung von Konfusion in der internationalen Kommunikation durch zahlreiche unterschiedliche Schreibweisen und erhöht die Portabilität von Benutzerschnittstellen für Computer. ISO 8601 bietet noch weitere wichtige Vorteile für die Nutzung in Computern im Gegensatz zu anderen traditionellen Datums- und Uhrzeitformaten.

Die internationale Normnotierung für das Datum ist JJJJ-MM-TT, erst das Jahr vierstellig, dann der Monat zweistellig, schließlich der Tag zweistellig. Das Jahr wird nach dem Gregorianischen Kalender gezählt. Zum Beispiel wird der 8. März 2003 als 2003-03-08 geschrieben. Die Trennzeichen sind optional, so dass das Datum auch die Form 20030308 haben kann. CDateToIso gibt dieses Format zurück, s. Listing 126. Andere Komponenten des Datumsformats gehen über den Rahmen dieses Buches hinaus. Das ISO-Format hat einige Vorteile:

- ISO 8601 ist als String leicht vergleichbar und sortierbar. Daher verwende ich am liebsten dieses Format, wenn ich einem Dateinamen ein Datum anfüge.
- ISO 8601 kann mit Software leicht gelesen und geschrieben werden, denn es ist keine Übersetzung der Monatsnamen erforderlich.
- ISO 8601 ist unabhängig von Sprachen und Ländern.
- Gemessen an anderen Datumsformaten gibt es keine Mehrdeutigkeit. In anderen Formaten ist es oft unklar, ob der Monat oder der Tag am Anfang steht. In Europa ist es zum Beispiel üblich, den fünften Tag im Juni 2003 als 5/6/03 oder 5.6.03 zu bezeichnen, wohingegen in den Vereinigten Staaten – und zwar ganz allgemein in Nord- und Südamerika – dasselbe Datum gewöhnlich als 6/5/03 erscheint. Man kann sich leicht vertun, vor allem, wenn um den Monatsbeginn herum Rechnungen fällig werden! In der Tabelle 37 ist das in den Zeilen Print Month(d) und Print Day(d) zu erkennen. Im Listing 111 werden Datumsangaben zum entsprechenden ISO-8601-String konvertiert.

Listing 111. *Konvertierung nach ISO 8601.*

```
Sub ExampleCDateToISO
  Print CdateToISO("30.12.1899")          '18991230
  Print CDateToISO(Now)                   '20110719
  Print CDateFromISO("20030313")          '13.03.2003
End Sub
```

6.6. Probleme mit Datumsangaben

Über Datums- und Uhrzeitangaben muss man sich normalerweise kaum Gedanken machen. Man benutzt die Datums- und Uhrzeitfunktionen, und das Ergebnis entspricht den Erwartungen. Leider kann das aber schiefgehen, wenn man mit dem Datum zurück in die Vergangenheit geht. In jeder Funktion sind drei besondere Datums- und Uhrzeitangaben zu beachten:

- 30. Dezember 1899 um 00:00:00. Dieser Zeitpunkt wird intern durch den numerischen Wert null dargestellt. Jeder reine Uhrzeitswert, das heißt ohne Datumsanteil, bezieht sich somit auf den 30. Dezember 1899!

- 4. Oktober 1582. Der Julianische Kalender endet.
- 15. Oktober 1582. Der Gregorianische Kalender beginnt.

Im Laufe der Geschichte wurden und werden auch heute rund um die Welt ganz verschiedene Kalendersysteme verwendet. Der Gregorianische Kalender, auf dem Basic aufbaut, ist nahezu universell in Gebrauch. Sein Vorgänger ist der Julianische Kalender. Diese beiden Kalender sind fast gleich, sie unterscheiden sich nur in der Behandlung der Schaltjahre. Der Julianische Kalender hat alle vier Jahre ein Schaltjahr, wohingegen der Gregorianische Kalender zwar auch alle vier Jahre ein Schaltjahr hat, aber nicht in Hunderter-Jahren, die nicht genau durch 400 teilbar sind.

Der Wechsel vom Julianischen zum Gregorianischen Kalender wurde im Oktober 1582 vollzogen, nach einem Schema, das Papst Gregor XIII dekretierte. Der Julianische Kalender wurde bis zum Donnerstag, den 4. Oktober 1582 verwendet. Zu diesem Zeitpunkt wurden 10 Tage übersprungen und der nächste Tag, ein Freitag, wurde zum 15. Oktober 1582. Üblicherweise wird für ein Datum bis zum 4. Oktober 1852 der Julianische Kalender benutzt und für ein Datum vom 15. Oktober 1582 an der Gregorianische. Somit existiert zwar eine Lücke von zehn Tagen in Kalenderdaten, aber ohne Diskontinuität der Julianischen Zählung oder der Wochentage. Astronomen bevorzugen jedoch die Julianische Datumsrechnung, weil sie die Lücke von zehn Tagen nicht haben, denn numerische Berechnungen vertragen sich nicht so gut mit unstetigen Daten. Wie man im Listing 121 sehen kann, fußt die Ausgabe eines Datums auf dem Gregorianischen Kalender, der Zugriff auf einzelne Komponenten allerdings auf dem Julianischen.

Die Norm ISO 8601, eingeführt zur Standardisierung des Austausches von Datum und Uhrzeit, bringt gleichzeitig eine Komplizierung mit sich. Die Norm legt fest, dass jedes Datum kontinuierlich sein muss. Der Wechsel zum Julianischen Kalender verletzt also die Norm, denn am Tag des Wechsels wäre das Datum nicht kontinuierlich.

Die folgenden Beispiele demonstrieren das Konvertieren von Datum/Uhrzeit mit CDate, DateValue und CDateToIso. DateValue verwendet den von CDate zurückgegebenen Wert. Die Werte für ein Datum ab 1900 sind wie nach der Norm zu erwarten, s. Tabelle 38. Der 1. Januar 2001 ist 36892 Tage nach dem 30. Dezember 1899, und der 1. Januar 1900 ist 2 Tage nach dem 30. Dezember 1899.

Tabelle 38. *Datumswerte nach dem 1. Januar 1900 funktionieren gut.*

Datum/Uhrzeit	DateValue	CDate	CDateToIso
01.01.1900 00:00	2	2	19000101
01.01.1900 06:00	2	2,25	19000101
02.01.1900 00:00	3	3	19000102
02.01.1900 06:00	3	3,25	19000102
01.01.2001 00:00	36892	36892	20010101
01.01.2001 06:00	36892	36892,25	20010101
01.01.2001 12:00	36892	36892,5	20010101

Werte nahe dem 30. Dezember 1899 sind nicht frei von Fehlern.

- DateValue(CDate("30.12.1899")) erzeugt einen Error am 30. Dezember 1899. Das mag beabsichtigt sein, denn DateValue erzeugt einen Error mit einem reinen Uhrzeitwert, dessen ganzzahliger Anteil ja null ist und sich daher nicht von einem Datum/Uhrzeit-Wert am 30. Dezember 1899 unterscheidet. Allerdings ergibt DateValue("30.12.1899") das korrekte Ergebnis „00:00:00“, und CDbl(DateValue("30.12.1899") ergibt 0 -korrekt.
- DateValue liefert für jedes Datum vor dem 30. Dezember 1899 falsche Ergebnisse, für alle Uhrzeiten außer Mitternacht.

- CDateToIso produzierte in früheren Versionen falsche Datumswerte für die Zeit vor dem 1.1.1900. In AOO 4.1.4 und LO 5.3 sind aber alle Werte korrekt.

Tabelle 39. *Datumswerte nahe dem 30. Dezember 1899 sind problematisch.*

Datum/Uhrzeit	DateValue	CDate	CDateToISO
28.12.1899 00:00	-2	-2	18991228
28.12.1899 06:00	-1	-1,75	18991228
29.12.1899 00:00	-1	-1	18991229
29.12.1899 06:00	Error	-0,75	18991229
30.12.1899 00:00	Error	0	18991230
30.12.1899 06:00	Error	0,25	18991230
31.12.1899 00:00	1	1	18991231
31.12.1899 06:00	1	1,25	18991231

Ein wirklich ungültiges Datum erzeugt pflichtgemäß einen Error, zum Beispiel ein Datum zwischen dem Ende des Julianischen und dem Beginn des Gregorianischen Kalenders. Die Bugs in DateValue bleiben gleich bei Uhrzeiten außerhalb Mitternacht.

Tabelle 40. *Datumswerte nahe dem Wechsel vom Julianischen zum Gregorianischen Kalender.*

Datum/Uhrzeit	DateValue	CDate	CDateToIso
04.10.1582 00:00	-115859	-115859	15821014
04.10.1582 06:00	-115858	-115858,75	15821014
05.10.1582 00:00	Error	Error	Error
05.10.1582 06:00	Error	Error	Error
15.10.1582 00:00	-115858	-115858	15821015
15.10.1582 06:00	-115857	-115857,75	15821015

Listing 112. *Demonstration der Seltsamkeiten bei der Datumsbehandlung.*

```
Sub OddDateTimeBehavior
  On Error GoTo Oops
  Dim Dates()
  Dim i As Integer
  Dim s As String
  Dates() = Array("04.10.1582 00:00:00", "04.10.1582 06:00:00", _
                  "05.10.1582 00:00:00", "05.10.1582 06:00:00", _
                  "15.10.1582 00:00:00", "15.10.1582 06:00:00", _
                  "28.12.1899 00:00:00", "28.12.1899 06:00:00", _
                  "29.12.1899 00:00:00", "29.12.1899 06:00:00", _
                  "30.12.1899 00:00:00", "30.12.1899 06:00:00", _
                  "31.12.1899 00:00:00", "31.12.1899 06:00:00", _
                  "01.01.1900 00:00:00", "01.01.1900 06:00:00", _
                  "02.01.1900 00:00:00", "02.01.1900 06:00:00", _
                  "1.1.1 00:00:00",      "1.1.1 06:00:00", _
                  "1.1.1 12:00:00")
  For i = LBound(Dates()) To UBound(Dates())
    s = s & CStr(i) & "   " & Dates(i) & " => "
    s = s & CDbl(DateValue(CDate(Dates(i))))
    s = s & " => "
    s = s & CDbl(CDate(Dates(i))) & " => " & CDateToIso(Dates(i))
    s = s & Chr$(10)
  Next
  MsgBox s, 0, "Seltsame Uhrzeitwerte"
```

```
  Exit Sub
  Oops:
  s = s & " Error"
  Resume Next
End Sub
```

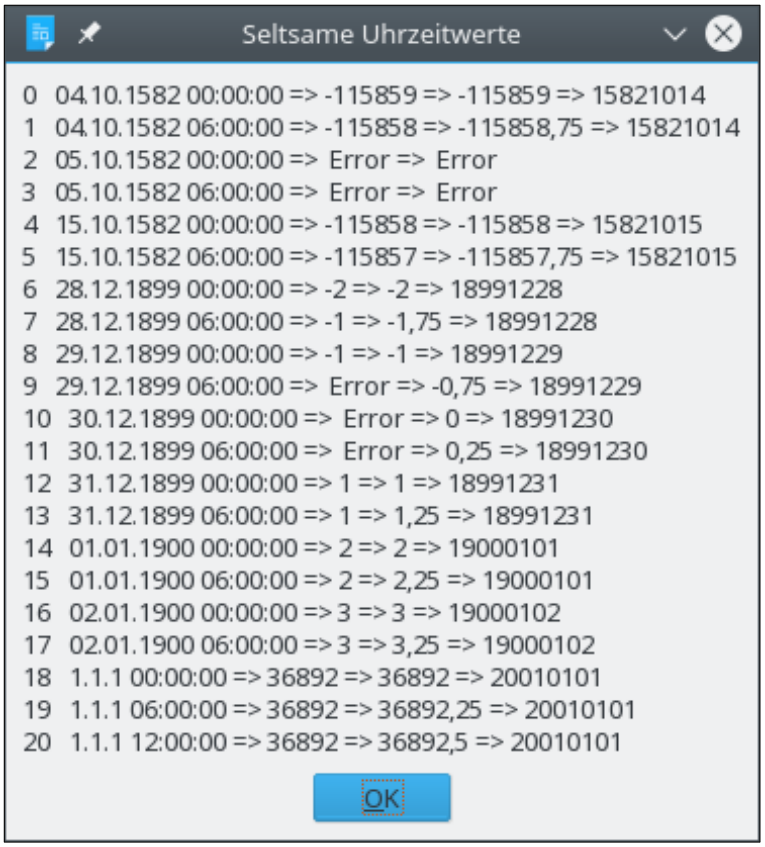


Bild 48. *Manche Datumswerte werden schlecht konvertiert.*

Achtung DateValue versagt mit einem Laufzeitfehler bei einem Datum, dessen Tageskomponente null ist, zum Beispiel DateValue(CDate("30.12.1899 06:00:00")). Noch frühere Datumsangaben führen zu falschen Ergebnissen. Man kann es auch als Bug ansehen, dass Datum und Uhrzeit denselben Datentyp verwenden, denn so ist es nicht möglich, zwischen einer reinen Uhrzeit und dem Datum 30.12.1899 zu unterscheiden.

DateValue schneidet zur Datumsbestimmung den Dezimalteil der Zahl ab. Die Funktion Int hingegen rundet immer in Richtung negativ unendlich, wodurch das Resultat korrekt wird, s. Listing 113. Sie erinnern sich, dass die Funktion Int in Richtung negativ unendlich rundet und einen Double-Wert zurückgibt.

Listing 113. *Rundung in Richtung negativ unendlich und Konvertierung zu Date.*

```
Function SafeDateValue(v) As Date
  SafeDateValue = CDate(Int(CDate(v)))
End Function
```

SafeDateValue in Listing 113 korrigiert das falsche Verhalten. Listing 114 nimmt Listing 112 mit SafeDateValue wieder auf, so dass sich nun korrekte Werte ergeben.

Listing 114. *Rundung in Richtung negativ unendlich und Konvertierung zu Date.*

```
Sub SafeDateTimeBehavior
REM Sicheres Datum/Uhrzeit-Verhalten
```

```
    On Error GoTo Oops
    Dim Dates()
    Dim i As Integer
    Dim s As String
    Dates() = Array("04.10.1582 00:00:00", "04.10.1582 06:00:00", _
                    "05.10.1582 00:00:00", "05.10.1582 06:00:00", _
                    "15.10.1582 00:00:00", "15.10.1582 06:00:00", _
                    "28.12.1899 00:00:00", "28.12.1899 06:00:00", _
                    "29.12.1899 00:00:00", "29.12.1899 06:00:00", _
                    "30.12.1899 00:00:00", "30.12.1899 06:00:00", _
                    "31.12.1899 00:00:00", "31.12.1899 06:00:00", _
                    "01.01.1900 00:00:00", "01.01.1900 06:00:00", _
                    "02.01.1900 00:00:00", "02.01.1900 06:00:00", _
                    "1.1.1 00:00:00",      "1.1.1 06:00:00", _
                    "1.1.1 12:00:00")
    For i = LBound(Dates()) To UBound(Dates())
      s = s & CStr(i) & "   " & Dates(i) & " => "
      s = s & CDbl(SafeDateValue(Dates(i)))
      s = s & " => "
      s = s & CDbl(CDate(Dates(i))) & " => " & CDateToISO(SafeDateValue(Dates(i)))
      s = s & Chr$(10)
    Next
    MsgBox s, 0, "Korrekte Uhrzeitwerte"
    Exit Sub
    Oops:
    s = s & " Error"
    Resume Next
  End Sub
```

6.7. Entnahme einzelner Komponenten eines Datums

Date-Objekte basieren auf Fließkommazahlen des Typs Double, so dass mathematische Operationen und Vergleiche mit Date-Objekten durchgeführt werden können. Die Funktionen Date und Time geben jedoch Strings zurück, so dass sie für Berechnungen nicht zur Verfügung stehen. Basic stellt Funktionen bereit, die einzelnen Komponenten eines Datums zu ermitteln, s. Tabelle 41.

Tabelle 41. *Funktionen zur Entnahme einzelner Komponenten eines Basic-Datums.*

Funktion	Beschreibung
Year	Gibt das Jahr eines Datumswerts als Integer zurück.
Month	Gibt den Monat eines Datumswerts als Integer zurück.
Day	Gibt den Tag eines Datumswerts als Integer zurück.
Hour	Gibt die Stunde eines Datumswerts als Integer zurück.
Minute	Gibt die Minute eines Datumswerts als Integer zurück.
Second	Gibt die Sekunde eines Datumswerts als Integer zurück.
WeekDay	Gibt je nach Wochentag eines Datumswerts einen Integer-Wert von 1 bis 7 zurück, gezählt von Sonntag bis Samstag.

Die Funktionen in Tabelle 41 erwarten alle ein Date-Objekt, das intern auf Double basiert. Basic konvertiert das Argument automatisch zum passenden Typ, wenn es möglich ist. Die Funktion Date gibt einen String zurück, der keine Uhrzeit enthält, denn alles rechts vom Dezimaltrenner ist abgetrennt. Somit fehlt für die Ausgabe auch jede Information über Stunde, Minute und Sekunde. So ähnlich ist es auch bei der Funktion Time. Sie gibt einen String ohne Datumsangabe zurück, denn alles links vom Dezimaltrenner ist auf null gesetzt. Damit entspricht der Wert dem vom 30. Dezember 1899.

```
Print "Jahr = " & Year(0.223)       '1899, 0 als Datum meint den 30. Dezember 1899
Print "Jahr = " & Year(Time)        '1899, keine Datumsinformation von Time()
Print "Monat = " & Month(Date)      'Aktueller Monat
Print "Tag = " & Day(Now)           'Aktueller Tag des Monats. Now = Datum und Uhrzeit
Print "Stunde = " & Hour(Date)      '0, keine Uhrzeitinformationen von Date()
Print "Minute = " & Minute(Now)     'Aktuelle Minute
Print "Sekunde = " & Second(Time)   'Aktuelle Sekunde
```

Mit der Funktion WeekDay bestimmen Sie den Wochentag. Manche Kalender beginnen die Woche mit dem Montag, manche mit dem Sonntag. Basic setzt voraus, dass der Sonntag der erste Tag der Woche ist, s. Listing 115.

Listing 115. *Ermittlung des Wochentags.*

```
Sub ExampleWeekDayText
  Print "Heute ist " & WeekDayText(Date)
End Sub

Function WeekDayText(d) As String
  Select Case WeekDay(d)
    Case 1
      WeekDayText = "Sonntag"
    Case 2
      WeekDayText = "Montag"
    Case 3
      WeekDayText = "Dienstag"
    Case 4
      WeekDayText = "Mittwoch"
    Case 5
      WeekDayText = "Donnerstag"
    Case 6
      WeekDayText = "Freitag"
    Case 7
      WeekDayText = "Samstag"
  End Select
End Function
```

Der Funktion DatePart können Sie als erstes Argument einen String übergeben, der das gewünschte Datumsintervall festlegt. Tabelle 42 zeigt die Ergebnisse, die Sie erhalten, wenn Sie DatePart mit dem 15. September 2010 um 19:13:20 aufrufen.

Tabelle 42. *Intervallkennzeichner in DatePart.*

Format	Beschreibung	Ergebnis
yyyy	Vierstelliges Jahr	2010
q	Quartal	3
m	Monat	9
y	Tag des Jahres	258
w	Wochentag	4
ww	Kalenderwoche	38
d	Tag des Monats	15
h	Stunde	19
n	Minute	13
s	Sekunde	20

Listing 116. *Ermittlung von Datumskomponenten mit DatePart.*

```
Sub ExampleDatePart
  Dim theDate As Date
  Dim f
  Dim i As Integer
  Dim s$
  theDate = Now
  f = Array("yyyy", "q", "m", "y", "w", "ww", "d", "h", "n", "s")
   s = "Jetzt = " & theDate & Chr$(10)
  For i = LBound(f) To UBound(f)
    s = s & "DatePart(" & f(i) & ", " & theDate & ") = " _
          & DatePart(f(i), theDate) & Chr$(10)
  Next
  MsgBox s
End Sub
```

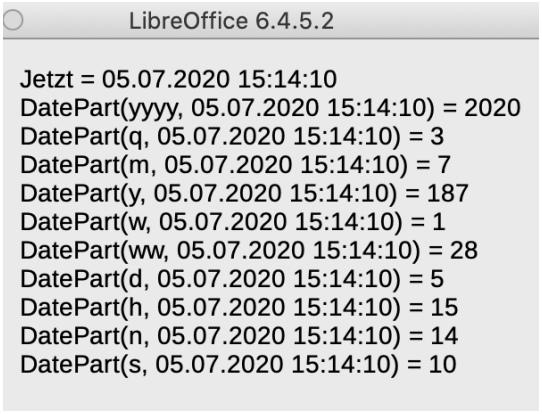


Bild 49. *Ausgabe Datumskomponenten mit DatePart.*

Tipp Ein deutscher Benutzer meldete, dass DatePart nur mit CompatibilityMode(True) funktioniere (Beim Übersetzer funktioniert es auch ohne CompatibilityMode). In der neuesten US/Englischen Version arbeitet MonthName nur mit CompatibilityMode(True).

Ein optionales drittes Argument in DatePart legt fest, an welchem Tag die Woche beginnen soll (s. Tabelle 43), ein optionales viertes Argument legt fest, wann das Jahr beginnen soll. Diese Argumente beeinflussen den Wochentag und die Kalenderwoche, nicht aber andere Werte wie den Tag des Jahres.

Tabelle 43. *Werte für Wochenbeginn und Jahresbeginn in DatePart.*

Wert	Beschreibung Wochenbeginn	Beschreibung Jahresbeginn
0	Standardwert des Systems	Standardwert des Systems
1	Sonntag (Standard)	Woche 1 ist die Woche mit dem 1. Januar (Standard).
2	Montag	Woche 1 ist die erste Woche des Jahres mit vier oder mehr Tagen.
3	Dienstag	Woche 1 ist die erste Woche, die nur Tage des neuen Jahres enthält.
4	Mittwoch	
5	Donnerstag	
6	Freitag	
7	Samstag	

Wenn man Teile des Datums extrahiert hat, möchte man die Werte auch in einer leicht lesbaren Form sehen. Mit der Funktion MonthName wird die Monatsnummer zu einem Monatsnamen konvertiert. Wenn Sie das zweite, optionale Argument auf True setzen, wird der Monatsname abgekürzt.

Listing 117. *Ausgabe des Monats als String.*

```
Sub ExampleMonthName
  Dim i%
  Dim s$
  CompatibilityMode(True)
  For i = 1 To 12
    s = s & i & " = " & MonthName(i, True) & " = " & MonthName(i) & Chr$(10)
  Next
  CompatibilityMode(False)
  MsgBox s, 0, "Monatsname"
End Sub
```

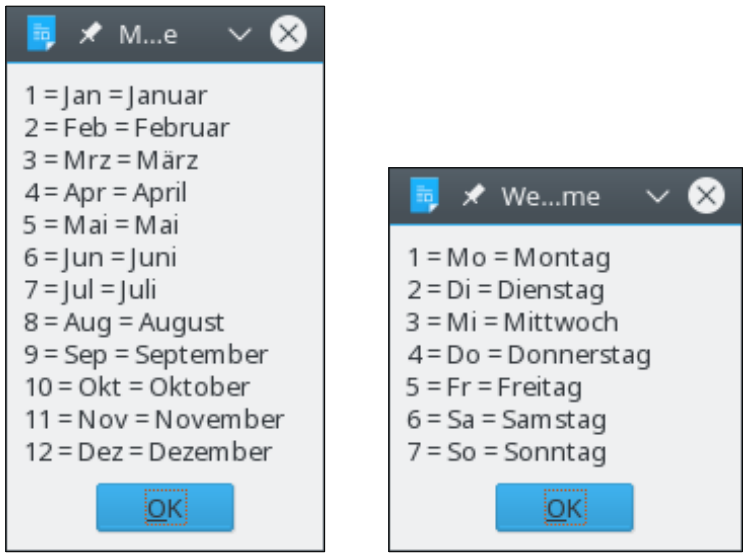


Bild 50. *Mit MonthName und WeekDayName wird eine Ganzzahl zum Namen konvertiert.*

Ähnlich wie MonthName gibt WeekDayName den Wochentag als String zurück. Erstaunlich ist, dass auf manchen Systemen CompatibilityMode sowohl für MonthName als auch für WeekDayName auf True gesetzt sein muss, auf anderen aber nicht für MonthName. WeekDayName unterstützt noch ein drittes Argument zur Festlegung des Wochenbeginns, s. Tabelle 44.

Listing 118. *Ausgabe des Wochentags als String.*

```
Sub ExampleWeekDayName
  Dim i%
  Dim s$
  CompatibilityMode(True)
  For i = 1 To 7
    s = s & i & " = " & WeekDayName(i, True) & " = " & WeekDayName(i) & Chr$(10)
  Next
  CompatibilityMode(False)
  MsgBox s, 0, "Wochentagsname"
End Sub
```

Tabelle 44. *Das dritte Argument für WeekDayName legt den ersten Tag der Woche fest.*

Wert	Beschreibung
0	Wert der nationalen Standardspracheinstellung

Wert	Beschreibung
1	Sonntag (Standard)
2	Montag
3	Dienstag
4	Mittwoch
5	Donnerstag
6	Freitag
7	Samstag

Mit der Funktion WeekDay erhält man den Wochentag aus einem Datum. Das optionale zweite Argument bestimmt den Tag, an dem die Woche beginnt. Die Werte sind dieselben wie für WeekDayName (s. Tabelle 44), außer dass 0 die Systemstandardeinstellung bezeichnet.

```
Print WeekDay(Now)
```

Zur Ausgabe von Datum und Uhrzeit in üblichen Formaten dient die Funktion FormatDateTime. Das erste Argument ist das Datum. Das zweite Argument ist optional und legt das Format fest, in dem das Datum erscheinen soll (s. Tabelle 45).

Listing 119. *Formatierter Datum/Uhrzeit-String.*

```
Sub ExampleFormatDateTime
  Dim s$, i%
  Dim d As Date
  d = Now
  CompatibilityMode(True)
  s = "FormatDateTime(d) = " & FormatDateTime(d) & Chr$(10)
  For i = 0 To 4
    s = s & "FormatDateTime(d, " & i & ") = " & FormatDateTime(d, i) & Chr$(10)
  Next
  CompatibilityMode(False)
  MsgBox s, 0, "FormatDateTime"
End Sub
```

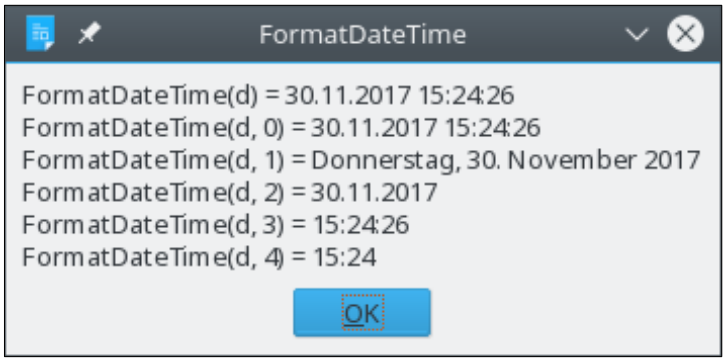


Bild 51. *Formatierung von Datum und Uhrzeit mit FormatDateTime.*

Tabelle 45. *Das zweite Argument zu FormatDateTime zur Formatauswahl.*

Wert	Beschreibung
0	Standardformat mit kurzem Datum und langer Uhrzeit
1	Datum im Langformat ohne Uhrzeit
2	Datum im Kurzformat ohne Uhrzeit
3	Uhrzeit im Regionalformat des Rechners
4	Stunden und Minuten(hh:mm, Stunden im 24-Stunden-Format)

6.8. Datumsarithmetik

Intern wird ein Datum als Fließkommazahl dargestellt. Der Dezimalteil steht für die Uhrzeit und der ganzzahlige Teil steht für das Datum. Diese Darstellung erlaubt mathematische Operationen, aber man muss ein paar Dinge bedenken, zum Beispiel bei der Addition:

```
Now + 1      'Plus ein Tag
Now + 1/24   'Plus eine Stunde
Now + 365    'Plus ein Jahr, aber man beachte die Schaltjahre.
```

Mit DateAdd wird der Prozess vereinfacht. Das dritte Argument ist das Datum. Das erste Argument (s. Tabelle 42) bestimmt, wie das zweite Argument verwendet wird. Mit DateAdd sieht die Liste oben so aus:

```
Print DateAdd("d", 1, Now)     'Plus ein Tag.
Print DateAdd("h", 1, Now)     'Plus eine Stunde.
Print DateAdd("yyyy", 1, Now)  'Plus ein Jahr.
```

Mit DateDiff wird die Anzahl der „Intervalle“ zwischen zwei Datumswerten bestimmt, zum Beispiel die Anzahl der dazwischen liegenden Wochen. Mit dem ersten Argument wird das Intervall gewählt (s. Tabelle 42). Das zweite Argument ist das erste Datum, und das dritte Argument ist das zweite Datum. Das vierte und fünfte Argument sind optional. Sie legen den ersten Tag der Woche beziehungsweise die erste Woche des Jahres fest (s. Tabelle 43).

```
Print DateDiff("yyyy", "13/03/1965", Date(Now))  'Jahre vom 13. März 1965 bis heute.
Print DateDiff("d",    "13/03/1965", Date(Now))  'Tage vom 13. März 1965 bis heute.
```

6.9. Ein Datum aus Einzelkomponenten zusammensetzen

Mit den Funktionen Hour, Minute, Second, Year, Month und Day wird ein Datum in Einzelkomponenten zerlegt. Mit den Funktionen DateSerial und TimeSerial werden die Werte wieder zusammengefügt. Die Funktion DateSerial erstellt ein Date-Objekt aus Jahr, Monat und Tag, TimeSerial erstellt ein Date-Objekt aus Stunde, Minute und Sekunde.

```
Print DateSerial(2003, 10, 1) '01.10.2003
Print TimeSerial(13, 4, 45)   '13:04:45
```

Das erste Argument zu DateSerial ist das Jahr, das zweite der Monat und das letzte der Tag. Wenn Monat oder Tag keine gültigen Werte sind, wird ein Laufzeitfehler erzeugt. Jahre über 100 werden direkt verwendet. Jahre unter 100 werden jedoch zu 1900 addiert. Zweistellige Jahre werden also auf die Jahre 1900 und später abgebildet, s. Listing 120.

Listing 120. *DateSerial addiert 1900 zu Jahren unter 100.*

```
Sub ExampleDateSerial
  On Error Goto OOPS
  Dim x
  Dim i%
  Dim s$

  x = Array(2003, 10,  1, _
            1899, 12, 31, _
            1899, 12, 30, _
            1899, 12, 29, _
            1899, 12, 28, _
              99, 10,  1, _
               3, 10,  1, _
               0,  1,  1, _
              -3, 10,  1, _
             -99, 10,  1, _
```

```
            -100, 10,  1, _
           -1800, 10,  1, _
           -1801, 10,  1)
  i = LBound(x)
  Do While i < UBound(x)
    s = s & DateSerial(x(i), x(i + 1), x(i + 2))
    s = s & " <= (" & ToStrWithLen(x(i), 4) & ","
    s = s & ToStrWithLen(x(i + 1), 3) & ","
    s = s & ToStrWithLen(x(i + 2), 3) & ")"
    s = s & Chr$(10)
    i = i + 3
  Loop
  MsgBox s
  Exit Sub
  OOPS:
    s = s & ERROR
    Resume Next
End Sub

REM Gibt eine Zahl x rechtsbündig als String einer bestimmten Länge iLen zurück.
Function ToStrWithLen(x, iLen%) As String
  Dim s$
  s = CStr(x)
  If Len(s) < iLen Then
    s = Space(iLen - Len(s)) & s
  End If
  ToStrWithLen = s
End Function
```

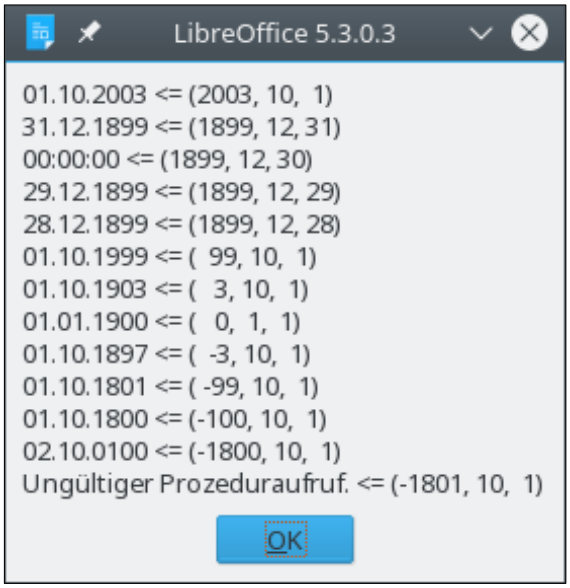


Bild 52. *DateSerial erstellt ein Datum aus den Werten für Jahr, Monat und Tag.*

Achtung DateSerial addiert 1900 zu Jahren unter 100. Negative Jahre werden also subtrahiert. Ein Fehler resultiert, wenn das errechnete Jahr immer noch unter 100 liegt. Somit sind nur Jahre von -1800 bis -1 möglich, was aber eher ein Fehler sein sollte. Der Typ Date kann sehr wohl mit einem Datum vor dem 1.1.0000 umgehen, DateSerial aber offensichtlich nicht.

Die Datumsausgabe fußt auf dem Gregorianischen Kalender, der Zugriff auf einzelne Komponenten allerdings auf dem Julianischen.

Listing 121. *DateSerial akzeptiert ein Gregorianisches Datum vor dem 15.10.1582.*

```
Print DateSerial(1582, 10, 15)       '15.10.1582
Print DateSerial(1582, 10, 14)       '04.10.1582
Print Day(DateSerial(1582, 10, 14)) '14
```

6.10. Messung kurzer Zeitverläufe

Einfache Zeitdifferenzen erhalten Sie, indem Sie zwei Datumswerte subtrahieren. Zum Beispiel ermittelt CLng(Now - CDate("1/1/2000")) die Anzahl der Tage, die seit dem 1. Januar 2000 verstrichen sind. Basic ermöglicht, die verstrichene Zeit in Sekunden (Timer) oder in Systemticks (GetSystemTicks) zu ermitteln, s. Tabelle 46. Intern haben Rechner eine Systemuhr, die in einer bestimmten Frequenz läuft. Welche Frequenz es ist, ist abhängig von der Hardware. In Rechnern mit Intel-Hardware ist es 1/17 Sekunde. Jedes Vorrücken des Zeitgebers um 1 nennt man einen „Systemtick“. Die von GetSystemTicks gelieferte Anzahl basiert immer auf Millisekunden, auch wenn die Systemuhr nicht so genau ist.

Tabelle 46. *Basic-Funktionen zur Ermittlung verstrichener Zeit.*

Funktion	Beschreibung
GetSystemTicks	Gibt die Anzahl der Systemticks als Long zurück. Die Zeitangabe ist immer in Millisekunden, auch wenn der Systemzeitgeber weniger genau ist.
Timer	Gibt die Anzahl der Sekunden seit Mitternacht als Date zurück. Wird am besten direkt zu Long konvertiert.

Mit GetSystemTicks erhält man die systemabhängige Anzahl an Ticks des Systemzeitgebers. Diesen Wert nutzt man gerne zur Laufzeitmessung interner Operationen, weil er eine höhere Präzision hat als die internen Zeit- und Datumsfunktionen, s. Listing 122 und Bild 53. Der Rückgabewert ist Long.

Listing 122. *Messung verstrichener Zeit.*

```
Sub ExampleElapsedTime
  Dim startTicks As Long
  Dim endTicks As Long
  Dim startTime As Date
  Dim endTime As Date
  startTicks = GetSystemTicks()
  startTime = Timer
  Wait(200) 'Unterbricht die Ausführung für 0,2 Sekunden
  endTicks = GetSystemTicks()
  endTime = Timer
  MsgBox "Nach einer Wartezeit von 200 ms (0,2 Sekunden), " & Chr$(10) &_
         "Systemticks = " & CStr(endTicks - startTicks) & Chr$(10) &_
         "Verstrichene Zeit = " & CStr((endTime - startTime)) &_
         " Sekunden" & Chr$(10), 0, "Verstrichene Zeit"
End Sub
```

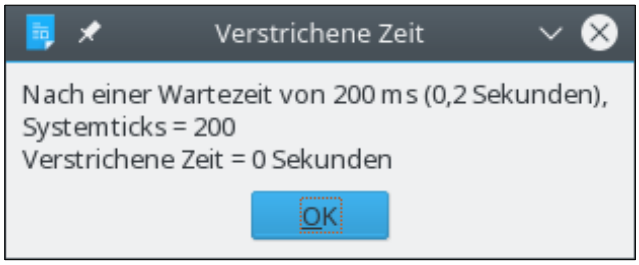


Bild 53. *GetSystemTicks hat eine bessere Auflösung als Now.*

Führen Sie das Makro in Listing 122 mehrere Male aus. Manchmal zeigt es 0 Sekunden, und manchmal zeigt es 1 Sekunde. Die Auflösung ist zu grob für ganz kurze Zeitintervalle.

Die Funktion Timer gibt die Anzahl der Sekunden seit Mitternacht als Date zurück. Das Problem damit ist, dass um 10 Sekunden nach Mitternacht der Rückgabewert 10 beträgt. Das Date-Objekt interpretiert 10 jedoch als „10 Tage“. Konvertieren Sie den Rückgabewert (Typ Date) direkt zu einem numerischen Typ (CLng oder CDbl), um die verstrichenen Sekunden zu erhalten.

```
Dim nSeconds As Long
nSeconds = Timer
Print "Anzahl Sekunden = " & nSeconds
Print "Anzahl Sekunden = " & CLng(Timer)
```

Tipp Die Funktion Timer gibt die Anzahl der Sekunden seit Mitternacht zurück. Bei der Messung von Zeitintervallen, die vor Mitternacht beginnen und nach Mitternacht enden, liefert Timer unbrauchbare Ergebnisse.

6.11. Wie schnell läuft dies ab? Ein Beispiel aus der realen Welt!

Der größte gemeinsame Teiler (ggT, engl. GCD = Greatest Common Divisor) zweier Ganzzahlen ist die größte Ganzzahl, die beide Ganzzahlen ohne Rest teilt. Der ggT von 6 und 9 zum Beispiel ist 3. Die Zahlen 1 und 3 teilen beide die Zahlen 6 und 9, s. Tabelle 47. Die größte davon ist 3.

Tabelle 47. *Teilbarkeit von 6 und 9 durch Ganzzahlen.*

Zahl	Teilt 6	Teilt 9	Teilt beide
1	6	9	Ja
2	3	4 Rest 1	Nein
3	2	3	Ja
4	1 Rest 2	2 Rest 1	Nein
5	1 Rest 1	1 Rest 4	Nein
6	1	1 Rest 3	Nein
7	0 Rest 6	1 Rest 2	Nein
8	0 Rest 6	1 Rest 1	Nein
9	0 Rest 6	1	Nein

Dieses Beispiel geht weit zurück zum Jahr 300 v. Chr. ins antike Griechenland zu jemandem namens Euklid, der ein sehr kluger Mann war. Er schrieb viele Bücher, unter anderem *Data*, über Problemlösungen mittels geometrischer Analysis, *Über die Teilung der Figuren*, *Optika*, *Phainomena*, eine Schrift über sphärische Geometrie für Astronomen, *Die Elemente*, ein 13-bändiges Textbuch über Geometrie sowie eine Anzahl verlorengegangener Werke über höhere Geometrie. Seine Bedeutung für die Gesellschaft war immens. Einer seiner bekanntesten Beiträge ist ein außerordentlich effizienter Algorithmus zur Lösung der ggT-Aufgabe. Überspringen wir nun ein paar tausend Jahre zu Olivier Bietzer, der bemerkte, dass ich einen für die Praxis zu langsamen Algorithmus zur Lösung der ggT-Aufgabe verwendete. Olivier, der gewiss eine Menge über diese Dinge weiß, schrieb das Makro in Listing 123, das die ggT-Aufgabe über Euklids Algorithmus löst, und sandte es mir zu.

Listing 123. *Berechnung des ggT.*

```
'Autor: Olivier Bietzer
'e-mail: olivier.bietzer@laposte.net
'Verwendet wird Euklids Algorithmus. Ist sehr schnell!
'GCD (greatest common divisor) = englisch für ggT
Function GCD_1(ByVal x As Long, ByVal y As Long) As Long
  Dim pgcd As Long, test As Long
```

```
  ' Wir brauchen x >= y und positive Werte
  x = abs(x) : y = abs(y)
  If (x < y) Then
   test = x : x = y : y = test
  End If
  If y = 0 Then Exit Function

                  ' Euklid sagt: ....
  pgcd = y        ' Per Definition ist PGCD der kleinste
  test = x Mod y  ' Rest nach der Division.
  Do While (test) ' Solange test nicht 0 ist
   pgcd = test    ' pgcd ist der Rest
   x = y          ' x, y und der aktuelle pgcd werden permutiert
   y = pgcd
   test = x Mod y ' Nächster Test
  Loop
  GCD_1 = pgcd    ' pgcd ist der letzte Rest, der nicht 0 ist! Reine Magie!
End Function
```

Zur Geschwindigkeitssteigerung einer Problemlösung mit dem Computer gibt es im allgemeinen nichts Besseres als einen besseren Algorithmus. Der Algorithmus in Listing 123 läuft etwa 1000 mal schneller ab als die Routine, die ich vorher hatte. Wenn ein schnellerer Algorithmus nicht zur Verfügung steht, kann man nach anderen Wegen suchen, die Ausführung zu beschleunigen. (Manchmal ist es möglich, einen gänzlich neuen und verbesserten Algorithmus zu finden, aber das sagt sich so leicht! Wenn es einem gelingt, für eine allgemein bekannte Aufgabenstellung einen neuen, schnelleren Algorithmus zu entwickeln, hat man großes Karrierepotenzial als professioneller Mathematik- oder Informatikprofessor.) Der Code in Listing 123 ist schon ziemlich schlank. Es gibt nicht viel zu entfernen, aber ich dachte, ich könnte die Anzahl der Zuweisungen reduzieren, s. Listing 124.

Listing 124. *Berechnung des ggT (auf andere Art).*

```
Function GCD_2(ByVal x As Long, ByVal y As Long) As Long
  Dim pgcd As Long, test As Long

  ' Wir brauchen x >= y und positive Werte
  x = abs(x) : y = abs(y)
  If (x < y) Then
   test = x : x = y : y = test
  End If
  If y = 0 Then Exit Function

  Do While (y)      ' Solange y nicht 0 ist
   pgcd = y         ' pgcd ist der Rest
   y = x Mod pgcd   ' Nächster Test
   x = pgcd         ' x, y und der aktuelle pgcd werden permutiert
  Loop
  GCD_2 = pgcd      ' pgcd ist der letzte Rest, der nicht 0 ist! Reine Magie!
End Function
```

Die Frage ist nun, welche Funktion schneller ist. Wenn Sie mit einer Stoppuhr messen, wie schnell ich blinzeln kann, werden die Ergebnisse infolge von Messfehlern nicht sehr genau sein. Es ist viel leichter, mir zu sagen, ich solle in vier Sekunden so oft blinzeln, wie ich kann, oder zu messen, wie schnell ich 50 mal blinzeln kann. Der Code in Listing 125 macht das gleiche. In einer straffen Schleife wird die GCD-Funktion 5000 mal aufgerufen. Ich will wissen, welche Zeit diese 5000 Aufrufe benötigen, tatsächlich aber messe ich die Dauer von 5000 Schleifen-Läufen, von 10000 gerechneten

Zufallszahlen und von 5000 Aufrufen der GCD-Funktion. Zur Kompensation wird die Dauer von 5000 Schleifen und 10000 gerechneten Zufallszahlen gemessen.

Tipp Das Testprogramm braucht ein paar Sekunden. Haben Sie Geduld.

Listing 125. *Zeitmessung der beiden verschiedenen ggT-Funktionen.*

```
Sub TestGCD
  Dim nStartTicks As Long          'Beginn der Zeitmessung
  Dim nEndTicks As Long            'Ende der Zeitmessung
  Dim nLoopTicks As Long           'Ticks für die reine Schleife
  Dim nGCD_1_Ticks As Long         'Ticks für GCD_1
  Dim nGCD_2_Ticks As Long         'Ticks für GCD_2
  Dim nMinIts As Long              'Anzahl der Durchläufe
  Dim x&, y&, i&, n&               'Provisorische Zahlen als Long
  Dim s$                           'Der Ausgabestring

  nMinIts = 5000                   'Anzahl der Durchläufe wird gesetzt
  Randomize(2)                     'Fester Startwert für den Zufallszahlengenerator
  nStartTicks = GetSystemTicks()   'Startticks
  For i& = 1 To nMinIts            'Schleife für die Anzahl der Durchläufe
    x = 10000 * Rnd()              'Zufallszahl wird gerechnet
    y = 10000 * Rnd()              'Zufallszahl wird gerechnet
  Next
  nEndTicks = GetSystemTicks()
  nLoopTicks = nEndTicks - nStartTicks

  Randomize(2)                     'Fester Startwert für den Zufallszahlengenerator
  nStartTicks = GetSystemTicks()   'Startticks
  For i& = 1 To nMinIts            'Schleife für die Anzahl der Durchläufe
    x = 10000 * Rnd()              'Zufallszahl wird gerechnet
    y = 10000 * Rnd()              'Zufallszahl wird gerechnet
    GCD_1(x, y)                    'Die Arbeit, um die es uns geht
  Next
  nEndTicks = GetSystemTicks()
  nGCD_1_Ticks = nEndTicks - nStartTicks - nLoopTicks

  Randomize(2)                     'Fester Startwert für den Zufallszahlengenerator
  nStartTicks = GetSystemTicks()   'Startticks
  For i& = 1 To nMinIts            'Schleife für die Anzahl der Durchläufe
    x = 10000 * Rnd()              'Zufallszahl wird gerechnet
    y = 10000 * Rnd()              'Zufallszahl wird gerechnet
    GCD_2(x, y)                    'Die Arbeit, um die es uns geht
  Next
  nEndTicks = GetSystemTicks()
  nGCD_2_Ticks = nEndTicks - nStartTicks - nLoopTicks

  s = "Die Schleife braucht " & nLoopTicks & " Ticks für " & nMinIts &_
      " Durchläufe" & Chr$(10) &_
      "Der Aufruf von GCD_1 braucht " & nGCD_1_Ticks & " Ticks oder " &_
      Format(nMinIts * 100 / nGCD_1_Ticks, "#####00.00") &_
      " Durchläufe pro Sekunde" & Chr$(10) &_
      "Der Aufruf von GCD_2 braucht " & nGCD_2_Ticks & " Ticks oder " &_
      Format(nMinIts * 100 / nGCD_2_Ticks, "#####00.00") &_
      " Durchläufe pro Sekunde"

  MsgBox s, 0, "GCD-Vergleich"
End Sub
```

Ein Problem bei Routinen zur Zeitmessung ist die Entscheidung über die Anzahl der Durchläufe. Ich arbeite häufig an Rechnern mit unterschiedlichen Rechengeschwindigkeiten. Das Bild 54 zeigt die Ergebnisse des Makros in Listing 125, ausgeführt auf meinem (das heißt hier: des Übersetzers) Heimrechner. Das Makro führt eine bestimmte Anzahl an Durchläufen aus. Manchmal begrenze ich die Anzahl der Durchläufe durch ein Zeitintervall statt einer festen numerischen Größe. Das macht die Messung des Overhead komplizierter und soll dem Leser als interessantes, wenn auch nicht allzu schwieriges Problem überlassen sein.

Das Ergebnis im Bild 54 wird auf Ihrem Rechner ganz anders sein und sich bei jedem neuen Makroaufruf ändern. Die Verbesserungsrate beruht auf dem Durchschnitt von 1000 Makroaufrufen.

Bild 54. *Vergleich zweier Schleifendurchläufe: die Verbesserung beträgt etwa 6 Prozent.*

6.12. Große Zeitintervalle und spezielle Datumsermittlungen

Es ist einfach, die über einen längeren Zeitraum verstrichene Zeit zu ermitteln: man subtrahiert die Datumswerte. Zur Bestimmung präziser Daten und Intervalle können Sie die Einzelkomponenten kreativ nutzen. Sie haben zum Beispiel ein Datum und wollen wissen, welches der erste Tag des Monats war. Das ist einfach, denn der erste Tag jedes Monats ist der Tag 1. Mit den Funktionen Year und Month extrahieren Sie das Jahr und den Monat, und dann fügen Sie das Ganze mit DateSerial und dem Tag 1 wieder zusammen. Das Beispielsmakro ruft auch WeekDayText (s. Listing 115) auf.

Listing 126. *Der erste Tag des Monats.*

```
Function FirstDayOfMonth(d As Date) As Date
  FirstDayOfMonth() = DateSerial(Year(d), Month(d), 1)
End Function

Sub FirstDayOfThisMonth()
  Dim d As Date
  d = FirstDayOfMonth(Now())
  MsgBox "Der erste Tag dieses Monats (" & d & ") ist ein " & WeekDayText(d)
End Sub
```

Um den letzten Tag des Monats zu finden, benötigen Sie zuerst den ersten Tag des nächsten Monats, von dem Sie dann 1 subtrahieren. Wenn der aktuelle Monat der Dezember ist, setzen Sie den Monat auf Januar und erhöhen das Jahr um 1.

Listing 127. *Der letzte Tag des Monats.*

```
Function LastDayOfMonth(d As Date) As Date
  Dim nYear As Integer
  Dim nMonth As Integer
  nYear = Year(d)          'Aktuelles Jahr
  nMonth = Month(d) + 1    'Der nächste Monat
  If nMonth > 12 Then      'Wenn es Dezember war, hat der nächste Monat nun die Zahl 13
    nMonth = 1             'Die Monatszählung beginnt wieder bei 1,
    nYear = nYear + 1      'aber mit einem um 1 höheren Jahr
```

```
  End If
  LastDayOfMonth = CDate(DateSerial(nYear, nMonth, 1) - 1)
End Function

Sub LastDayOfThisMonth()
  Dim d As Date
  d = LastDayOfMonth(Now())
  MsgBox "Der letzte Tag dieses Monats (" & d & ") ist ein " & WeekDayText(d)
End Sub
```

Der erste Tag des Jahres ist leicht zu finden: es ist immer der 1. Januar. Mit der Funktion Year erhalten Sie das aktuelle Jahr. Dann setzen Sie Tag und Monat gleichermaßen auf 1. Den letzten Tag des Jahres zu finden, ist nur unwesentlich schwieriger. Sie suchen den ersten Tag des nächsten Jahres: Jahr plus 1 und Tag und Monat gleich 1. Dann subtrahieren Sie 1 vom ersten Tag des nächsten Jahres und erhalten den letzten Tag dieses Jahres.

```
d = Now
Print DateSerial(Year(d), 1, 1)                  '01.01.2011
Print CDate(DateSerial(Year(d) + 1, 1, 1) - 1)   '31.12.2011
```

Mit der Funktion WeekDay finden Sie den ersten und den letzten Tag der Woche. Subtrahieren Sie den Wochentag und addieren Sie 1 für den Sonntag. So finden Sie den Beginn der aktuellen Woche.

```
d = Date
Print CDate(CDbl(d) - WeekDay(d) + 1)   'Der 24.07.2011 ist ein Sonntag
Print CDate(CDbl(d) - WeekDay(d) + 7)   'Der 31.07.2011 ist ein Samstag
```

Auf ähnliche Weise können Sie Problemstellungen zu Datumsfragen lösen, wie die Bestimmung der Arbeitswoche, oder wie viele Tage es noch bis zu Ihrem Jahrestag dauert, oder das Alter einer Person in Jahren, Monaten und Tagen.

6.13. Fazit

Datumswerte sind in Basic im allgemeinen unkompliziert und leicht benutzbar. Aber man muss achtgeben bei einem Datum vor dem 15. Oktober 1582. Der Wechsel zwischen dem Gregorianischen und dem Julianischen Kalender kann unerwartete Schwierigkeiten machen. Achtung ist auch geboten, wenn der intern verwendete Double-Wert negativ wird. Das geschieht um den 30. Dezember 1899 herum. Dieses Kapitel behandelt auch Methoden zur Zeitmessung und zur Ermittlung spezieller Datumswerte.

7. String-Routinen

Dieses Kapitel behandelt die Subroutinen und Funktionen, die OOo zur Behandlung von Strings bereitstellt. Das sind Methoden zur Bearbeitung von Strings, zur Konvertierung anderer Datentypen zu Strings und zu speziellen Formatierungen.

Textdaten werden in Strings gespeichert, die eine Folge von vorzeichenlosen 16-Bit-Integer-Werten nach Unicode 2.0 bilden. Unicode, die weltweite Norm für Schriftzeichen, ist eine Liste binär kodierter Textelemente oder Schriftzeichen, die erstellt wurde, weil ASCII, die vorherige Norm, nur 256 verschiedene Zeichen bewältigen kann. Die ersten 128 Zeichen (von 0 bis 127 gezählt) entsprechen den Buchstaben und Symbolen auf einer normalen U.S.-amerikanischen Tastatur. Die nächsten 128 Zeichen (von 128 bis 255 gezählt) sind Sonderzeichen wie Akzente, weitere Buchstaben auf der Basis der lateinischen Schrift sowie einige Symbole. Die übrigen 65.280 Werte – von denen momentan nur etwa 34.000 im Gebrauch sind – decken eine breite Vielfalt der auf der Welt verwendeten Zeichen ab: Buchstaben, mathematische Symbole, Akzentzeichen (Diakritika) und technische Symbole.

OOo bietet eine große Zahl an Funktionen zur Bearbeitung von Strings, angefangen von der Konvertierung von Groß- zu Kleinbuchstaben (oder umgekehrt) bis zur Selektion von Teilstrings aus längeren Strings. In der Tabelle 48 sind die Funktionen aufgelistet, die in diesem Kapitel behandelt werden. Die Funktionen in der Tabelle 49 gehören sowohl zur Stringbearbeitung als auch zur Zahlen- oder Array-Bearbeitung. Sie werden in anderen Kapiteln vertieft.

Tabelle 48. *Im Kapitel 7 vorgestellte Funktionen zur Stringbearbeitung.*

Funktion	Beschreibung
Asc(String)	Gibt den ASCII-Wert des ersten Zeichens des Strings zurück. Unterstützt auch 16-Bit-Unicode-Werte.
Chr(n)	Konvertiert eine ASCII-Zahl zu einem Stringzeichen.
CStr(Objekt)	Konvertiert Standardtypen zu Strings.
Format(Objekt, Format)	Beliebige Formatierung. Nur für Strings.
Hex(n)	Gibt eine Zahl als String in hexadezimaler Darstellung zurück.
InStr(String1, String2) InStr(Start, String1, String2) InStr(Start, String1, String2, Modus)	Sucht String2 in String1. Rückgabewert ist entweder die Startposition des Treffers oder 0 bei keinem Treffer. Das optionale Argument Start bestimmt die Position, an dem die Suche starten soll. Das Argument Modus steuert den Vergleich mit oder ohne Beachtung der Groß- und Kleinschreibung: 1 (Standard) = mit, 0 = ohne.
InStrRev(String, Suche, Start, Modus)	Gibt die Position des ersten Vorkommens eines Strings in einem anderen zurück, von der rechten Seite des Strings ausgehend. Nur verfügbar mit „Option VBA-Support 1“. Start und Modus sind optional.
Join(s()) Join(s(), String)	Gibt einen String zurück, der die einzelnen Array-Elemente in Folge enthält, jeweils getrennt durch den optionalen Trennstring. Standardtrenner ist das Leerzeichen. Umkehrung der Funktion Split.
LCase(String)	Gibt ein Kopie des Strings in Kleinbuchstaben zurück.
Left(String, n)	Gibt die ersten n Zeichen des Strings zurück.
Len(String)	Gibt die Länge (Anzahl der Zeichen) des Strings zurück.
LSet String1 = String2	Ordnet einen String linksbündig im Platz aus, den ein anderer String beansprucht.
LTrim(String)	Gibt eine Kopie des Strings zurück, worin alle Leerzeichen am Anfang entfernt sind.
Mid(String, Start) Mid(String, Start, Länge) Mid(String, Start, Länge, String)	Gibt den Teilstring zurück, der an der Position Start beginnt. Wenn die Länge nicht angegeben ist, wird der gesamte Rest des Strings zurückgegeben. Wenn das vierte Argument (String) angegeben ist, wird der spezifizierte Teilstring damit ersetzt.

Funktion	Beschreibung
Oct(n)	Gibt eine Zahl als String in oktaler Darstellung zurück.
Replace(String, Suche, Ersetze, Start, Anzahl, Modus)	Durchsucht String nach Suche und ersetzt den Teilstring mit Ersetze. Start, Anzahl und Modus sind optional.
Right(String, n)	Gibt die letzten n Zeichen des Strings zurück.
RSet String1 = String2	Ordnet einen String rechtsbündig im Platz aus, den ein anderer String beansprucht.
RTrim(String)	Gibt eine Kopie des Strings zurück, worin alle Leerzeichen am Ende entfernt sind.
Space(n)	Gibt einen String zurück, der aus n Leerzeichen besteht.
Split(String) Split(String, String)	Splittet einen String in ein Array von Strings anhand des optionalen Trennstrings (Standard: Leerzeichen). Umkehrung der Funktion Join.
Str(n)	Konvertiert eine Zahl zu einem String (ohne Lokalisierung).
StrComp(s1, s2) StrComp(s1, s2, Modus)	Vergleicht zwei Strings und gibt -1, 0 oder 1 zurück, je nachdem, ob der erste String in alphabetischer Folge kleiner ist als der zweite oder ob er gleich oder ob er größer ist. Das optionale dritte Argument steuert den Vergleich mit oder ohne Beachtung der Groß- und Kleinschreibung: 1 (Standard) = mit, 0 = ohne.
StrConv(String, Modus [, Lokal])	Konvertiert einen String gemäß dem Argument Modus: 1=Großbuchstaben, 2=Kleinbuchstaben, 4=breit, 8=schmal, 16=Katakana, 32=Hiragana, 64=zu Unicode, 128=aus Unicode.
String(n, Char) String(n, ASCII)	Gibt einen String zurück, in dem ein Zeichen mehrfach wiederholt wird. Das erste Argument ist die Anzahl der Wiederholungen, der zweite ist das Zeichen oder sein ASCII-Wert.
StrReverse(String)	Kehrt einen String um. Braucht „Option VBASupport 1“ oder Compatibility-Mode(True).
Trim(String)	Gibt eine Kopie des Strings zurück, worin alle Leerzeichen am Anfang und Ende entfernt sind.
UCase(String)	Gibt ein Kopie des Strings in Großbuchstaben zurück.
Val(String)	Konvertiert einen String zu Double. Sehr tolerant bei nicht-numerischem Text.

Die Subroutinen und Funktionen, die Basic zur String-Behandlung bietet, sind alle in Tabelle 48 aufgeführt. Einige dieser Funktionen (s. Tabelle 49) werden in anderen Kapiteln genauer behandelt, weil sie direkt mit deren Inhalt zu tun haben. Im Abschnitt 7.8. Konvertierung anderer Daten zu Strings werde ich kurz darauf eingehen.

Tabelle 49. *In anderen Kapiteln vorgestellte Funktionen zur Stringbearbeitung.*

Funktion	Behandelt in	Beschreibung
Join(s()) Join(s(), String)	5. . Array-Routinen	Gibt einen String zurück, der die einzelnen Array-Elemente in Folge enthält, jeweils getrennt durch den optionalen Trennstring.
Split(String) Split(String, String)	5. . Array-Routinen	Splittet einen String in ein Array von Strings anhand des optionalen Trennstrings.
CStr(Objekt)	4. . Numerische Routinen	Konvertiert Standardtypen zu Strings.
Str(n)	4. . Numerische Routinen	Konvertiert eine Zahl zu einem String (ohne Lokalisierung).
Hex(n)	4. . Numerische Routinen	Gibt eine Zahl als String in hexadezimaler Darstellung zurück.
Oct(n)	4. . Numerische Routinen	Gibt eine Zahl als String in oktaler Darstellung zurück.

Funktion	Behandelt in	Beschreibung
Val(String)	4. . Numerische Routinen	Konvertiert einen String zu Double. Sehr tolerant bei nicht-numerischem Text.

7.1. ASCII- und Unicode-Werte

In den frühen Tagen des Computerzeitalters gab es unterschiedliche Gerätetypen zur Datenverarbeitung, und es gab keine verbindliche Methode zur Textdarstellung. Um das Problem zu entschärfen, stellte das American National Standards Institute (ANSI) den „American Standard Code for Information Interchange (ASCII)“ vor. Diese Norm wurde 1968 fertiggestellt. Sie enthielt eine Zuordnung von 128 Buchstaben, Ziffern, Satzzeichen und Steuerungszeichen zu den Zahlen 0 bis 127 (s. Tabelle 50). Dem Leser mit Computerverstand wird auffallen, dass dafür 7 Bits benötigt werden, also kein komplettes Byte.

Tabelle 50. *Die originalen 128 ASCII-Zeichen.*

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	“	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Tabelle 50 führt alle originalen 128 ASCII-Zeichen auf. Die obere Reihe und die linke Spalte dienen zur Identifizierung der hexadezimalen ASCII-Werte. Zum Beispiel hat der Großbuchstabe A den ASCII-Wert 41 hexadezimal, und Z hat den Wert 5A. Wenn in einem Kästchen mehr als ein Buchstabe steht, handelt es sich um ein Steuerzeichen (s. Tabelle 51). Einige dieser Steuerzeichen dienen der Datenübertragung, andere sind für Dateiformate, und manche finden sich sogar auf der Tastatur.

Tabelle 51. *Nicht druckbare ASCII-Zeichen.*

Hex	Dez	Symbol	Beschreibung
00	0	NUL	Nullzeichen, steht normalerweise für nichts.
01	1	SOH	Start of Heading = Beginn der Kopfzeile
02	2	STX	Start of Text = Beginn der Nachricht
03	3	ETX	End of Text = Ende der Nachricht
04	4	EOT	End of Transmission = Ende der Übertragung – nicht dasselbe wie ETB
05	5	ENQ	Enquiry = Anfrage
06	6	ACK	Acknowledge = Positive Bestätigung – Ich bin da oder Datenempfang erfolgreich
07	7	BEL	Bell = Tonsignal – Bringt Fernschreiber und viele Terminals zum Klingeln
08	8	BS	Backspace = Rückschritt – Cursor oder Druckkopf ein Zeichen zurück nach links
09	9	TAB	Horizontal Tab = Horizontaler Tabulator – Cursor oder Druckkopf nach rechts zum nächsten Tab-Stop in derselben Zeile
0A	10	LF	Line Feed = Zeilenvorschub oder Anfang neuer Zeile – Cursor oder Druckkopf zu neuer Zeile

Hex	Dez	Symbol	Beschreibung
0B	11	VT	Vertical Tab = Vertikaler Tabulator
0C	12	FF	Form Feed = Seitenvorschub – Cursor oder Druckkopf zum Anfang einer neuen Seite
0D	13	CR	Carriage return = Wagenrücklauf – Cursor oder Druckkopf zum linken Seitenrand
0E	14	SO	Shift out = Umschaltung – Schaltet das Ausgabegerät auf einen alternativen Zeichensatz um
0F	15	SI	Shift in = Rückschaltung – Schaltet das Ausgabegerät auf den Standardzeichensatz um
10	16	DLE	Data link escape = Datenverbindungs-Fluchtsymbol
11	17	DC1	Device control 1 = Gerätekontrollzeichen 1
12	18	DC2	Device control 2 = Gerätekontrollzeichen 2
13	19	DC3	Device control 3 = Gerätekontrollzeichen 3
14	20	DC4	Device control 4 = Gerätekontrollzeichen 4
15	21	NAK	Negative acknowledge = Negative Bestätigung
16	22	SYN	Synchronous idle = Synchronisierungssignal
17	23	ETB	End of transmission block = Ende des Übertragungsblockes – nicht dasselbe wie EOT
18	24	CAN	Cancel = Abbruch
19	25	EM	End of medium = Ende des Mediums
1A	26	SUB	Substitute = Ersatz
1B	27	ESC	Escape = Fluchtsymbol – Die Esc-Taste auf der Tastatur
1C	28	FS	File separator = Dateitrenner
1D	29	GS	Group separator = Gruppentrenner
1E	30	RS	Record separator = Datensatztrenner
1F	31	US	Unit separator = Einheitentrenner
7F	127	DEL	Delete = Zeichen löschen – Die Entf-Taste auf der Tastatur

In den meisten Rechnern ist die kleinste einfach zu speichernde und abzurufende Dateneinheit das Byte, das aus 8 Bits besteht. Die Zeichen in der Tabelle 50 benötigen nur 7 Bits. Um Platz zu sparen, wurde Extended ASCII eingeführt für die Zeichen von Nummer 128 bis 255. Auch wenn dieser Zeichensatz Sonderzeichen sowie mathematische und grafische Zeichen und fremdsprachige Buchstaben bereitstellte, so war das doch nicht ausreichend für den internationalen Gebrauch. Um 1986 herum begann Xerox damit, den Zeichensatz mit asiatischen Schriftzeichen zu erweitern. Diese Arbeit führte schließlich zu dem heutigen Unicode-Zeichensatz mit 16-Bit-Integer-Werten für 65.536 definierte Zeichen.

OOo speichert Zeichen als vorzeichenlose Unicode-Integer-Werte. Die Funktionen Asc und Chr konvertieren zwischen Integer- und Zeichenwert, zum Beispiel zwischen 65 und A. Mit der Funktion Asc ermitteln Sie den numerischen Unicode-Wert des ersten Zeichens in einem String. Der Rückgabewert ist ein 16-Bit-Integer, groß genug für Unicode-Werte. Nur das erste Zeichen des Strings wird herangezogen, der Rest wird ignoriert. Bei einem String mit der Länge null wird ein Laufzeitfehler erzeugt. Asc ist im Wesentlichen die Umkehrung der Funktion Chr$, mit der die Zahl in ein Zeichen konvertiert wird.

Obwohl es eine Unicode-Nummer ist, die von Asc zurückgegeben wird, bezeichnet man sie häufig einfach als den „ASCII-Wert“. Genau genommen ist das falsch, als Jargon aber weit verbreitet. Die Zahlen 0 bis 255 korrespondieren ja direkt mit den ASCII-Werten, und da die Programmierer sich

über Jahre an diese Terminologie gewöhnt haben, sind sie wohl nicht mehr davon abzubringen. Wenn Sie also in diesem Buch „ASCII-Wert“ lesen, denken Sie „Unicode-Wert“.

Tipp Die Funktion Chr wird häufig als Chr$ geschrieben. In Visual Basic gibt Chr$ einen String zurück und kann mit dem Argument null nichts anfangen, wohingegen Chr ein Variant zurückgibt, mit dem Null-Argumente akzeptiert und weiterverarbeitet werden. In StarBasic verhalten sich beide gleich, sie geben einen String zurück und erzeugen einen Laufzeitfehler bei einem Null-Argument.

Die Funktion Chr konvertiert einen 16-Bit-ASCII-Wert zu dem dazu gehörenden Zeichen. Das ist vor allem dann nützlich, wenn Sie ein Sonderzeichen in einen String einfügen wollen. Chr(10) ist zum Beispiel das Zeichen für eine neue Zeile. Die Funktion Chr ist die Umkehrung der Funktion Asc.

Listing 128. *Ausgabe eines Zeilenumbruchs.*

```
Sub ShowChrAsc
  Dim s$
  Print Chr$(65)              'A
  Print Asc("Andrew")         '65
  s = "1" & Chr$(10) & "2"    'Neue Zeile zwischen 1 und 2
  MsgBox s
End Sub
```

Tipp Nehmen Sie die Anweisung MsgBox, wenn Sie Strings ausgeben, die Chr$(10) oder Chr$(13) enthalten – mit beiden wird durch Basic ein Zeilenvorschub vorgenommen. Die Anweisung Print produziert bei jedem Zeilenumbruchzeichen einen neuen Ausgabedialog. Mit MsgBox hingegen werden die neuen Zeilen säuberlich in einem einzigen Dialog angezeigt.

Bei meinen Versuchen, die internen OOo-Funktionen zu entschlüsseln, stoße ich häufig auf Strings, die Zeichen enthalten, die nicht unmittelbar sichtbar sind, zum Beispiel Leerzeichen am Ende, Zeilenvorschübe und Zeilenrückläufe. Die Konvertierung des Strings in eine Folge von ASCII-Zeichen erleichtert das Erkennen des wahren Stringinhalts, s. Listing 129 und Bild 55.

Listing 129. *Konvertierung eines Strings zu ASCII-Werten.*

```
Sub ExampleStringToASCII
  Dim s As String
  s = "AB""""  """"BA"
  MsgBox s & Chr$(10) & StringToASCII(s), 0, "String zu ASCII"
End Sub

Function StringToASCII(sInput$) As String
  Dim s As String
  Dim i As Integer
  For i = 1 To Len(sInput$)
    s = s & CStr(Asc(Mid(sInput$, i, 1))) & " "
  Next
  StringToASCII = s
End Function
```

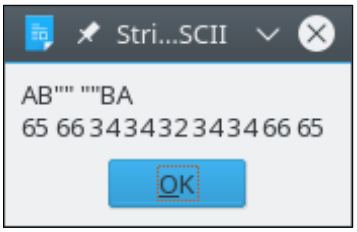


Bild 55. *Ein String mit den entsprechenden ASCII-Werten: A=65, B=66, "=34, usw.*

Bei mehr als einer Gelegenheit musste ich genau wissen, wie OOo die Daten in einem Textdokument speicherte. So etwas kommt dann vor, wenn man Zeilen- und Absatzwechsel in einer Weise bearbeiten möchte, die durch reguläre Ausdrücke nicht so einfach möglich ist. Die Subroutine in Listing 130 gibt den aktuell markierten Text als Folge von ASCII-Werten aus. Der wesentliche Punkt in diesem Kapitel besteht darin zu lernen, wie man die ASCII-Werte eines Textes ausgibt. Dabei sieht man auch die Zeichen, die zum Beispiel zwischen den Absätzen verwendet werden. Welche Methoden zur Selektion und Manipulation eines markierten Textes zweckmäßig sind, werden Sie später lernen. Markieren Sie einmal einen Textbereich und führen Sie das Makro aus.

Listing 130. *Gibt den markierten Text als Folge von ASCII-Werten aus.*

```
Sub SelectedTextAsASCII()
  Dim vSelections             'Mehrere unverbundene Selektionen
  Dim vSel                    'Eine einzelne Selektion
  Dim vCursor                 'OOo-Dokument-Cursor
  Dim i As Integer            'Indexvariable
  Dim s As String             'Temporärer Hilfsstring
  Dim bIsSelected As Boolean  'Ist überhaupt Text selektiert?
  bIsSelected = True          'Angenommen, dass ja

  'Die aktuelle Selektion im aktuellen Controller.
  'Wenn es aktuell keinen Controller gibt, wird NULL zurückgegeben.
  'ThisComponent referenziert das aktuelle Dokument.
  vSelections = ThisComponent.getCurrentSelection()
  If IsNull(vSelections) Or IsEmpty(vSelections) Then
    bIsSelected = False
  ElseIf vSelections.getCount() = 0  Then
    bIsSelected = False
  End If

  If Not bIsSelected Then            'Wenn nichts selektiert ist, wird das mitgeteilt
    Print "Es ist nichts selektiert" 'und die Subroutine beendet.
    Exit Sub
  End If

  'Die Selektionen werden von null aufwärts gezählt.
  'Ausgabe der ASCII-Werte einer jeden Selektion.
  For i = 0 To vSelections.getCount() - 1
    vSel = vSelections.getByIndex(i)
    vCursor = ThisComponent.Text.createTextCursorByRange(vSel)
    s = vCursor.getString()
    If Len(s) > 0 Then
      MsgBox StringToASCII(vCursor.getString()), 0, "ASCII der Selektion " & i
    ElseIf vSelections.getCount() = 1 Then
      Print "Es ist nichts selektiert"
    End If
  Next
End Sub
```

7.2. Standard-Stringfunktionen

Die grundlegenden Vergleichsoperatoren (=, <, <=, >, >= und <>) funktionieren mit Strings ebenso wie mit Zahlen. Die Vergleiche unterscheiden Groß- und Kleinschreibung. Das heißt, dass „a“ nicht als gleich mit „A“ gewertet wird. Man kann Strings auch mit der Funktion StrComp vergleichen. Die Standardeinstellung ist auch da der Vergleich mit Unterscheidung von Groß- und Kleinschreibung. Die Funktion StrComp gibt -1, 0 oder 1 zurück, je nachdem, ob das erste Argument (String) kleiner als, gleich oder größer als das zweite Argument (String) ist. Wenn Sie keine Unterscheidung von Groß- und Kleinschreibung wollen, setzen Sie das dritte Argument auf 0.

Tipp

Mit der Funktion StrComp(string1, string2, 0) vergleichen Sie Strings auf der Basis des aktuell eingestellten lokalen Gebietsschemas.

Beachten Sie, dass StrComp nur die ASCII-Werte der Zeichen vergleicht und nicht die Regeln zur lexikalischen Sortierung von Zeichen mit Diakritika in der Sprache des Gebietsschemas anwendet.

Normalerweise haben die klein geschriebenen Buchstaben einen höheren Wert als ihre groß geschriebenen Varianten. Es gibt aber Fälle, in denen es anders ist. Führen Sie sorgfältige Tests durch, falls dieser Umstand wichtig sein sollte.

StrComp scheint einen Bug im Modus Nichtbeachtung der Groß-/Kleinschreibung zu enthalten.

Der folgende Pseudocode gibt Ihnen eine Vorstellung davon, wie StrComp mit Unterscheidung von Groß- und Kleinschreibung arbeitet. Bei einem Verzicht auf eine solche Unterscheidung werden einfach beide Strings in Großbuchstaben konvertiert und danach verglichen.

```
Let s1 = string1
Let s2 = string2
Let min_len = minimum(Len(s1), Len(s2))
For i = 1 To min_len
  If Asc(Mid(s1, i, 1)) < Asc(Mid(s2, i, 1)) Then
    Set returnValue To -1
    Exit Function
  End If
  If Asc(Mid(s1, i, 1)) > Asc(Mid(s2, i, 1)) Then
    Set returnValue To 1
    Exit Function
  End If
Next
If Len(s1) < Len(s2) Then
  Set returnValue To -1
  Exit Function
End If
If Len(s1) > Len(s2) Then
  Set returnValue To 1
  Exit Function
End If
Set returnValue To 0
Exit Function
```

Der numerische Unicode-Wert des ersten Zeichens im ersten String wird mit dem numerischen Unicode-Wert des ersten Zeichens im zweiten String verglichen. Wenn das erste Zeichen numerisch kleiner als das zweite Zeichen ist, wird -1 zurückgegeben. Wenn das erste Zeichen numerisch größer als das zweite Zeichen ist, wird 1 zurückgegeben. Wenn die beiden Zeichen gleich sind, wird der Vergleich mit dem zweiten Zeichen beider Strings fortgesetzt. Wenn die entsprechenden numerischen Unicode-Werte aller Zeichen gleich sind und die Strings gleich lang sind, wird 0 zurückgegeben. Wenn die entsprechenden Zeichen zwar gleich sind, die Strings aber ungleich lang sind, wird der kürzere String als kleiner als der längere String bewertet.

Listing 131. *Beispiel für StrComp.*

```
Sub ExampleStrComp
  Print StrComp("A",  "AA")   '-1 weil "A"  < "AA"
  Print StrComp("AA", "AA")   ' 0 weil "AA" = "AA"
  Print StrComp("AA", "A")    ' 1 weil "AA" >  "A"

  Print StrComp("a", "A")     ' 1 weil  "a" > "A"
  Print StrComp("a", "A", 1)  ' 1 weil  "a" > "A"
  Print StrComp("a", "A", 0)  ' 0 weil  "a" = "A", wenn Groß- und Kleinschreibung
                              '                    ignoriert wird
End Sub
```

Die Funktionen UCase und LCase geben eine Kopie des Strings zurück, in der alle Zeichen in Groß- oder Kleinbuchstaben umgesetzt sind.

Listing 132. *Beispiel für UCase und LCase.*

```
Sub ExampleCase
  Dim s$
  S$ = "Köln am Rhein"
  Print LCase(s) REM Gibt "köln am rhein" zurück
  Print UCase(s) REM Gibt "KÖLN AM RHEIN" zurück
End Sub
```

Wenn zahlreiche Vergleiche notwendig sind, ist es manchmal schneller, LCase oder UCase zu verwenden, als jedes Mal einen Vergleich ohne Beachtung der Groß- und Kleinschreibung durchzuführen. Und manchmal ist es schlicht einfacher.

```
If LCase(Right(sFileName, 3)) = "odt" Then
```

StrConv(String, Modus, Locale_ID) konvertiert einen String flexibler als die einzelnen Methoden UCase und LCase. Die unterstützten Modi (s. Tabelle 52) sind Bitwerte, die addiert werden können. Zum Beispiel werden mit dem Modus 1+64=65 alle Zeichen zu Unicode-Großbuchstaben konvertiert. Das letzte Argument, Locale_ID als Integer, ist eine optionale lokale Kennung, die momentan (OOo 3.2.1) nicht unterstützt wird.

Tabelle 52. *Modi, die von StrConv unterstützt werden.*

Modus	Beschreibung
0	Keine Veränderung.
1	Konvertiert alle Zeichen zu Großbuchstaben.
2	Konvertiert alle Zeichen zu Kleinbuchstaben.
4	Konvertiert schmale (halbe Breite) Zeichen im String zu breiten (volle Breite) Zeichen.
8	Konvertiert breite (volle Breite) Zeichen im String zu schmalen (halbe Breite) Zeichen.
16	Konvertiert Hiragana-Zeichen im String zu Katakana-Zeichen.
32	Konvertiert Katakana-Zeichen im String zu Hiragana-Zeichen.
64	Konvertiert alle Zeichen zu Unicode.
128	Konvertiert alle Zeichen aus Unicode.

Die Funktionen LTrim, RTrim und Trim geben Kopien eines Strings zurück, in denen Leerzeichen am Anfang oder am Ende oder an beiden Seiten entfernt sind. Alle inneren Leerzeichen bleiben erhalten. Ich tue das regelmäßig mit Daten, die aus Dateien und Datenbanken stammen oder direkt vom Nutzer kommen. Der Originalstring bleibt unverändert. Manche Trimroutinen in anderen Programmiersprachen trennen alle Arten unsichtbarer Zeichen ab, Zeilenrücklauf, Zeilenvorschub und Tabulatoren. In Basic wird aber nur das Leerzeichen mit dem ASCII-Wert 32 abgetrennt.

Listing 133. *Beispiel für LTrim und RTrim.*

```
Sub ExampleTrim
  Dim s$
  s = "  Hallo  Welt   "
  Print "(" & LTrim(s) & ")"  '(Hallo  Welt   )
  Print "(" & RTrim(s) & ")"  '(  Hallo  Welt)
  Print "(" & Trim(s) & ")"   '(Hallo  Welt)
End Sub
```

Die Funktion Len gibt die Zahl der Zeichen in einem String zurück. Wenn das Argument kein String ist, wird es zu einem String konvertiert. Wahrscheinlich ist es sicherer, Argumente, die keine Strings sind, mit CStr zu Strings zu konvertieren, als sich auf den Automatismus zu verlassen. Zum Beispiel wird die automatische Konvertierung eines Typs wie Byte nicht das gewünschte Ergebnis bringen. Der Byte-Wert wird nämlich als ASCII-Wert behandelt und zu einem Einzelzeichen konvertiert. Mit der Funktion CStr hat man das Problem nicht.

Listing 134. *Beispiel für Len.*

```
Sub ExampleLen
  Print Len("")     '0
  Print Len("1")    '1
  Print Len("123") '3
  Print Len(12)     '2, die Zahl wird zu einem String konvertiert
End Sub
```

Mit der Funktion String erzeugt man einen String, in dem ein einzelnes Zeichen mehrfach wiederholt wird. Das erste Argument ist ein Integer-Wert für die Häufigkeit des Zeichens. Null ist ein durchaus gültiger Wert: es wird ein leerer String zurückgegeben. Das zweite Argument enthält das zu wiederholende Zeichen. Ebenso wie die Funktion Asc verwendet auch die Funktion String nur das erste Zeichen eines Strings und ignoriert den Rest. Wenn das zweite Argument eine Zahl ist, wird sie als ASCII-Wert behandelt und das entsprechende Unicode-Zeichen erzeugt.

Listing 135. *Beispiel für String.*

```
Sub ExampleString
  Print String(2, 65)     'AA  65 ist ASCII für A
  Print String(2, "AB")   'AA  Nur das erste Zeichen wird verwendet
  Print Asc(String(2))    '0   Bug: String erzeugt mit zwei ASCII-0-Zeichen
  Print Len(Space(4))     '4   Vier Leerzeichen
End Sub
```

Mit der Funktion InStr finden Sie heraus, wo (und ob überhaupt) ein String in einem anderen vorkommt. InStr kann vier Argumente akzeptieren. Das erste Argument ist optional, ein Integer-Wert, der angibt, an welcher Stelle im String die Suche beginnt. Wenn das Argument fehlt, wird 1 angenommen: das erste Zeichen des Strings. InStr durchsucht nun das zweite Argument danach, ob es das dritte Argument enthält. Mit dem vierten Argument kann die Unterscheidung von Groß- und Kleinschreibung geregelt werden. Im Standardverhalten (1) wird nicht unterschieden, mit der Angabe 0 wird unterschieden. Wenn Sie das vierte Argument verwenden, ist auch das erste Argument obligatorisch.

Tipp Die Funktion StrComp hat 0 zur Ignorierung der Groß- und Kleinschreibung und 1 – als Standard – zur Berücksichtigung. Die Funktion InStr hingegen hat 0 zur Berücksichtigung der Groß- und Kleinschreibung und 1 – als Standard – zur Ignorierung. Gemeinsam habe beide Funktionen nur, dass der Wert 1 den Standard darstellt.

Listing 136. *Beispiel für InStr.*

```
Sub ExampleInStr
  Print InStr("CBAABC", "abc")      '4 Standard, keine Unterscheidung von Groß/Klein
  Print InStr(1, "CBAABC", "b")     '2 Das erste Argument ist automatisch 1
  Print InStr(2, "CBAABC", "b")     '2 Start beim zweiten Zeichen
  Print InStr(3, "CBAABC", "b")     '5 Start beim dritten Zeichen
  Print InStr(1, "CBAABC", "b", 0) '0 Unterscheidung von Groß/Klein
  Print InStr(1, "CBAABC", "b", 1) '2 Keine Unterscheidung von Groß/Klein
  Print InStr(1, "CBAABC", "B", 0) '2 Unterscheidung von Groß/Klein
End Sub
```

InStrRev steht nur im VB-Kompatibilitätsmodus zur Verfügung. Im Gegensatz zur Funktion InStr durchsucht InStrRev den String von rechts nach links. Die Startposition ist das dritte Argument, im Gegensatz zu InStr, wo sie das erste Argument ist. Eine Startposition von -1 steht für das Zeichen ganz rechts, das letzte also. Ich hätte mir gewünscht, dass -2 für das zweitletzte stünde, aber es bewirkt einen Laufzeitfehler.

Listing 137. *Beispiel für InStrRev.*

```
Sub ExampleInStrRev
  CompatibilityMode(True)
  Print InStrRev("CBAABC", "ABC")         '4 Standard, Unterscheidung von Groß/Klein
  Print InStrRev("CBAABC", "abc")         '0 Standard, Unterscheidung von Groß/Klein
  Print InStrRev("CBAABC", "abc", -1, 1) '4 Erzwingt Unterscheidung von Groß/Klein
  Print InStrRev("CBAABC", "B", 1)        '0 Start mit dem ersten Zeichen
  Print InStrRev("CBAABC", "B", 2)        '2 Start mit dem zweiten Zeichen
  Print InStrRev("CBAABC", "B", -1)       '5 Start mit dem letzten Zeichen
  Print InStrRev("CBAABC", "B", 5)        '5 Start mit dem fünften Zeichen
  Print InStrRev("CBAABC", "B", 4)        '2 Start mit dem vierten Zeichen
  Print InStrRev("CBAABC", "b", -1, 0)    '0 Unterscheidung von Groß/Klein
  Print InStrRev("CBAABC", "b", -1, 1)    '5 Keine Unterscheidung von Groß/Klein
  Print InStrRev("CBAABC", "B", -1, 0)    '5 Unterscheidung von Groß/Klein
End Sub
```

In den Versionen vor OOo 2.0 ist der Rückgabewert von InStr ein Integer, mit dem Wertebereich von -32.768 bis 32.767. Ein String kann aber bis zu 65.635 Zeichen lang sein, was Probleme mit sich bringt, wenn InStr lange Strings durchsucht. Seit OOo 2.0 ist jedoch die Rückgabe vom Typ Long.

Listing 138. *Beispiel für InStr mit einem langen String.*

```
Sub ExampleLongStringInStr
  Dim s1 As String
  s1 = String(44000, "*") & "XX"    'Dieser String hat 44002 Zeichen.
  Print InStr(s1, "XX")             '44001
End Sub
```

7.3. Strings und Gebietsschema

Über das Menü **Extras | Optionen | Spracheinstellungen | Sprachen** erfahren Sie, welches Gebietsschema aktuell von OO verwendet wird. Das Gebietsschema des Übersetzers ist „Standard - Deutsch (Deutschland)“. Wenn Sie dieses Gebietsschema wechseln, müssen Sie OO beenden und neu starten, damit die geänderte Einstellung wirksam wird. Das folgende Listing zeigt Ihnen einen Unterschied zwischen den Gebietsschemata Deutsch und Türkisch.

Listing 139. *Manche Stringfunktionen nutzen das lokal eingestellte Gebietsschema.*

```
Sub LocaleStringTests
  ' Im Türkischen wird zwischen i (mit Punkt)
  ' und ı (ohne Punkt) unterschieden.
  ' Es handelt sich um verschiedene Laute.
  ' Derselbe Unterschied besteht in den Großbuchstaben: ı - I bzw. i - İ.
```

```
  Dim s$
  s = "Vergleich zwischen ı (ohne Punkt) und großem ı: " _
      & StrComp("ı", "I", 0) & Chr$(10)
  s = s & "Vergleich zwischen i (mit Punkt) und großem i:  " _
      & StrComp("i", "I", 0) & Chr$(10)
  s = s & "Kleines I = " & LCase("I") & Chr$(10)
  s = s & "Großes i = " & UCase("i")
  MsgBox s
End Sub
```

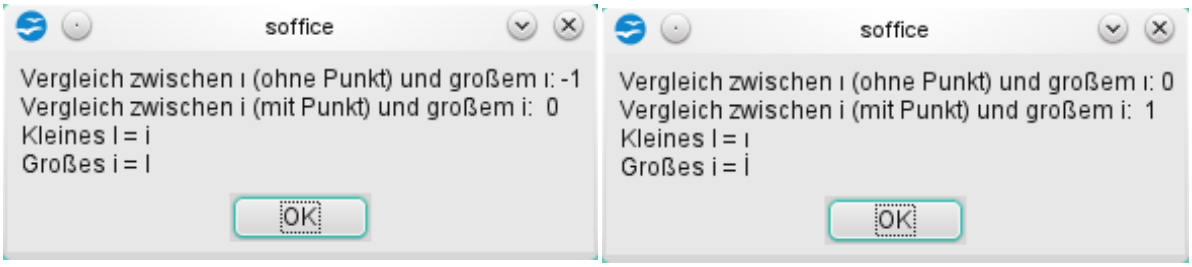


Bild 56. *Stringvergleich: Gebietsschema Deutsch links, Türkisch rechts*

7.4. Teilstrings

Mit der Funktion Left wird ein Teil des Stringanfangs entnommen. Das erste Argument ist der String, aus dem die Zeichen kommen sollen, und das zweite Argument bestimmt die Anzahl der zurückzugebenden Zeichen. Entsprechend gibt die Funktion Right Zeichen vom Ende des Strings zurück. Wenn die Anforderung die Länge 0 hat, wird ein leerer String zurückgegeben. Wenn die geforderte Länge zu groß ist, wird der gesamte String zurückgegeben.

```
Print  Left("12345", 2)  '12
Print  Left("12345", 8)  '12345
Print  Right("12345", 2) '45
```

Das Argument für die Länge ist bei den Funktionen Left und Right vom Typ Long und kann daher auch mit Strings umgehen, die bis zu 2 Milliarden Zeichen haben, s. Listing 140.

Listing 140. *Strings können bis zu 2 Milliarden Zeichen enthalten.*

```
Dim s1 As String
s1 = String(44002, "*")      'Dieser String hat 44002 Zeichen
Print Len(s1)                '44002
Print Len(Left(s1, 44000))   '44000
Print Len(Right(s1, 44000)) '44000
```

Mit der Funktion Mid werden beliebige Teilstrings entnommen wie auch Teilstrings in einem existierenden String ersetzt. Im allgemeinen gibt eine Stringfunktion einen neuen String zurück, ohne Änderung des existierenden Strings. Die Funktion Trim gibt zum Beispiel einen neuen String zurück, in dem die Leerzeichen am Anfang und am Ende entfernt sind, und lässt die Leerzeichen am Anfang und am Ende des ursprünglichen Strings bestehen. Die Funktion Mid kann jedoch genutzt werden, um den String zu modifizieren, anstatt einfach einen neuen zurückzugeben. In ihrer einfachsten Form ähnelt die Funktion Mid der Funktion Right. Das erste Argument ist ein String, das zweite Argument ist die Startposition. Das optionale dritte Argument bestimmt die Länge des zurückzugebenden Strings.

Listing 141. *Beispiele für Mid.*

```
Print Mid("123456", 3)         '3456
Print Mid("123456", 3, 2)      '34
s1 = String(44000, "*") & "XX"
```

```
Print Mid(s1, 44000)           '*XX    Kein Problem mit großen Argumenten
Print Len(Mid(s1, 2, 40000))   '40000  Kein Problem mit großen Argumenten
```

Die Funktion Mid kann genauso wie die Funktion Left wirken.

```
Left(s, n) = Mid(s, 1, n)
```

Die Funktion Mid akzeptiert ein viertes, optionales Argument, einen String, der im ersten Argument den spezifizierten Teilstring ersetzt. Anders gesagt, wenn es vier Argumente gibt, bestimmen die ersten drei Argumente einen Teilstring, und das vierte Argument ersetzt den Teilstring. Wenn das letzte Argument länger als der spezifizierte Teilstring ist, werden nur so viele Zeichen des letzten Arguments zum Ersetzen genommen, wie der Teilstring lang ist. Dadurch mag der String kürzer werden, aber nicht länger, wenn die Ersetzung am Anfang oder in der Mitte stattfindet.

Listing 142. *Beispiele für Mid mit Ersetzen.*

```
Sub ExampleMidReplace
  Dim s$
  s = "123456789"
  Mid(s, 3, 5, "")         'Ersetzt fünf Zeichen durch nichts
  Print s                  '1289

  s = "123456789"
  Mid(s, 3, 5, "XX")       'Ersetzt fünf Zeichen durch zwei
  Print s                  '12XX89

  s = "123456789"
  Mid(s, 3, 5, "ABCDEFG") 'Kann nicht mehr einfügen, als in der Mitte weggenommen wurde
  Print s                  '12ABCDE89

  s = "123456789"
  Mid(s, 7, 12, "ABCDEFG")'Auch am Ende kann man nicht mehr einfügen,
                           'als weggenommen wurde. In früheren Versionen konnte
                           'der String auf 123456ABCDEFG verlängert werden.
  Print s                  '123456ABC
End Sub
```

7.5. Ersetzen

Die Funktion ReplaceInString (s. Listing 143) bildet die Funktion Mid nach, mit zwei Ausnahmen: Sie fügt den gesamten neuen String ein, auch wenn er länger ist als der zu ersetzende Teilstring, und sie verändert den originalen String nicht.

Listing 143. *Eine generelle Methode zur Stringersetzung.*

```
REM Diese Funktion hat ähnlich wie Mid vier Argumente.
REM Diese Funktion verändert nicht den originalen String.
REM Diese Funktion ersetzt Text, der länger ist als n.
Function ReplaceInString(s$, i&, n&, sNew$) As String
  If i <= 1 Then
    'Der String wird am Anfang eingefügt.
    'Die Frage ist noch, wie viele Zeichen entfernt werden müssen.

    If n < 1 Then                    'Nichts wird entfernt
      ReplaceInString = sNew & s
    ElseIf n >= Len(s) Then          'Alles wird entfernt
      ReplaceInString = sNew
    Else                             'Von links wird ein Teil entfernt
      ReplaceInString = sNew & Right(s, Len(s) - n)
    End If
```

```
  ElseIf i + n > Len(s) Then
    'Eingefügt wird über das Ende hinaus.
    'Also wird der linke Teil extrahiert.
    'Mid funktioniert auch, wenn das Argument für die Länge größer
    'ist als der String. Der neue Text wird ans Ende angefügt.

    ReplaceInString = Mid(s, 1, i - 1) & sNew

  Else
    'Eingefügt wird irgendwo in der Stringmitte.
    'Zuerst wird der linke Teil des Strings entnommen.
    'Dann wird der neue Text, falls vorhanden, angefügt.
    'Schließlich wird der rechte Teil des Strings entnommen.

    ReplaceInString  = Mid(s, 1, i - 1) & sNew & Right(s, Len(s) - i - n + 1)
  End If
End Function
```

Aber halt! Es gibt die nicht dokumentierte Anweisung Replace(String, Suche, Ersetze, Start, Anzahl, Modus), die eine Kopie von String zurückgibt, in der alle Vorkommen von Suche durch Ersetze ersetzt sind. Die letzten drei Argumente sind optional.

Das Argument Start bestimmt, von wo ab der String zurückgegeben wird, nicht wo das Ersetzen beginnen soll. Mit dem Wert 1 werden alle Zeichen zurückgegeben. Mit dem Wert 3 werden die ersten beiden Zeichen ausgelassen.

Das Argument Anzahl bestimmt die maximale Anzahl der Ersetzungen. Mit dem Wert -1 werden alle Treffer ersetzt.

Das Argument Modus betrifft den Vergleich von Groß- und Kleinschreibung bei der Suche nach Textgleichheit. Mit dem Wert 1 wird nicht unterschieden, mit dem Wert 0 wird unterschieden.

7.6. Strings mit LSet und RSet ausrichten

Mit den Anweisungen LSet und RSet werden Strings links- oder rechtsbündig in dem Feld ausgerichtet, der von einem anderen String eingenommen würde. Das ist zum Beispiel für Spaltenköpfe nützlich, die durch Leerzeichen am Anfang oder am Ende rechts- oder linksbündig sein sollen. RSet und LSet haben dieselbe Syntax.

```
LSet string_1 = Ausdruck
RSet string_1 = Ausdruck
```

Der String auf der linken Seite kann beliebige Daten enthalten. Hauptsache, er hat die gewünschte Länge. Der Ausdruck auf der rechten Seite muss einen String ergeben. Dieser String wird in dem Feld ausgegeben, dessen Länge durch den String auf der linken Seite definiert wurde. Im Gegensatz zum Verhalten vieler Basic-Funktionen in wird der Ausdruck nicht automatisch zu einem String konvertiert.

Listing 144. *Beispiel für RSet.*

```
Dim s As String      'String-Variable als Behälter für die Ausgabe
s = String(10, "*") 'Der Ausgabebehälter hat eine Breite von 10 Zeichen
RSet s = CStr(1.23) 'Die Zahl wird nicht automatisch zu einem String konvertiert
Print "$" & s        '$      1,23
```

Das einzig Wichtige an dem String auf der linken Seite ist seine Länge – die Breite des Feldes, in dem der eigentliche String ausgegeben werden soll. Der einfachste Weg zu einem String mit einer spezifizierten Länge führt über die Funktion String. Das darin angegebene Zeichen ist unwesentlich, weil alle Füllzeichen in der Ausgabe aus Leerzeichen bestehen.

Listing 145. *Beispiel für LSet.*

```
Dim s As String     'String-Variable als Behälter für die Ausgabe
s = String(10, "X") 'Der Ausgabebehälter hat eine Breite von 10 Zeichen
LSet s = CStr(1.23) 'Die Zahl wird nicht automatisch zu einem String konvertiert
Print s & "%"       '1,23      %
```

Wenn der String auf der linken Seite kürzer ist als der String-Ausdruck auf der rechten, wird der Ausdruck auf die passende Größe beschnitten. Sowohl LSet als auch RSet schneiden die Zeichen vom Ende des Ausdrucks ab, um ihn auf die definierte Stringlänge zu bringen.

Listing 146. *LSet und RSet trunkieren.*

```
Dim s As String      'String-Variable als Behälter für die Ausgabe
s = String(4, "X")   'Der Ausgabebehälter hat eine Breite von 4 Zeichen
LSet s = CStr(21.23) 'Rechts trunkiert
Print "$" & s & "%"  '$21,2%
RSet s = CStr(21.23) 'Rechts trunkiert
Print "$" & s & "%"  '$21,2%
```

Der Code in Listing 147 zeigt das Verhalten der Anweisungen LSet und RSet. Das Ergebnis sehen Sie in Bild 57.

Listing 147. *Vollständiges Beispiel für LSet und RSet.*

```
Sub ExampleLSetAndRSet
  Dim s As String
  Dim sVar As String
  Dim sTmp As String

  sTmp = "12345"
  sVar = String(10, "*")
  LSet sVar = sTmp
  s = "LSet " & String(10, "*") & " = " & sTmp &_
      " == >" & sVar & "<" & Chr$(10)
  sVar = String(10, "*")
  RSet sVar = sTmp
  s = s & "RSet " & String(10, "*") & " = " & sTmp &_
          " == >" & sVar & "<" & Chr$(10) & Chr$(10)

  sVar = String(2, "*")
  LSet sVar = sTmp
  s = s & "LSet " & String(2, "*") & " = " & sTmp &_
          " == >" & sVar & "<" & Chr$(10)

  sVar = String(2, "*")
  RSet sVar = sTmp
  s = s & "RSet " & String(2, "*") & " = " & sTmp &_
          " == >" & sVar & "<" & Chr$(10)

  MsgBox s, 0, "RSet und LSet"
End Sub
```

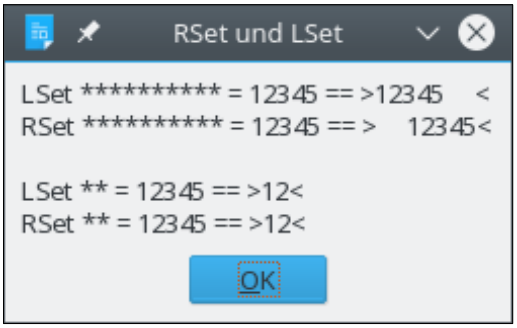


Bild 57. *RSet und LSet richten Strings bündig aus.*

Tipp	In Visual Basic erlaubt Ihnen LSet, Daten eines benutzerdefinierten Typs durch Daten eines anderen zu überladen, das heißt alle Bytes einer Datenstruktur auf einer anderen abzubilden, ungeachtet der zugrunde liegenden Struktur. In StarBasic funktioniert LSet nur mit Strings.

7.7. Beliebige Formatierung mit Format

Mit der Funktion Format können Sie eine Zahl zu einem String konvertieren und das Format mit einem optionalen Formatstring bestimmen. Und Sie können mehrere Formate in einem einzigen Formatstring einschließen, s. Tabelle 53. Der zurückgegebene formatierte String orientiert sich am aktuellen lokalen Gebietsschema. Das Gebietsschema setzen Sie über das Menü **Extras | Optionen | Spracheinstellungen | Sprachen**. Wenn der Formatstring fehlt, ist die Rückgabe von Format ähnlich wie bei der Funktion CStr.

Listing 148. *Einfache Format-Anweisungen.*

```
Sub ExampleFormatSimple
  Print Format(1223, "00.00")           '1223,00
  Print Format(1234.56789, "###00.00")  '1234,57
End Sub
```

Jeder einzelne Formatstring wird durch ein Semikolon angeschlossen. Der erste Formatstring gilt für positive Zahlen, der zweite für negative Zahlen und der dritte für null. Ist nur ein Formatstring angegeben, gilt er für alle Zahlen.

Listing 149. *Der Formatstring kann getrennte Formate enthalten: für Zahlen, die positiv, negativ oder null sind.*

```
Sub ExampleFormat3
  Dim s As String
  s = "P 00000.000;N ####.00;Z 0.0"
  Print Format(-12.3, s)   'N 12,30
  Print Format(0, s)       'Z 0,0
  Print Format(12.3, s)    'P 000012,300
End Sub
```

Tabelle 53. *Formatkennungen für Zahlen.*

Kennung	Beschreibung
0	Wenn in der Zahl an der Position der 0 im Formatstring eine Ziffer steht, wird die Ziffer angezeigt, ansonsten erscheint 0. Das bedeutet, dass Nullen am Anfang und am Ende angezeigt werden. Führende Nullen werden nicht abgeschnitten, niedrigere Dezimalstellen werden gerundet.
#	Wie bei 0, außer dass Nullen am Anfang und am Ende abgeschnitten werden.
.	Der Dezimaltrenner bestimmt die Anzahl der Dezimalstellen links und rechts des Trenners. Auch wenn im Formatstring ungeachtet des aktuellen Gebietsschemas immer ein Punkt stehen muss, wird für die Ausgabe der für das Gebietsschema korrekte Dezimaltrenner verwendet.

Kennung	Beschreibung
%	Multipliziert die Zahl mit 100 und fügt das Prozentzeichen (%) an der entsprechenden Stelle ein.
E- E+ e- e+	Wenn im Formatstring wenigstens ein Ziffernplatzhalter (0 oder #) rechts von der Kennung steht, wird die Zahl in der wissenschaftlichen Notation formatiert. Zwischen Zahl und Exponent wird E oder e eingefügt. Die Anzahl der Ziffernplatzhalter rechts von der Kennung bestimmt die Anzahl der Ziffern im Exponenten. Wenn der Exponent negativ ist, wird direkt davor ein Minuszeichen (-) gesetzt. Wenn der Exponent positiv ist, wird ein Pluszeichen (+) nur im Falle des Symbols E+ oder e+ gesetzt.
,	Das Komma ist ein Platzhalter für den Tausendertrenner. Er trennt die Tausender von den Hundertern in einer Zahl mit mindestens vier Ziffern. Der Tausendertrenner wird ausgegeben, wenn im Formatstring der Trenner von Ziffernplatzhaltern (0 oder #) eingefasst ist.
- + $ ( ) Leerzeichen	In den Formatstring direkt eingefügte Pluszeichen (+), Minuszeichen (-), Dollarzeichen ($), Leerzeichen oder Klammern werden als literale Zeichen ausgegeben.
\	Der umgekehrte Schrägstrich (Backslash) kennzeichnet das nächste Zeichen zur direkten Ausgabe. Mit anderen Worten, er verhindert, dass das nächste Zeichen als Sonderzeichen verstanden wird. Der Backslash selbst wird nicht ausgegeben, außer er wird im Formatstring doppelt (\\) aufgeführt. Zeichen, denen im Formatstring ein Backslash vorausgehen muss, damit sie ausgegeben werden, sind Zeichen zur Formatierung von Datum und Uhrzeit (a, c, d, h, m, n, p, q, s, t, w, y, /, :), von Zahlen (#, 0, %, E, e, Komma, Punkt) und von Strings (<, >). Man kann stattdessen auch Zeichen in doppelte Anführungszeichen setzen.
General Number	Zahlen werden so ausgegeben, wie sie eingegeben wurden.
Currency	Ein Währungssymbol wird je nach Gebietsschema vor oder hinter die Zahl gesetzt. Negative Zahlen stehen in eckigen Klammern.
Fixed	Wenigstens eine Ziffer steht vor dem Dezimaltrenner, zwei Dezimalstellen stehen dahinter.
Percent	Multipliziert die Zahl mit 100 und fügt ein Prozentzeichen (%) an.
Standard	Gibt Zahlen mit dem lokalen Tausendertrenner und zwei Dezimalstellen aus.
Scientific	Gibt Zahlen in wissenschaftlicher Notation mit zwei Dezimalstellen aus.

Die Funktion Format ist in den letzten Jahren gewaltig verbessert worden, und die meisten, aber nicht alle Bugs sind behoben. In der Tabelle 53 sind die Formatkennungen für Zahlen aufgelistet.

Listing 150. *Beispiele für Kennungen für numerische Formate.*

```
Sub ExampleFormat
  MsgBox Format(6328.2, "##,##0.00")              REM 6.328,20
  MsgBox Format(123456789.5555, "##,##0.00")      REM 123.456.789,56
  MsgBox Format(0.555, ".##")                     REM ,56
  MsgBox Format(123.555, "#.##")                  REM 123,56
  MsgBox Format(123.555, ".##")                   REM 123,56
  MsgBox Format(0.555, "0.##")                    REM 0,56
  MsgBox Format(0.1255555, "%#.##")               REM %12,56
  MsgBox Format(123.45678, "##E-####")            REM 12E1            LO: Fehler: 1E2
  MsgBox Format(.0012345678, "0.0E-####")         REM 1,2E-003
  MsgBox Format(123.45678, "#.e-###")             REM 1,e002          LO: Fehler: 1e002
  MsgBox Format(.0012345678, "#.e-###")           REM 1,e-003         LO: Fehler: 1e-003
  MsgBox Format(123.456789, "#.## \u\n\d ###")    REM 123,46 und 679
  MsgBox Format(8123.456789, "General Number")    REM 8123,456789
  MsgBox Format(8123.456789, "Fixed")             REM 8123,46
  MsgBox Format(8123.456789, "Currency")          REM 8.123,46 €      LO: Fehler: 8.123
  MsgBox Format(8123.456789, "Standard")          REM 8.123,46
  MsgBox Format(8123.456789, "Scientific")        REM 8,12E+03
  MsgBox Format(0.00123456789, "Scientific")      REM 1,23E-03
  MsgBox Format(0.00123456789, "Percent")         REM 0,12%
End Sub
```

Die Formatkennungen zur Datums- und Uhrzeitformatierung finden Sie in Tabelle 54. Aus mir unerfindlichen Gründen sind sie in den Hilfetexten nicht enthalten.

Tabelle 54. *Formatkennungen für Datum und Uhrzeit.*

Kennung	Beschreibung
q	Das Quartal des Jahres als Q1 bis Q4
qq	Das Quartal des Jahres als 1. Quartal bis 4. Quartal
y	Der Tag des Jahres (1 bis 365).
yy	Das Jahr zweistellig.
yyyy	Das Jahr vierstellig.
m	Der Monat als Zahl ohne führende Null.
mm	Der Monat als Zahl zweistellig, falls nötig mit führender Null.
mmm	Der Monatsname auf die ersten drei Buchstaben gekürzt (Jan bis Dez).
mmmm	Der vollständige Monatsname.
mmmmm	Der erste Buchstabe des Monatsnamens
d	Der Monatstag ohne führende Null.
dd	Der Monatstag zweistellig, falls nötig mit führender Null.
ddd	Der Wochentagsname auf die ersten zwei Buchstaben gekürzt (So bis Sa).
dddd	Der Wochentagsname (Sonntag bis Samstag). Wie nnn.
ddddd	Das komplette Datum in Kurzform.
dddddd	Das komplette Datum in Langform.
w	Der Wochentag als Zahl (1 bis 7).
ww	Die Woche des Jahres (1 bis 52).
h	Die Stunde ohne führende Null.
hh	Die Stunde zweistellig, falls nötig mit führender Null.
n	Die Minute ohne führende Null.
nn	Die Minute zweistellig, falls nötig mit führender Null.
nnn	Der Wochentagsname (Sonntag bis Samstag). Wie dddd.
s	Die Sekunde ohne führende Null.
ss	Die Sekunde zweistellig, falls nötig mit führender Null.
ttttt	Die komplette Uhrzeit in Langform.
c	Das komplette Datum und die komplette Uhrzeit.
/	Datumstrenner. Ausgabe gemäß dem lokalen Gebietsschema.
:	Uhrzeittrenner. Ausgabe gemäß dem lokalen Gebietsschema.

Mir ist zumindest ein Bug aufgefallen. Im Listing 151 wird mit dem Formatstring „d/mmmm/yyyy h:nn:ss“ in der letzten Zeile das „nn“ nicht korrekt expandiert, s. Bild 58. Ersetzen Sie es durch „mm“, dann wird die Ausgabe wunschgemäß sein.

Listing 151. *Beispiele für die Formatkennungen für Datum und Uhrzeit.*

```
Sub FormatDateTimeStrings
  Dim i%
  Dim d As Date
  d = Now()
```

```
    Dim s$
    Dim formats
    formats = Array("q", "qq", "y", "yy", "yyyy", _
                    "m", "mm", "mmm", "mmmm", "mmmmm", _
                    "d", "dd", "ddd", "dddd", "ddddd", "dddddd", _
                    "w", "ww", "h", "hh", "n", "nn", "nnn", "s", "ss", _
                    "ttttt", "c", "d/mmmm/yyyy h:nn:ss")
    For i = LBound(formats) To UBound(formats)
      s = s & formats(i) & " => " & Format(d, formats(i)) & Chr$(10)
    Next
    MsgBox s
End Sub
```

***Bild 58**. Formatkennungen für Datum und Uhrzeit.*

Formatkennungen für Strings finden Sie in Tabelle 55.

***Tabelle 55.** Formatkennungen für Strings.*

Kennung	Beschreibung
<	String in Kleinbuchstaben.
>	String in Großbuchstaben.

Andere Kennungen für das Stringformat wurden immer dokumentiert, aber nie eingesetzt, s. Tabelle 56. Ich habe sie mit aufgeführt, weil die Kennungen aus Tabelle 54 und Tabelle 55 immer dokumentiert, aber nicht eingesetzt waren. Nun sind sie eingesetzt, aber nicht dokumentiert.

Tabelle 56. *Weitere Formatkennungen für Strings.*

Kennung	Beschreibung
@	Platzhalter. Wenn der String an dieser Stelle leer ist, wird ein Leerzeichen ausgegeben. „@@@“ wird zum Beispiel mit einem Leerstring zu „( )“ formatiert.
&	Platzhalter. Wenn der String an dieser Stelle leer ist, wird nichts ausgegeben. „(&&&)“ wird zum Beispiel mit einem Leerstring zu „()“ formatiert.
!	Normalerweise werden Platzhalter von rechts nach links gefüllt. Das ! bewirkt, dass die Platzhalter von links nach rechts gefüllt werden.

Im Augenblick (OOo 3.2.1) sind für Strings nur die Formatkennungen zur Ausgabe in Groß- oder Kleinbuchstaben eingesetzt.

Listing 152. *Formatkennungen für Strings.*

```
Sub FormatStrings
  Dim i%
  Dim s$
  Dim formats
  formats = Array("<", ">",  _
                  "@@", "(@@@)", "[@@@@]", _
                  "&&", "(&&&)", "[&&&&]", _
                  )
  For i = LBound(formats) To UBound(formats)
    s = s & formats(i) & " => (" & Format("On", formats(i)) & ")" & Chr$(10)
  Next
  MsgBox s
End Sub
```

7.8. Konvertierung anderer Daten zu Strings

Basic bietet Funktionen zur Konvertierung anderer Datentypen zu Strings. Obwohl die Funktion Format die vielseitigste Methode zur Konvertierung einer Zahl zu einem String darstellt, ist häufig eine so weitgehende Kontrolle nicht erforderlich. Die Funktion Str konvertiert eine Zahl ohne Lokalisierung zu einem String, und die Funktion Val konvertiert sie zurück zu einer Zahl.

Die Funktionen Hex und Oct konvertieren eine Ganzzahl vom Typ Long zu ihrer Hexadezimal- oder Oktalnotation. Es wird nicht „&H“ und „&O“ davor gesetzt. Um diese Strings zurück zu numerischen Werten zu konvertieren, müssen Sie sie explizit mit den entsprechenden Kennungen einleiten.

Die Funktion CStr ist in der Lage, nahezu jeden Datentyp gemäß dem lokalen Gebietsschema zu einem String zu konvertieren, s. Tabelle 57. Die Funktion Str ist auf Zahlen begrenzt und berücksichtigt nicht das lokale Gebietsschema.

Tabelle 57. *Mit CStr konvertierte Datentypen.*

Typ	Konvertiert zu einem String
Boolean	"True" oder "False".
Date	Formatiertes Datum wie 08.06.2010.
Null, nicht initialisiertes Objekt	Laufzeitfehler.
Empty, nicht initialisierter Variant	"" = String mit der Länge null.
Jeder numerische Wert	Zahl als String.

Listing 153. *CStr mit einigen Datentypen.*

```
Sub ExampleCStr
  On Error Goto Handler
  Dim b As Boolean
  Dim o As Object
  Dim v As Variant  ' Ist leer
  Dim d As Double : d = Pi()

  Print "Boolean (" & CStr(b) & ")"
  Print "Date (" & CStr(Now) & ")"
  Print "Leerer Variant (" & CStr(v) & ")"
  Print "Double (" & CStr(d) & ")"
  Print "Null-Object (" & CStr(o) & ")"
  Exit Sub
Handler:
  Print "Aufgetretener Fehler: " & Error
  Resume Next
End Sub
```

Die Funktion CStr ist dann notwendig, wenn Sie einen Wert explizit zu einem String konvertieren müssen, weil die Standardkonvertierung falsche Ergebnisse liefert. Zum Beispiel bestimmt der erste Operand des Additionsoperators, ob das Ergebnis ein String oder eine Zahl ist. Das ist übrigens auch ein Grund gegen die Verwendung des Additionsoperators (+) zur Stringverkettung. Dafür gibt es schließlich den eigens geschaffenen Operator &.

```
Sub CstrConvertNumbers
  Print 3 + "4"        '7
  Print CStr(3) + "4"  '34
End Sub
```

Die Funktion Join verkettet alle Elemente eines eindimensionalen Arrays zu einem einzelnen String. Ohne Angabe eines Trennstrings werden die Elemente durch ein Leerzeichen getrennt.

```
Sub JoinArray
  Print Join(Array(3, 4, 5))      '3 4 5
  Print Join(Array(3, 4, 5), "X") '3X4X5
End Sub
```

Die Funktion Split zergliedert einen String in Einzelteile jeweils an einem optionalen Trennstring. Das ist im Prinzip das Gegenteil der Funktion Join und bietet den schnellsten Weg, einen String anhand eines Trennstrings in eine Reihe von Teilstrings aufzutrennen.

```
Split("3 4 5")      'Gibt das Array (3, 4, 5) zurück
Split("3X4X5", "X") 'Gibt das Array (3, 4, 5) zurück
```

7.9. Weitergehende Methode zur Textsuche

Die üblichen Methoden zur Textsuche sind StrComp (Listing 131), InStr (Listing 136) und InStrRev (Listing 137). Das sind Funktionen, die Basic zur Verfügung stellt. In allen Fällen können Sie nur rein literal suchen, das heißt, dass der Vergleichsstring eine feste Zeichenfolge ist. Wenn Sie aber mit einem regulären Ausdruck oder nach Ähnlichkeit suchen wollen, müssen Sie auf einen UNO-Service ausweichen. Reguläre Ausdrücke kennen Sie aus dem Writer- oder Calc-Dialog **Suchen & Ersetzen** im Menü **Bearbeiten**. Was ein UNO-Service ist, lernen Sie im Kapitel 10. . Universal Network Objects (UNO) kennen. Mit dem Service TextSearch können Sie die Textsuche vielfältig gestalten. Es stehen Ihnen die Modi ABSOLUTE (= literale Suche), REGEXP (= regulärer Ausdruck) oder APPROXIMATE (= Ähnlichkeitssuche) zur Verfügung. Sie können vorwärts und rückwärts suchen und auch Textteile ersetzen. Ein Beispiel für die Verwendung von TextSearch finden Sie im Listing 207. Nutzung des Service TextSearch.

7.10. Fazit

Es macht sich bezahlt, wenn man die von Basic unterstützten Funktionen kennt. Bevor ich auf die Funktion Split stieß, verbrachte ich viel Zeit damit, ein Makro zu schreiben, das einen String in Einzelteile zerlegte. Ich schrieb mein Makro dann mit der Funktion Split neu, und das Makro war deutlich schneller.

Basic hat ein großes Potenzial zur Textformatierung. Unter anderem ist durch den Unicode-Zeichensatz die Verarbeitung beinahe jeder Sprache der Welt möglich. Hinzu kommt eine Anzahl an leistungsfähigen Funktionen zum Zusammenfügen, Trennen und Formatieren von Textstrings.

8. Dateiroutinen

Dieses Kapitel stellt die Subroutinen und Funktionen vor, die Basic für Dateien und Verzeichnisse bereitstellt. Nachdem Sie dieses Kapitel gelesen haben, können Sie Dateien und Verzeichnisse erstellen, löschen, umbenennen und verschieben. Sie lernen Methoden zur Inspektion von Dateien, geöffnet oder geschlossen, und Verzeichnissen kennen. Sie werden auch erfahren, welche Eigenheiten und Bugs beim Lesen und Schreiben von Dateien zu beachten sind und welche Unterschiede zwischen den Betriebssystemen bestehen.

Basic bietet Funktionen, mit denen Sie auf das Dateisystem zugreifen können (s. Tabelle 58), für einfache und komplexe Anforderungen. Sie können Verzeichnisse anlegen und löschen oder auch Dateien öffnen und durchsuchen. Ich werde in der gebotenen Breite auf Verzeichnisse eingehen, auf Dateiattribute und auf die verschiedenen Dateitypen. Ich werde im Einzelnen zeigen, wie Dateien organisiert und bearbeitet werden, wie die verschiedenen Dateitypen strukturiert sind und welche Funktionen es zur Datenein- und -ausgabe für diese verschiedenen Dateitypen gibt. Es ist eine wahre Freude, wie einfach es ist, Makros zum Verschieben und Umbenennen von Dateien zu schreiben. Andererseits fühlen sich die Funktionen zur Bearbeitung von Binärdateien und zum wahlfreien Zugriff rau und kantig an.

Tabelle 58. *Basic-Dateifunktionen.*

Funktion	Beschreibung
ChDir(Pfad)	Wechsel des aktuellen Verzeichnisses oder Laufwerks. Veraltet. Nicht mehr verwenden!
ChDrive(Pfad)	Wechsel des aktuellen Laufwerks. Veraltet. Nicht mehr verwenden!
Close #n	Schließt eine oder mehrere vorher geöffnete Dateien. Mehrere Dateinummern werden durch Komma getrennt.
ConvertFromURL(String)	Konvertiert einen als URL angegebenen Pfad zu einem systemspezifischen Pfad.
ConvertToURL(String)	Konvertiert einen systemspezifischen Pfad zu einem URL.
CurDir CurDir(Laufwerk)	Gibt das aktuelle Arbeitsverzeichnis als Systempfad zurück. Mit der optionalen Laufwerksangabe wird das aktuelle Arbeitsverzeichnis des Laufwerks zurückgegeben.
Dir(Pfad) Dir(Pfad, Attribute)	Gibt eine Dateiliste aufgrund des angegebenen Pfads zurück. Der Pfad kann eine Dateiauswahl enthalten – zum Beispiel "/home/andy/*.txt". Optionale Attribute bestimmen, ob eine Datei- oder Verzeichnisliste zurückgegeben wird.
EOF(n)	Gibt True zurück, wenn das Ende der Datei mit der Nummer n erreicht ist.
FileAttr(n, 1)	Gibt den Modus zurück, in dem die Datei mit der Nummer n geöffnet wurde. Das zweite Argument bestimmt, ob der Dateizugriff oder der Modus des Betriebssystems gemeint ist. Derzeit wird aber nur der Dateizugriff unterstützt.
FileCopy(Quelle, Ziel)	Kopiert eine Datei von der „Quelle“ zum „Ziel“.
FileDateTime(Pfad)	Gibt Datum und Uhrzeit der Datei als String zurück.
FileExists(Pfad)	Gibt True zurück, wenn die Datei oder das Verzeichnis existiert.
FileLen(Pfad)	Gibt die Größe der Datei als Long zurück.
FreeFile()	Gibt die nächste verfügbare Dateinummer zum Gebrauch zurück.
Get #n, Variable Get #n, Pos, Variable	Liest einen Datensatz aus einer strukturierten Datei oder eine Folge von Bytes aus einer Binärdatei in eine Variable. Wenn das Positionsargument fehlt, werden die Daten von der aktuellen Position in der Datei gelesen. Für Dateien, die im Binärmodus geöffnet wurden, ist es die Position der Bytezählung.
GetAttr(Pfad)	Gibt die Attribute des Dateityps als Bitmuster zurück.
GetPathSeparator()	Gibt den systemspezifischen Pfadtrenner zurück.
Input #n, Variable	Liest numerische oder String-Datensätze sequenziell aus einer geöffneten Datei in eine oder mehrere Variablen. Zeilenrücklauf (Asc=13), Zeilenvorschub (Asc=10) und Komma wirken als Trennzeichen. Input kann keine Kommas oder Anführungszeichen lesen, weil sie als Datensatztrenner gelten. Nötigenfalls nehmen Sie die Anweisung Line Input.

Funktion	Beschreibung
Kill(Pfad)	Löscht eine Datei vom Datenträger.
Line Input #n, Variable	Liest der Reihe nach zeilenweise Strings in eine Variable, jeweils bis zum ersten Wagenrücklauf (Asc=13) oder Zeilenvorschub (Asc=10). Die Zeilenendezeichen werden nicht mit zurückgegeben.
Loc(n)	Gibt die aktuelle Position in einer geöffneten Datei zurück.
LOF(n)	Gibt die Größe einer geöffneten Datei in Bytes zurück.
MkDir(Pfad)	Erstellt das Verzeichnis.
Name Quelle As Ziel	Benennt eine Datei oder ein Verzeichnis um.
Open Pfad For Modus As #n	Öffnet einen Datenkanal (Datei): Modus Input = Lesen, Modus Output = Schreiben ...
Put #n, Variable Put #n, Pos, Variable	Schreibt einen Datensatz in eine strukturierte Datei oder eine Folge von Bytes in eine binäre Datei.
Reset	Schließt alle geöffneten Dateien und gleicht alle Daten aus dem Arbeitsspeicher mit denen auf dem externen Speicher ab.
RmDir(Pfad)	Löscht ein Verzeichnis.
Seek #n, Pos	Setzt die Position für den nächsten Schreib- oder Lesezugriff in einer Datei.
SetAttr(Pfad, Attribute)	Setzt die Dateiattribute.
Write #n, String	Schreibt Daten in eine Datei.

8.1. Der Dateipfad in URL-Notation

Viele der Funktionen in Tabelle 58 benötigen eine Datei oder einen Dateipfad. Name oder Pfad werden sowohl in der systemspezifischen Form als auch als URL (Uniform Resource Locator) akzeptiert. Das ist genau das Format, das auch Ihr Webbrowser nutzt. Tabelle 59 zeigt ein paar Beispiele.

Tabelle 59. *Beispiele für URLs.*

System	Systempfad	URL-Pfad
Windows	c:\Temp\help.txt	file:///c:/Temp/help.txt
Windows	c:\My Documents	file:///c:/My%20Documents
Unix	/home/andy/Temp/help.txt	file:///home/andy/Temp/help.txt
Unix	/home/andy/My Documents	file:///home/andy/My%20Documents

Tipp Die Anweisung „Shell("C:\Prog Files\calc.exe",2)“ scheitert, weil ein Leerzeichen im Pfad ist. Die Anweisung Shell übergibt den String an den Kommandozeileninterpreter (engl. Shell), der den Teil des Pfads vor dem Leerzeichen als auszuführendes Programm betrachtet. Mit der URL-Notation tritt dieses Problem nicht auf.

Ein Vorteil der URL-Notation ist, dass Sonderzeichen kodiert werden. Bei Argumenten, die einem Kommandozeileninterpreter (engl. Shell) übergeben werden, besteht zum Beispiel häufig das Problem, dass Pfadangaben Leerzeichen enthalten. In der URL-Notation werden Leerzeichen als „%20“ kodiert (s. Tabelle 59). Die Funktion ConvertToURL konvertiert eine systemspezifische Pfadangabe zur URL-Notation, und ConvertFromURL konvertiert wiederum zum Systempfad.

Listing 154. *Konvertierung zu und von URL.*

```
Sub ToFromURL
  Print ConvertToURL("/home/andy/logo.miff")
  Print ConvertFromURL("file:///home/andy/logo.miff") 'Dies geht nur mit UNIX
  Print ConvertToURL("c:\My Documents")               'Dies geht nur mit Windows
  Print ConvertFromURL("file:///c:/My%20Documents")   'Dies geht nur mit Windows
End Sub
```

Sonderzeichen wie das Leerzeichen werden mit einem Prozentzeichen (%) und nachfolgendem ASCII-Wert des Zeichens als zweistellige Hexadezimalzahl kodiert. Das Leerzeichen hat den ASCII-Wert 32, hexadezimal 20. Daher wird es als %20 kodiert.

Listing 155. *Sonderzeichen in URLs.*

```
Sub URLSpecialEncoding
  Print ConvertFromURL("file:///%41%42%43/%61%62%63")   '/ABC/abc   (UNIX)
  Print ConvertFromURL("file:///c:/%41%42%43/%61%62%63")'c:\ABC\abc   (Windows)
End Sub
```

Achtung ConvertFromURL behandelt Pfadangaben immer so, als ob sie für das aktuelle OS gälten.

Auf Unix-basierten Systemen (Linux, macOS) werden Windows-Laufwerksbuchstaben als Verzeichnisnamen interpretiert: aus „file:///c:/%41%42%43/%61%62%63“ wird „/c:/ABC/abc“.

Auf Windows werden Pfadangaben ohne Laufwerksbuchstaben gar nicht umgesetzt.: „file:///%41%42%43/%61%62%63“ bleibt also „file:///%41%42%43/%61%62%63“.

Die URL-Notation ist systemunabhängig, so dass URL-Pfade genauso wie auf einem Apple- auch auf einem Windows-Rechner funktionieren. Zum Erzeugen eines systemspezifischen Pfades benötigt man den entsprechenden Pfadtrenner, den die Funktion GetPathSeparator liefert. Wie man GetPathSeparator nutzt, um einen kompletten Pfad zu erzeugen, zeigt Listing 156. Unter Windows wird „\“ als Pfadtrenner verwendet, unter Unix jedoch „/“. In der URL-Notation ist der Pfadtrenner immer „/“ – ungeachtet des Betriebssystems.

Listing 156. *Verwenden Sie GetPathSeparator() anstatt „\“ oder „/“.*

```
sPathToFile = "C:\temp"
sBookName   = "OOME.odt"
sPathToBook = sPathToFile & GetPathSeparator() & sBookName
```

Tipp Visual Basic for Applications (VBA) kennt die Funktion GetPathSeparator nicht, hat aber die Eigenschaft Application.PathSeparator, die immer einen Backslash (\) zurückgibt, sogar auf Macintosh-Rechnern. VBA kennt auch nicht die Funktionen ConvertToURL und ConvertFromURL.

8.2. Funktionen zur Bearbeitung von Verzeichnissen

Manche Funktionen sind sowohl auf Verzeichnisse als auch auf Dateien anwendbar. In diesem Abschnitt geht es um solche, die nur mit Verzeichnissen arbeiten.

Die Funktion CurDir gibt – mit dem Laufwerk als Argument – das aktuelle Arbeitsverzeichnis des angegebenen Laufwerks zurück, s. Listing 157 und Bild 59. Wenn das Argument fehlt, wird das aktuell genutzte Laufwerk angenommen. Auf Unix-Systemen wird die Laufwerksangabe ignoriert. Mit welchem Arbeitsverzeichnis OOo startet, ist einerseits abhängig vom System und andererseits davon, auf welche Weise OOo gestartet wird. Wenn Sie OOo von einer Kommandozeile aus aufrufen, werden Sie wahrscheinlich ein anderes Arbeitsverzeichnis haben als wenn Sie OOo aus einem Menü oder einer anderen Anwendung heraus aufrufen. Wenn Sie über „Datei | Öffnen“ ein bestehendes Dokument öffnen, wird unter manchen Betriebssystemen das Verzeichnis, in dem das geöffnete Dokument liegt, zum aktuellen Arbeitsverzeichnis (ich habe das Verhalten unter Windows gesehen). Unter anderen Betriebssystemen wie zum Beispiel Linux wird das aktuelle Arbeitsverzeichnis davon nicht berührt. Unter Apples macOS erhalten Sie das Wurzelverzeichnis „/“, außer wenn Sie OOo aus einem Terminal heraus starten. Verlassen Sie sich also nicht auf das Ergebnis!

Listing 157. *Ausgabe des aktuellen Arbeitsverzeichnisses.*

```
Sub ExampleCurDir
  MsgBox "Das aktuelle Arbeitsverzeichnis auf diesem Rechner ist " &_
         CurDir, 0, "Beispiel für das aktuelle Arbeitsverzeichnis"
End Sub
```

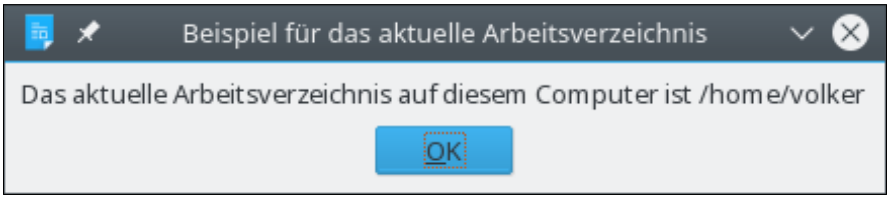


Bild 59. *CurDir gibt das aktuelle Arbeitsverzeichnis zurück.*

Obwohl es in Basic die Funktionen ChDir und ChDrive gibt, so bewirken Sie nichts und werden wohl aus der Sprache verschwinden. Sie hatten ursprünglich die Aufgabe, das aktuelle Laufwerk wie auch das aktuelle Verzeichnis zu wechseln. Das aber wirkt sich systemweit aus und ist in unseren heutigen Multitasking-Umgebungen gefährlich. Das ursprüngliche Arbeitsverzeichnis hängt vom Betriebssystem ab und von der Art und Weise, wie OOo gestartet wird. Man kann also nicht von einem gesicherten Wert ausgehen.

Die Funktion MkDir erstellt ein Verzeichnis, RmDir löscht ein Verzeichnis. Im Listing 158 wird ein Verzeichnispfad aus dem MkDir-Argument erstellt. Wenn es keine absolute Pfadangabe ist, wird das neue Verzeichnis relativ zum aktuellen Arbeitsverzeichnis erstellt, demselben, das mit der Funktion CurDir ermittelt wird. Die Funktion RmDir löscht das Verzeichnis, alle Verzeichnisse darunter und auch alle darin enthaltenen Dateien.

Das folgende Makro ruft OOMEWorkDir (Listing 242) auf, das einen Pfad mit dem von AOO oder LO als Nutzerarbeitsverzeichnis geführten Verzeichnis und dem Unterverzeichnis „OOMEWork“ zurückgibt. Es wird erst im Abschnitt 10.12 Services für Dateien und Verzeichnisse vorgestellt, weil es einen UNO-Service nutzt. Das Makro CreateOOMEWorkDir (Listing 243) legt dieses Unterverzeichnis an und gibt False zurück, falls der Vorgang gescheitert ist.

Das Makro Listing 158 legt im Unterverzeichnis „OOMEWork“ ein Unterverzeichnis „a“ mit einem Unterverzeichnis „b“ an. Während die „Erstellt“-Meldung angezeigt wird, können Sie sich davon überzeugen, dass diese Verzeichnisse tatsächlich existieren. Wenn Sie auf „OK“ klicken, wird das gesamte „OOMEWork“-Verzeichnis mit allen Inhalten gelöscht. Für den Fall, dass Sie überraschenderweise ein solches Verzeichnis mit anderen Inhalten nutzen, starten Sie das folgende Makro nur, wenn Sie bereit sind, alle Inhalte zu verlieren.

Listing 158. *Erstellt Verzeichnisse im OOo-Arbeitsverzeichnis und löscht sie wieder.*

```
Sub ExampleCreateRmDirs
  'Im OO-Arbeitsverzeichnis wird ein Unterverzeichnis "OOMEWork" erstellt.
  'Abbruch, falls der Schreibvorgang auf der Festplatte fehlschlägt.
  If Not CreateOOMEWorkDir() Then
    Exit Sub
  End If

  Dim sWorkDir$              'Pfad des OOMEWork-Verzeichnisses
  Dim sPath$                 'Pfad für die Unterverzeichnisse
  sWorkDir = OOMEWorkDir()
  sPath = sWorkDir & "a" & GetPathSeparator() & "b"  '.../OOMEWork/a/b
  MkDir  sPath
  Print "Erstellt: " & sPath
  RmDir  sWorkDir            'OOMEWork wird gelöscht, mit allen Inhalten.
  Print "Gelöscht: " & sWorkDir
End Sub
```

Der Code im Listing 158 verwendet absolute Pfade. Relative Pfadangaben sind auch möglich, ich möchte aber dringend davon abraten, weil die Ermittlung des Arbeitsverzeichnisses systemabhängig ist.

Dateifunktionen, die auch auf Verzeichnisse angewendet werden können, sind Dir, FileDateTime, FileExists, FileLen, GetAttr und Name. Zu diesen kommen wir jetzt.

8.3. Funktionen zur Dateibearbeitung

Dieses Kapitel behandelt Funktionen, mit denen man Dateien als Einheit bearbeiten und ihre Eigenschaften ermitteln kann, im Gegensatz zur Bearbeitung der Dateiinhalte. Manche dieser Funktionen sind sowohl für Dateien als auch für Verzeichnisse anwendbar. Jedenfalls akzeptieren die Funktionen wenigstens ein Argument als Datei- oder Verzeichnisangabe. Für Argumente, die Dateien oder Verzeichnisse benennen, gilt Folgendes:

Wenn der Pfad fehlt, wird das aktuelle Arbeitsverzeichnis genommen – wie von CurDir geliefert.

Sowohl die systemspezifische als auch die URL-Notation sind erlaubt. Zum Beispiel beziehen sich „C:\tmp\foo.txt“ und „file:///c:/tmp/foo.txt“ auf dieselbe Datei.

Einzelne Dateien oder Verzeichnisse müssen eindeutig identifizierbar sein, wenn nicht ausdrücklich anders festgelegt. Einzig die Funktion Dir akzeptiert Namensmuster und gibt eine Liste von Dateien zurück, die dem Muster entsprechen.

Alle Dateien und Verzeichnisse haben Attribute, s. Tabelle 60. Jedes Attribut wird durch ein bestimmtes Bit in einer Zahl dargestellt, so dass man jedem Element in einem Pfad gleichzeitig mehrere Attribute zuweisen kann. Von manchen Attributen ist abzuraten, weil sie systemabhängig sind. Nicht alle Systeme kennen zum Beispiel verborgene oder Systemdateien. Mit GetAttr lesen Sie die Attribute aus.

Tabelle 60. *Datei- und Verzeichnisattribute.*

Abzuraten	Attribut	Beschreibung
Nein	0	Normal; keine Bits gesetzt
Nein	1	Schreibgeschützt
Ja	2	Verborgen
Ja	4	System
Nein	8	Datenträger
Nein	16	Verzeichnis
Nein	32	Archiv (Datei wurde nach dem letzten Backup verändert)

Die Funktion im Listing 159 akzeptiert ein Dateiattribut, geliefert von der Funktion GetAttr, und gibt einen leicht verständlichen Text zurück. Wenn keine Bits gesetzt sind, gilt das als Attribut für eine normale Datei.

Listing 159. *Gibt die Dateiattribute als String aus.*

```
REM Verwendet einen Bitvergleich, um die Attribute zu ermitteln
Function FileAttributeString(x As Integer) As String
  Dim s As String
  If (x = 0) Then
    s = "Normal"
  Else
    If (x And 16) <> 0 Then s = "Verzeichnis"         'Verzeichnis-Bit 00010000 gesetzt
    If (x And 1)  <> 0 Then s = s & " Schreibschutz" 'Schreibschutz-Bit 00000001
                                                      'gesetzt
    If (x And 2)  <> 0 Then s = s & " Verborgen"      'Abzuraten
    If (x And 4)  <> 0 Then s = s & " System"         'Abzuraten
    If (x And 8)  <> 0 Then s = s & " Datenträger"    'Datenträger-Bit 00001000 gesetzt
    If (x And 32) <> 0 Then s = s & " Archiv"         'Archiv-Bit 00100000 gesetzt
  End If
  FileAttributeString = s
End Function
```

Tipp Listing 159 führt Bitoperationen (siehe Abschnitt 8.4. Dateiattribute, Bitmasken und Binärzahlen) zur Ermittlung der gesetzten Attribute durch.

Mit der Funktion GetAttr werden die Dateiattribute ermittelt, mit SetAttr werden sie gesetzt. Das erste Argument zur Funktion SetAttr ist der Dateiname – relativ oder absolut. Das zweite Attribut ist eine Zahl, die die zu setzenden und zu löschenden Attribute repräsentiert. Nachdem Sie also SetAttr(Name, n) aufgerufen haben, sollte die Funktion GetAttr(Name) die Zahl n zurückgeben. Haben Sie zum Beispiel SetAttr mit dem Attribut 32 aufgerufen und damit das Archiv-Bit gesetzt und alle anderen gelöscht, so wird GetAttr die Zahl 32 zurückgeben. Um mehr als ein Bit in einem Aufruf zu setzen, kombinieren Sie die Attribute mit dem Operator Or. SetAttr(Name, 1 Or 32) setzt sowohl das Archiv- als auch das Schreibschutz-Bit. SetAttr wirkt auf Verzeichnisse wie auch auf Dateien.

Tipp Attribute bevorzugen die Windows-Umgebung. In Unix-basierten Systemen wie Linux und Sun werden durch das Setzen der Attribute die Rechte für Nutzer, Gruppe und Sonstige geändert. Das Attribut 0 (nicht schreibgeschützt) entspricht „rwxrwxrwx“, das Attribut 1 (schreibgeschützt) entspricht „r--r--r--“.

Mit der Funktion FileLen ermitteln Sie die Dateigröße. Der Rückgabewert ist vom Typ Long. Die Funktion im Listing 160 liest die Dateigröße und gibt sie als leicht lesbaren Text aus, und zwar in Bytes, je nach Größe auch mit K, MB, G oder T. Das Resultat ist leichter verständlich als eine lange Zahl.

Listing 160. *Gibt eine Zahl in leicht lesbarer Form aus, zum Beispiel als 2K statt 2048.*

```
Function PrettyFileLen(path$) As String
  PrettyFileLen = nPrettyFileLen(FileLen(path))
End Function

Function nPrettyFileLen(ByVal n As Double) As String
  Dim i As Integer    'Durchlaufzähler
  Dim v() As Variant  'Abkürzungen für Kilobytes, Megabytes, ...
  v() = Array("Bytes", "K", "MB", "G", "T") ' Abkürzungen

  REM Jedes Mal, wenn die Zahl um 1 Kilobyte verringert wird,
  REM wird der Zähler um 1 erhöht.
  REM Die Größe darf 1 Kilobyte nicht unterschreiten.
  REM Der Zähler darf die Größe des Arrays nicht überschreiten.
  Do While n > 1024 And i + 1 < UBound(v())
    n = Fix(n / 1024) 'Trunkierung nach der Division
    i = i + 1         'Start bei i=0 (Bytes), Heraufzählung zur nächsten Abkürzung
  Loop
  nPrettyFileLen = CStr(n) & v(i)
End Function
```

Mit der Funktion FileExists prüfen Sie, ob eine Datei oder ein Verzeichnis existiert. Mit FileDateTime erhalten Sie einen String mit Datum und Uhrzeit der Erstellung oder der letzten Änderung der Datei, und zwar im systemabhängigen Format. Auf meinem Rechner ist es „DD.MM.JJJJ HH:MM:SS“. Dieser String kann direkt an die Funktion CDate übergeben werden. Das Makro ExampleGetFileInfoPath() im Listing 161 verwendet die Funktion ChooseAFileName(), die einen Dialog zur Dateiauswahl öffnet (s. Listing 205). Die Funktion GetFileInfo nutzt alle Funktionen zur Überprüfung von Dateien und Verzeichnissen und gibt die Informationen in leicht lesbarer Form aus, s. Bild 60.

Listing 161. *Informationen über eine Datei ermitteln.*

```
Sub ExampleGetFileInfoPath()
  MsgBox GetFileInfo(ChooseAFileName())
End Sub

Function GetFileInfo(path) As String
  Dim s As String
  Dim iAttr As Integer
  s = "Der Pfad """ & path & """"
  If Not FileExists(path) Then
    GetFileInfo = s & " existiert nicht"
    Exit Function
  End If
  s = s & " existiert" & Chr$(10)
  s = s & "Datum und Uhrzeit = " & FileDateTime(path) & Chr$(10)
  iAttr = GetAttr(path)
  REM Die Größe eines Verzeichnisses ist immer null
  If (iAttr And 16) = 0 Then
    s = s & "Dateigröße = " & PrettyFileLen(path) & Chr$(10)
  End If
  s = s & "Attribute = " & FileAttributeString(iAttr) & Chr$(10)
  GetFileInfo = s
End Function
```

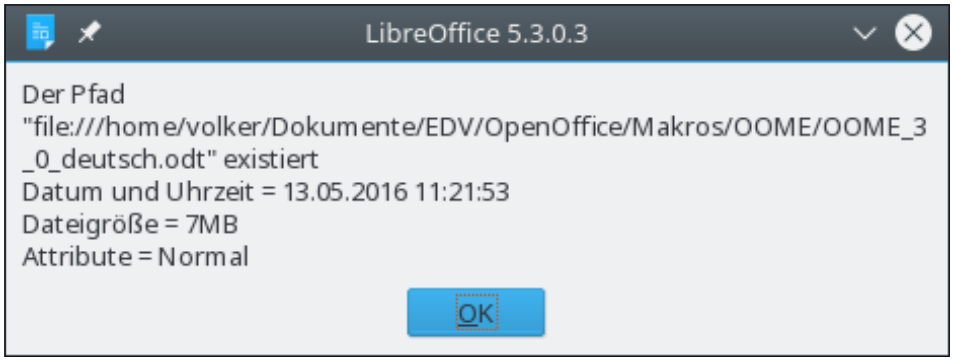


Bild 60. *Funktionen zur Ermittlung von Informationen über Dateien.*

Mit der Funktion Kill löschen Sie eine Datei vom Speichermedium. Wenn die Datei nicht existiert, wird ein Laufzeitfehler generiert.

```
Kill("C:\temp\BadFile.txt")
```

Mit der Funktion FileCopy werden Dateien kopiert. Das erste Argument enthält die zu kopierende Datei (Quelle), das zweite Argument die Zieldatei, s. Tabelle 61. FileCopy ist in der Lage, rekursiv komplette Verzeichnisbäume zu kopieren, kann aber nicht mit Namensmustern umgehen. Wenn das erste Argument eine Datei ist, muss das zweite Argument erstaunlicherweise auch eine Datei sein – ich hätte erwartet, dass man eine Datei mit FileCopy in ein anderes Verzeichnis hinein kopieren kann, etwa FCopy("C:\auto.bat", "C:\bak\"). Geht aber nicht.

Tabelle 61. *Argumente zu FileCopy.*

Gültig	Quelle	Ziel	Kommentar
Ja	Datei	Datei	Kopiert die Datei. Die Namen müssen nicht gleich sein.
Ja	Verzeichnis	Verzeichnis	Kopiert rekursiv alle in einem Verzeichnis enthaltenen Dateien und Verzeichnisse in ein anderes Verzeichnis.
Nein	Namensmuster		Dateinamensmuster (Joker, zum Beispiel *.*) sind nicht erlaubt.
Nein	Datei	Verzeichnis	Wenn die Quelle eine Datei ist, muss das Ziel auch eine Datei sein.

```
FileCopy("C:\auto.bat", "C:\auto.bak")        'Kopiert eine Datei
FileCopy("C:\auto.bat", "C:\tmp\auto.bat")    'Kopiert eine Datei
FileCopy("C:\logs", "C:\bak")                 'Kopiert ein Verzeichnis
```

Achtung Kopieren Sie nicht rekursiv ein Verzeichnis in sich selbst – Sie erzeugen eine Endlosschleife. Zum Beispiel wird FileCopy("C:\logs", "C:\logs\bak") unendlich weiterlaufen, weil das Unterverzeichnis „bak“ sofort ein Teil des Inhalts von „logs“ wird und somit auch kopiert werden muss. Keine gute Idee.

Die Anweisung Name benennt eine Datei oder ein Verzeichnis um. Diese Anweisung hat eine ungewöhnliche Syntax: zwischen Quell- und Zielnamen steht das Schlüsselwort As.

```
Name "C:\Joe.txt" As "C:\bill.txt"     'Umbenennung einer Datei
Name "C:\logs" As "C:\oldlogs"         'Umbenennung eines Verzeichnisses
Name "C:\Joe.txt" As "C:\tmp\joe.txt" 'Verschiebt die Datei in das Verzeichnis tmp
Name "C:\logs" As "C:\bak\logs"        'Verschiebt das Verzeichnis logs
```

Tipp Ein Power-User-Trick: Verwenden Sie das Kommando Name, um eine Datei oder ein Verzeichnis von hier nach dort zu verschieben.

8.4. Dateiattribute, Bitmasken und Binärzahlen

Es ist gar nicht nötig, Binärzahlen und Bitmasken zu verstehen, wenn man entweder Dateiattribute oder Bitmasken in Basic verwendet, also keine Panik. Überspringen Sie einfach die Abschnitte, bei denen Ihnen der Kopf raucht. Ein Verständnis dieser Materie erleichtert allerdings den Zugang zu den Dateiattributen.

Die Datei- und Verzeichnisattribute der Tabelle 60 sind gezielt so gewählt, dass sie, als Zahlen mit der Basis 2, eine feine Eigenschaft besitzen – jedes Attribut besetzt nur ein Bit. Null ist ein Sonderfall – es sind keine Bits gesetzt.

Tabelle 62. *Bitwerte der Datei- und Verzeichnisattribute.*

Attribut dezimal	Attribut binär	Beschreibung	Kommentar
00	0000 0000	Normal	Keine Bits gesetzt
01	0000 0001	Schreibschutz	Bit 1 gesetzt
02	0000 0010	Verborgen	Bit 2 gesetzt
04	0000 0100	System	Bit 3 gesetzt
08	0000 1000	Datenträger	Bit 4 gesetzt
16	0001 0000	Verzeichnis	Bit 5 gesetzt
32	0010 0000	Archiv	Bit 6 gesetzt

Mit GetAttr erhält man die Attribute einer Datei oder eines Pfads. Ist die Datei oder der Pfad ein Verzeichnis, ist das Bit 5 gesetzt. Ist die Datei oder der Pfad schreibgeschützt, ist Bit 1 gesetzt. Der Attributwert 0 bedeutet, dass keine Bits gesetzt sind und es sich um eine normale Datei handelt. Nehmen wir einmal den Attributwert 33, in Binärdarstellung 0010 0001. Bit 1 ist gesetzt, heißt also schreibgeschützt. Bit 6 ist gesetzt, heißt also, dass die Datei seit der letzten Archivierung verändert wurde. Sie brauchen demnach gar nicht zu wissen, wie eine Dezimalzahl in eine Binärzahl konvertiert wird. Sie müssen jedoch ein Makro schreiben können, das ermittelt, welche Bits gesetzt und welche nicht gesetzt sind. Mit dem Operator And stellen Sie fest, welche Bits gesetzt sind. Mit And müssen zwei Dinge gleich sein, damit die Antwort wahr ist. Meine Taschenlampe funktioniert zum Beispiel, wenn sie eine Birne hat And wenn Batterien eingesetzt sind.

Der Operator And führt bei Zahlen die logische Operation mit jedem einzelnen Bit durch. „3 And 5“ zum Beispiel stellt sich binär als „0011 And 0101 = 0001“ dar. Bit 1 – das Bit ganz rechts – ist in

beiden Zahlen gesetzt, also ist Bit 1 im Resultat auch 1. Alle anderen Bits haben an der vergleichbaren Stelle nicht übereinstimmend den Wert 1, somit werden die entsprechenden Resultatbits zu 0.

Wenden wir dieses Konzept nun auf die aktuelle Aufgabe an. Ist der numerische Wert eines Attributs nicht null, ist wenigstens eine Eigenschaft gesetzt. Unter dieser Voraussetzung können Sie jedes Attribut so prüfen, wie es die Tabelle 63 zeigt.

Tabelle 63. *Dateieigenschaften am Beispiel des Attributwerts 33 (100001).*

Schreibschutz	Verborgen	System	Datenträger	Verzeichnis	Archiv
10 0001 And 00 0001 (1) 00 0001	10 0001 And 00 0010 (0) 00 0000	10 0001 And 00 0100 (0) 00 0000	10 0001 And 00 1000 (0) 00 0000	10 0001 And 01 0000 (0) 00 0000	10 0001 And 10 0000 (32)10 0000

In Basic sieht der Code etwa so aus:

```
If TheAttribute = 0 Then
  REM Keine Attribute gesetzt
Else
  If (TheAttribute And 1) = 1 Then ...   'Schreibgeschützte Datei: Bit 1 ist gesetzt
  If (TheAttribute And 16) = 16 Then ... 'Verzeichnis: Bit 5 ist gesetzt
  If (TheAttribute And 4) <> 0 Then ...  'Dasselbe auf eine andere Weise
End If
```

Jede Datei und jedes Verzeichnis hat ein Attribut mit diesen Bitmustern. Wenn ein Bit in diesem Attribut gesetzt ist, das zu einer bestimmten Eigenschaft gehört, dann hat die Datei diese Eigenschaft. Wenn man den Operator And auf diese bestimmten Bitpositionen anwendet, erfährt man, ob die Datei diese Eigenschaften besitzt. Nach dieser Methode arbeitet die Funktion FileAttributeString im Listing 159.

Um das Archivbit und das Schreibschutzbit einer Datei zu setzen, kombinieren Sie die Bits und rufen die Funktion nur einmal auf. Die Bitmuster kombinieren Sie mit dem Operator Or. Wenn eines der beiden Bits gesetzt ist, liefert Or das Ergebnisbit 1. Um also das Archivbit und das Schreibschutzbit zu setzen, schreiben Sie „1 Or 32“. Wenn Sie den Attributwert komplett auf 1 setzen, werden alle anderen Attribute gelöscht und nur das Schreibschutzbit wird gesetzt.

8.5. Auflistung eines Verzeichnisinhalts

Mit der Funktion Dir wird der Inhalt eines Verzeichnisses ermittelt. Das erste Argument enthält das Dateinamensmuster. Eine Datei oder ein Verzeichnis darf zwar eindeutig benannt sein, üblicher ist aber der Gebrauch von Platzhaltern (auch bekannt als Joker). Das Kommando Dir("C:\temp*.txt") gibt zum Beispiel eine Liste aller Dateien mit der Namenserweiterung txt zurück. Das zweite Argument ist für Attribute, von denen zwei Werte verwendet werden können: 0 (Standard) für eine Liste von Dateien, 16 für Verzeichnisse.

Tipp Die meisten Betriebssysteme verwenden zwei besondere Verzeichnisnamen, den einfachen Punkt „.“ und zwei Punkte „..“. Der einfache Punkt steht für das aktuelle Verzeichnis, zwei Punkte stehen für das übergeordnete Verzeichnis. Diese beiden speziellen Verzeichnisnamen sind in der von der Funktion Dir zurückgegebenen Verzeichnisliste enthalten. Wenn Sie ein Makro schreiben, das rekursiv in jedes untergeordnete Verzeichnis schaut, und Sie diese beiden nicht berücksichtigt haben, wird Ihr Makro ewig weiterlaufen.

Der erste Aufruf von Dir gibt die erste passende Datei beziehungsweise das erste passende Verzeichnis zurück. Jeder weitere Aufruf, dann aber ohne Argument, gibt den jeweils nächsten Treffer zurück.

```
sFileName = Dir(path, attribute)      'Holt den ersten Treffer
Do While (sFileName <> "")            'Wenn überhaupt etwas gefunden wurde
```

```
    sFileName = Dir()                   'Holt den nächsten Treffer
  Loop
```

Wenn der Pfad ein eindeutiger Datei- oder Verzeichnisname ist, gibt es nur einen Treffer. Das Kommando Dir("C:\tmp\autoexec.bat") gibt zum Beispiel nur die Datei „autoexec.bat“ zurück. Weniger offensichtlich ist, dass das Kommando Dir("C:\tmp") nur das Verzeichnis „tmp“ zurückgibt. Wenn Sie auch die Verzeichnisinhalte sehen wollen, müssen Sie dem Pfad entweder Jokerzeichen anfügen (C:\tmp*.*) oder einen abschließenden Verzeichnistrenner (C:\tmp\). Der Code im Listing 162 erstellt eine einfache Liste des Inhalts des aktuellen Verzeichnisses. Der Verzeichnistrenner wird durch die Funktion GetPathSeparator ermittelt, damit das Makro unabhängig vom Betriebssystem ist. Unter Apples macOS ist das aktuelle Verzeichnis das Wurzelverzeichnis „/“, außer wenn OOo aus einem Terminal heraus gestartet wurde.

Listing 162. *Auflistung der Dateien des aktuellen Verzeichnisses.*

```
Sub ExampleDir
  Dim s As String                       'Temporärer String
  Dim sFileName As String               'Der jeweils letzte von DIR gelieferte Name
  Dim i As Integer                      'Zählt die Verzeichnisse und Dateien
  Dim sPath                             'Aktueller Pfad mit Verzeichnistrenner am Ende
  sPath = CurDir & GetPathSeparator()   'Ohne Verzeichnistrenner gibt DIR das Verzeichnis
  sFileName = Dir(sPath, 16)            'zurück und nicht dessen Inhalt
  i = 0                                 'Variableninitialisierung
  Do While (sFileName <> "")            'Wenn überhaupt etwas zurückgegeben wird
    i = i + 1                           'Zählung der Verzeichnisse
    s = s & "Verzeichnis " & CStr(i) &_
        " = " & sFileName & Chr$(10)    'Name in den Ausgabestring
    sFileName = Dir()                   'Der nächste Verzeichnisname
  Loop
  i = 0                                 'Start der Zählung für Dateien
  sFileName = Dir(sPath, 0)             'Jetzt werden Dateien geholt!
  Do While (sFileName <> "")
    i = i + 1
    s = s & "Datei " & CStr(i) & " = " & sFileName & "  " &_
        PrettyFileLen(sPath & sFileName) & Chr$(10)
    sFileName = Dir()
  Loop
  MsgBox s, 0, ConvertToURL(sPath)
End Sub
```

Ein Beispiel für Listing 162 sehen Sie im Bild 61. Zuerst werden die Verzeichnisse aufgelistet. Die ersten beiden Verzeichnisse „.“ und „..“ stehen für die folgenden:

```
file:///home/volker/Dokumente/OpenOffice/
file:///home/volker/Dokumente/
```

Die Einbeziehung von „.“ und „..“ sorgt immer wieder für Probleme. Eine Verzeichnisliste enthält zwar diese Verzeichnisse, man sollte sie aber im allgemeinen ignorieren.

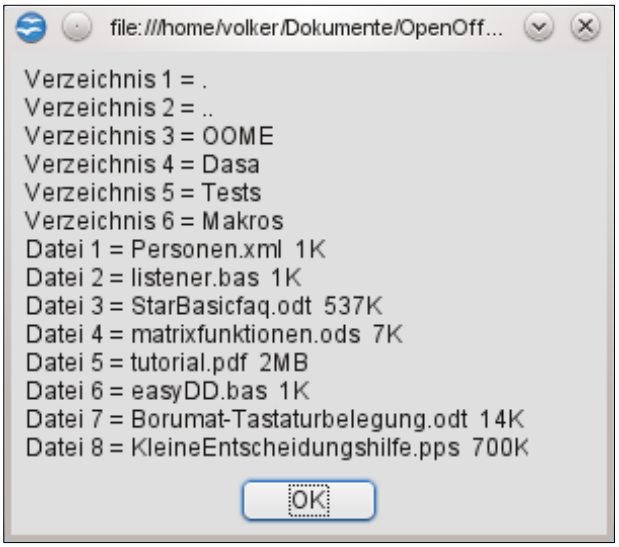


Bild 61. *Inhalt des aktuellen Verzeichnisses.*

8.6. Eine Datei öffnen

OOo greift zur Dateibearbeitung auf die systemspezifischen Methoden der unteren Ebene zu. Das Betriebssystem führt eine Liste der offenen Dateien und kennzeichnet sie mit einer Nummer, dem so genannten Dateihandle. In der Basic-Welt heißt sie „Dateinummer“ oder „Datenkanal“.

Zum Öffnen einer Datei brauchen Sie eine Dateinummer. Die Funktion FreeFile stellt Ihnen eine freie Dateinummer zur Verfügung, über die Sie eine Datei öffnen, auf die geöffnete Datei zugreifen und die Datei wieder schließen. Mit der Anweisung Open öffnen Sie eine Datei, bevor Sie zum Lesen oder Schreiben darauf zugreifen können. Dazu brauchen Sie die Dateinummer, die Sie immer zuvor mit der Funktion FreeFile ermitteln. Wenn Sie die Datei nicht mehr benötigen, schließen Sie sie mit der Anweisung Close. In einer einzigen Anweisung können Sie mehrere Dateien schließen: Fügen Sie dem Kommando Close eine durch Kommas getrennte Liste der Dateinummern an. Mit der Funktion Reset werden alle geöffneten Dateien zugleich geschlossen. Sie brauchen sie nicht einzeln aufzuführen. Die Dateien werden alle geschlossen und die Daten, die sich noch im Arbeitsspeicher befinden, auf den Datenträger geschrieben. Die Dateinummern sind nun wieder frei.

```
n = FreeFile()
Open DateiName For Modus [Access ioModus] [Lock Mode] As #n [Len = Datensatzlänge]
Close #n
```

„DateiName“ ist der Name der zu öffnenden Datei. Wenn der Dateiname keine Pfadangabe enthält, wird das aktuelle Arbeitsverzeichnis angenommen. „For Modus“ bestimmt den Dateistatus nach dem Öffnen und die beabsichtigte Zugriffsart, s. Tabelle 64.

Tabelle 64. *Gültige Werte für „Modus“ mit resultierendem Status ohne die „Access“-Angabe.*

For Modus	Dateizeiger	Datei existiert	Keine Datei	Lesen	Schreiben	Kommentar
For Append	Ende	Öffnen	Erstellen	Ja	Ja	Sequenzieller Zugriff
For Input	Anfang	Öffnen	Error	Ja	Nein	Sequenzieller Zugriff
For Output	Anfang	Löschen	Erstellen	Ja	Ja	Sequenzieller Zugriff
For Binary	Anfang	Löschen	Erstellen	Ja	Ja	Wahlfreier Zugriff
For Random	Anfang	Löschen	Erstellen	Ja	Ja	Wahlfreier Zugriff

Jeder Modus verhält sich auf eine eigene Art. Nehmen wir einmal die Zeile For Input, die folgendermaßen gelesen werden kann: Wenn eine Datei „For Input“ geöffnet wird, dann

1. wird der Dateizeiger an den Anfang der Datei gesetzt,

2. wird die Datei geöffnet, wenn sie existiert (sie wird nicht gelöscht),
3. wird ein Laufzeitfehler erzeugt, wenn die Datei nicht existiert,
4. wird die Datei für den lesenden Zugriff geöffnet, aber nicht für den schreibenden (immer noch angenommen, dass „Access“ nicht ausdrücklich beigefügt ist),
5. wird der lesende Zugriff sequenziell sein.

Leider hängt das konkrete Verhalten vom Betriebssystem ab und auch von dem Compiler, der Ihre OOo-Version erstellt hat. Auf manchen Systemen kann man zum Beispiel in eine Datei schreiben, die für lesenden Zugriff geöffnet wurde.

Beim Öffnen einer Datei wird ein Zeiger in die Datei bereitgestellt. Dieser Zeiger enthält die Position, an der der nächste Lese- oder Schreibzugriff erfolgt. Wenn der Dateizeiger zum Beispiel am Dateianfang steht, wird die nächste „Read“-Anweisung vom Dateianfang lesen. Wenn der Dateizeiger am Dateiende steht, wird die nächste „Write“-Anweisung Daten an das Dateiende anfügen. Man hat beim Öffnen der Datei eine gewisse Kontrolle über die Startposition des Zeigers, und man kann den Zeiger verschieben, nachdem die Datei geöffnet ist. Mit Ausnahme des Modus „For Append“ wird der Dateizeiger immer auf den Dateianfang gesetzt.

Der Zugriff auf eine Datei kann sequenziell oder wahlfrei sein. Eine sequenzielle Datei ist mit einem Videoband vergleichbar. Man kann zwar das Band mit schnellem Vor- und Rücklauf zu einer bestimmten Position spulen, dabei bewegt sich aber das gesamte Band am Schreib-/Lesekopf vorbei. Wenn man dann Abspielen oder Aufnehmen wählt, werden die Daten sequenziell vom Band gelesen oder auf das Band geschrieben. Eine wahlfreie Datei verhält sich wie eine Musik-CD. Obwohl man die CD sequenziell abspielen kann, muss das nicht sein, denn man kann direkt zu jedem Song springen und von dort abspielen. Zur korrekten Analogie müssten allerdings alle Songs auf der CD dieselbe Länge haben. Das ist der Nachteil des Modus „For Random“.

Gesetzt den Fall, Sie speichern in einer Datei Namen unterschiedlicher Länge. Es spart Platz, wenn Sie pro Zeile einen Namen unterbringen. Zwischen den Namen steht ein Zeilenumbruchzeichen. Wenn Sie einen Namen in der Datei suchen, starten Sie am Anfang und lesen die Daten so lange, bis Sie auf den gewünschten Namen treffen. Im Gegensatz dazu wissen Sie vielleicht, dass der längste Name 100 Zeichen lang ist. Somit können Sie kürzere Namen mit entsprechend vielen Leerzeichen ergänzen, so dass Sie für jeden Namen genau 100 Zeichen speichern. Das verschwendet Platz, dafür können Sie aber wegen der gleichmäßigen Dateistruktur schnell zwischen den Namen wechseln. Um den 1000. Namen zu lesen, springen Sie einfach direkt dorthin. Sie haben mit dieser Anordnung zwar Platz vergeudet, aber Geschwindigkeit gewonnen. Die „For“-Modi Append, Input und Output bieten sequenziellen, Binary und Random wahlfreien Zugang. In wahlfreien Dateien sind die Datensätze auf die Länge fixiert, die vom längsten möglichen Datensatz benötigt wird. Damit bieten sie einen sehr schnellen Zugriff auf die enthaltenen Daten.

Der „Access“-Modus der Tabelle 65 bestimmt das Standardverhalten einer Datei, wenn Sie geöffnet ist. Der Modus stellt das Lese- oder Schreibrecht sicher. Wenn Sie kein Schreibrecht an einer Datei haben, die mit „Access Write“ geöffnet wurde, wird ein Laufzeitfehler generiert. Der „Access“-Modus wirkt auf jeden „Open For“-Modus außer auf „For Append“ – dabei wird eine existierende Datei beim Öffnen auf keinen Fall gelöscht.

Tabelle 65. *Gültige Werte für „Access ioModus“.*

Access ioModus	Beschreibung
Access Read	Eine existierende Datei wird nicht gelöscht. Lesezugriff wird verifiziert.
Access Write	Eine existierende Datei wird gelöscht. Schreibzugriff wird verifiziert.
Access Read Write	Eine existierende Datei wird gelöscht. Lese- und Schreibzugriff wird verifiziert.

Wenn Sie eine Datei „For Input“ mit „Access Write“ öffnen, können Sie in diese Datei schreiben, wenn sie geöffnet ist. Die Datei wird erst gelöscht und dann neu angelegt. Nach dem Öffnen wird das Zugriffsrecht in den verschiedenen Betriebssystemen unterschiedlich gehandhabt. Seit OOo 1.1.1 können Sie auf eine wahlfreie oder Binärdatei, die Sie mit „Access Read“ geöffnet haben, schreibend zugreifen, aber nur unter Windows, nicht unter Linux. Der sicherste Weg führt immer über „For Append“. Springen Sie dann mit dem Dateizeiger an den Dateianfang.

Tipp Der einzig sichere Weg, eine Datei zum Lesen und Schreiben zu öffnen, ohne den Inhalt zu löschen, besteht darin, die Datei „For Append“ zu öffnen und dann den Dateizeiger an den Dateianfang zu setzen.

Der Lock-Modus dient dazu, den Zugang auf eine geöffnete Datei zu beschränken, s. Tabelle 66. Wenn Sie eine Datei geöffnet haben, können andere Nutzer dann nicht lesend oder schreibend darauf zugreifen. Dieser Modus ist nur in Mehrbenutzersystemen sinnvoll, weil Sie sonst keinen Einfluss darauf haben, was andere Nutzer versuchen könnten, während Sie an der Datei arbeiten.

Tabelle 66. *Gültige Schlüsselwörter für den Dateischutz.*

Lock-Modus	Beschreibung
Lock Read	Andere können auf die geöffnete Datei nicht lesend, aber schreibend zugreifen.
Lock Write	Andere können auf die geöffnete Datei nicht schreibend, aber lesend zugreifen.
Lock Read Write	Andere können auf die geöffnete Datei weder lesend noch schreibend zugreifen.

Mit dem Schlüsselwort Len bestimmen Sie die Datensatzlänge, wenn eine Datei „For Random“ geöffnet wird (s. weiter unten).

8.7. Informationen über geöffnete Dateien

Basic hat Funktionen, mit denen Sie Informationen über Dateien mit Hilfe der Dateinamen erhalten, s. Listing 161. Sie können auch Informationen über geöffnete Dateien mit Hilfe der Dateinummer erhalten. Der Rückgabewert der Funktion FileAttr zeigt, in welchem Modus die Datei mit der angegeben Dateinummer geöffnet wurde. Tabelle 67 enthält die Rückgabewerte und ihre Bedeutungen.

```
FileAttr(n, 1)  'Wie wurde die Datei in BASIC mit Open For ... geöffnet?
```

Tabelle 67. *Beschreibung der Rückgabewerte von FileAttr().*

Rückgabewert	Beschreibung
1	Geöffnet „For Input“
2	Geöffnet „For Output“
4	Geöffnet „For Random“
8	Geöffnet „For Append“
16	Geöffnet „For Binary“
32	Fehler in der Dokumentation: steht nicht für geöffnet „For Binary“

Achtung Die Dokumentation für FileAttr ist fehlerhaft. FileAttr(n, 2) gibt nicht die Dateinummer zurück. Wenn das Argument ungleich 1 ist, wird immer 0 geliefert. Ein weiterer Fehler: Binary ergibt nicht den Rückgabewert 32.

Mit der Funktion EOF prüfen Sie, ob das Dateiende erreicht ist. Sie brauchen diese Funktion, wenn Sie den Dateiinhalt bis zum „End of File“ lesen wollen:

```
n = FreeFile                    'Immer nötig. Nächste freie Dateinumer
Open FileName For Input As #n   'Datei zum Lesezugriff öffnen
Do While Not EOF(n)             'Solange Not End Of File
  Input #n, s                   'Daten werden gelesen
```

```
    REM Hier werden die Daten verarbeitet
  Loop
```

Die Funktion LOF liefert die Größe einer geöffneten Datei in Bytes.

```
LOF(n)
```

Die Funktion Loc liefert die aktuelle Position des Dateizeigers. Diese Zahl ist nicht immer korrekt, zudem ist ihre Bedeutung abhängig vom Modus, in dem die Datei geöffnet wurde. Für den Modus Binary liefert Loc die aktuelle Byteposition, für den Modus Random die Nummer des zuletzt gelesenen oder geschriebenen Datensatzes, für den sequenziellen Zugriff mit den Modi Input, Output oder Append jedoch die aktuelle Byteposition dividiert durch 128. Das geschieht aus Kompatibilitätsgründen mit anderen BASIC-Versionen.

```
Loc(n)
```

Tipp Wenn eine Datei in einem Modus anders als Random geöffnet wird und Basic diese Datei als Textdatei ansieht, wird von Loc die als nächste zu lesende Zeilennummer zurückgegeben. Ich kann nicht beurteilen, ob das ein Bug ist oder nur lückenhafte Dokumentation. Manchmal sind die von Loc zurückgegebenen Werte einfach falsch. Wenn Sie zum Beispiel eine Datei „For Output“ öffnen und dann Text hineinschreiben, liefert Loc den Wert 0.

Die Funktion Seek liefert – mit nur einem Argument – die nächste Position zum Lese- oder Schreibzugriff, ähnlich wie die Funktion Loc, mit der Ausnahme, dass es bei sequenziellem Zugriff immer die absolute Byteposition ist. Wenn Sie eine Position in einer Datei sichern wollen, weil Sie später dorthin zurückspringen wollen, holen Sie erst mit der Funktion Seek die aktuelle Position des Dateizeigers und geben dann der Funktion Seek diese Position zum Rücksprung als Argument mit:

```
position = Seek(n)     'Aktuelle Position erhalten und gesichert.
statements             'Hier wird gearbeitet.
Seek(n, position)      'Setzt den Dateizeiger zurück auf die frühere Position.
```

Das Argument zum Setzen des Dateizeigers in der Funktion Seek ist identisch mit dem von Seek zurückgegebenen Wert. Für Random-Dateien ist es die Datensatznummer, nicht die Byteposition. Für sequenzielle Dateien ist es die Byteposition innerhalb der Datei. Das Makro im Listing 163 identifiziert eine geöffnete Datei an der Dateinummer und gibt Informationen über den Öffnungsmodus, die Dateigröße und die Position des Dateizeigers aus. Listing 164 ruft Listing 163 auf. Das Resultat sehen Sie im Bild 62.

Listing 163. *Ausgabe von Informationen über eine geöffnete Datei.*

```
Function GetOpenFileInfo(n As Integer) As String
  Dim s As String
  Dim iAttr As Integer
  On Error GoTo BadFileNumber
  iAttr = FileAttr(n, 1)
  If iAttr = 0 Then
    s = "Dateihandle " & CStr(n) & " ist momentan nicht geöffnet" & Chr$(10)
  Else
    s = "Dateihandle " & CStr(n) & " ist geöffnet im Modus:"
    If (iAttr And 1)  =  1 Then s = s & " Input"
    If (iAttr And 2)  =  2 Then s = s & " Output"
    If (iAttr And 4)  =  4 Then s = s & " Random"
    If (iAttr And 8)  =  8 Then s = s & " Append"
    If (iAttr And 16) = 16 Then s = s & " Binary"
    iAttr = iAttr And Not (1 Or 2 Or 4 Or 8 Or 16)
    If iAttr And Not (1 Or 2 Or 4 Or 8 Or 16) <> 0 Then
      s = s & " nicht unterstütztes Attribut " & CStr(iAttr)
    End If
```

```
    s = s & Chr$(10)
    s = s & "Dateigröße = " & nPrettyFileLen(LOF(n)) & Chr$(10)
    s = s & "Zeigerposition mit Loc = " & CStr(LOC(n)) & Chr$(10)
    s = s & "Zeigerposition mit Seek = " & CStr(Seek(n)) & Chr$(10)
    s = s & "Dateiende (EOF) = " & CStr(EOF(n)) & Chr$(10)
  End If
AllDone:
  On Error GoTo 0
  GetOpenFileInfo = s
  Exit Function
BadFileNumber:
  s = s & "Fehler mit dem Dateihandle " & CStr(n) & Chr$(10) &_
          "Die Datei ist wahrscheinlich nicht geöffnet" & Chr$(10) & Error()
  Resume AllDone
End Function
```

Tipp Das Positionsargument für die Funktion Seek zählt von 1 an, nicht von 0. Das heißt, das erste Byte oder der erste Datensatz ist 1, nicht 0. Seek(n, 1) positioniert zum Beispiel den Dateizeiger auf das erste Byte oder den ersten Datensatz der Datei.

Das Makro im Listing 164 öffnet eine Datei mit Schreibzugriff. Es werden reichlich Daten in die Datei geschrieben, so dass sie auf eine gewisse Größe kommt. Das dauert eine Weile, also haben Sie Geduld. Zu diesem Zeitpunkt gibt die Funktion Loc den Wert 0 zurück und EOF True. Mit Hilfe der Funktion Seek wird der Dateizeiger zu einer Position innerhalb der Datei verschoben, von wo Daten gelesen werden können. Die Funktion Loc gibt immer noch 0 zurück. Um nun den Rückgabewert von Loc zu beeinflussen, werden 100 Datensätze aus der Datei gelesen. Zuletzt wird die Datei von dem Datenträger gelöscht. Bild 62 zeigt Informationen basierend auf dem Dateihandle.

Unter Apples macOS funktioniert das Listing nur, wenn OOo aus einem Terminal heraus gestartet wurde. Es nutzt, wie auch eine Reihe von folgenden Beispielmakros, einen Test, ob CurDir das Wurzelverzeichnis „/“ zurückgibt, und bricht in diesem Fall das Makro ab.

```
Function IsRootDir As Boolean
  If CurDir = "/" Then
    MsgBox "CurDir = '/'" & Chr(10) & _
           "Apple macOS?" & Chr(10) & _
           "Für dieses Makro müssen AOO und LO aus einem Terminal gestartet werden.", _
           16, "Abbruch"
    IsRootDir = True
  End If
End Function
```

Listing 164. *Erstellt WegMitMir.txt im aktuellen Arbeitsverzeichnis und gibt Informationen aus.*

```
Sub WriteExampleGetOpenFileInfo
  Dim FileName As String              'Der Dateiname
  Dim n As Integer                    'Die Dateinummer
  Dim i As Integer                    'Indexvariable
  Dim s As String                     'Temporärer String für Input
  If IsRootDir Then Exit Sub
  FileName = ConvertToURL(CurDir) & "/WegMitMir.txt"
  n = FreeFile()                                    'Nächste freie Dateinummer
  Open FileName For Output Access Read Write As #n  'Geöffnet mit Lese-/Schreibzugriff
  For i = 1 To 15032                                'Schreibt eine Menge Zeugs
    Write #n, "Dies ist Zeile ", CStr(i), "oder", i     'und zwar Text
  Next
  Seek #n, 1022       'Setzt den Dateizeiger auf die Position 1022
  For i = 1 To 100    'Liest 100 Datensätze; dadurch wird Loc gesetzt
    Input #n, s       'Liest einen Datensatz in die Variable s
```

```
    Next
    MsgBox GetOpenFileInfo(n), 0, FileName
    Close #n
    Kill FileName       'Löscht die Datei; wir brauchen sie nicht mehr
End Sub
```

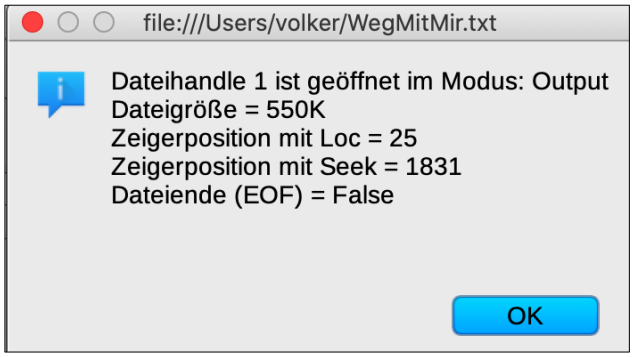


Bild 62. *Informationen basierend auf der Dateinummer.*

8.8. Daten aus einer Datei lesen und in eine Datei schreiben

Um Daten in Dateien zu schreiben oder aus Dateien zu lesen, die im Modus Random oder Binary geöffnet wurden, verwenden Sie die Anweisungen Put und Get. Für jeden anderen Modus verwenden Sie die Anweisungen Line Input, Print und Write. Wenn keine Eingabedaten angegeben sind, wird eine Leerzeile in die Datei geschrieben.

Die Anweisung Write akzeptiert mehrere Argumente zur Eingabe und fügt beim Schreiben automatisch Trennzeichen ein. In der Datei werden die Eingabeausdrücke voneinander durch Kommas getrennt, Zeichenketten werden in doppelte Anführungszeichen gesetzt, Zahlen werden nicht gekennzeichnet, und Datums- sowie boolesche Werte werden zwischen Doppelkreuze (#) gesetzt. Zahlen mit Dezimaltrennzeichen werden gemäß dem eingestellten Gebietsschema konvertiert.

```
Write #n, expression1, expression2, expression3, ...
Print #n, expression1, expression2, expression3, ...
```

Tipp Das Zeichen „#“ hat viele Namen, Doppelkreuz, Gartenzaun, Nummernzeichen, Raute, Hashzeichen, um nur einige zu nennen.

Auch die Anweisung Print kann mehrere Argument erhalten. Sie müssen entweder durch Semikolon oder durch ein Komma voneinander getrennt werden. An einem Semikolon wird kein Trennzeichen eingefügt. Nach einem Komma wird das nächste Argument zum nächsten Tabulator eingerückt. Zahlen werden üblicherweise in 13 Zeichen geschrieben.

```
    Write #n, i, "die Uhrzeit # ist", Now, CDbl(1.221), CBool(0)
    Print #n, i, "die Uhrzeit # ist", Now, CDbl(1.221), Cbool(0)
```

Der obige Code erzeugt den folgenden Text:

```
0,"die Uhrzeit # ist",#14.08.2011 21:05:49#,1,221,#False#
 0            die Uhrzeit # ist14.08.2011 21:05:49 1,221        False
```

Listing 165 zeigt den Unterschied zwischen Write und Print. Im aktuellen Arbeitsverzeichnis wird eine Datei angelegt, die obigen Eingaben werden in die Datei geschrieben, und das Ergebnis wird in einem Dialog ausgegeben. Unter Apples macOS funktioniert das Listing nur, wenn OOo aus einem Terminal heraus gestartet wurde.

Listing 165. *Der Unterschied zwischen Write und Print.*

```
Sub ExampleWriteOrPrint
  Dim FileName As String              'Der Dateiname
  Dim n As Integer                    'Die Dateinummer
  Dim i As Integer                    'Indexvariable
  Dim s As String                     'Temporärer String für Input
  Dim sTemp$
  If IsRootDir Then Exit Sub
  FileName = ConvertToURL(CurDir) & "/WegMitMir.txt"
  n = FreeFile()                                    'Nächste freie Dateinummer
  Open FileName For Output Access Read Write As #n  'Geöffnet mit Lese-/Schreibzugriff

  Write #n, i, "die Uhrzeit # ist", Now, CDbl(1.221), CBool(0)
  Print #n, i, "die Uhrzeit # ist", Now, CDbl(1.221), CBool(0)

  Seek #n, 1          'Setzt den Dateizeiger auf die Position 1
  Line Input #n, s
  Line Input #n, sTemp
  s = s & Chr$(10) & sTemp
  MsgBox s
  Close #n
  Kill FileName       'Löscht die Datei; wir brauchen sie nicht mehr
End Sub
```

Wie der Name schon sagt, liest die Anweisung Line Input eine komplette Textzeile ein (s. Listing 165), lässt aber das Trennzeichen weg. Jede Zeile ist von der nächsten durch einen Zeilenrücklauf (ASCII 13) oder einen Zeilenvorschub (ASCII 10) oder beide getrennt. Auf jedem Betriebssystem, auf dem OOo arbeitet, werden diese Zeilentrenner verwendet.

```
Line Input #n, stringVar     'Liest eine komplette Zeile ohne den Zeilentrenner.
```

Die Anweisung Input liest Text ein, der von folgenden Trennzeichen begrenzt wird: Komma, Zeilenrücklauf oder Zeilenvorschub. Mit einer einzigen Input-Anweisung können Werte in mehrere Variablen unterschiedlichen Typs gelesen werden. Wenn Sie im Listing 165 die Anweisungen „Line Input“ durch „Input“ ersetzen, erhalten Sie nur die ersten beiden Einträge anstatt zwei Zeilen.

```
Input #n, var1, var2, var3, ...
```

Die Anweisung Write fügt automatisch die passenden Trennzeichen ein, so dass man Strings und numerische Daten in die zugehörigen Variablentypen lesen kann. Das Kommando Input entfernt automatisch Kommas und doppelte Anführungszeichen, wenn diese Zeichen als Trenner fungieren. Beispiele dafür finden Sie im Listing 166 mit dem zugehörigen Bild 63. Unter Apples macOS funktioniert das Listing nur, wenn OOo aus einem Terminal heraus gestartet wurde.

Listing 166. *Text mit Input lesen, der mit Write geschrieben wurde.*

```
Sub ExampleInput
  Dim sFileName As String        'Dateiname
  Dim n As Integer
  Dim t As String, d As Double, s As String
  If IsRootDir Then Exit Sub
  sFileName = ConvertToURL(CurDir) & "/WegMitMir.txt"
  n = FreeFile()
  Open sFileName For Output Access Read Write As #n
  Write #n, 1.33, Now, "Ich bin ein String"
  Seek n, 1                      'Zeiger an den Dateianfang
  Input #n, d, t, s
  Close #n
  Kill sFileName
  s = "String (" & s & ")" & Chr$(10) &_
```

```
        "Zahl (" & d & ")" & Chr$(10) &_
        "Zeit   (" & t & ") <== Als String gelesen" & Chr$(10)
    MsgBox s, 0, "Beispiel für Input"
End Sub
```

Der folgenden Ausgabe liegt ein Gebietsschema mit Dezimalpunkt zugrunde, zum Beispiel Englisch. Zur Ausgabe in einem Gebietsschema mit Dezimalkomma, zum Beispiel Deutsch, beachten Sie den anschließenden roten Tipp.

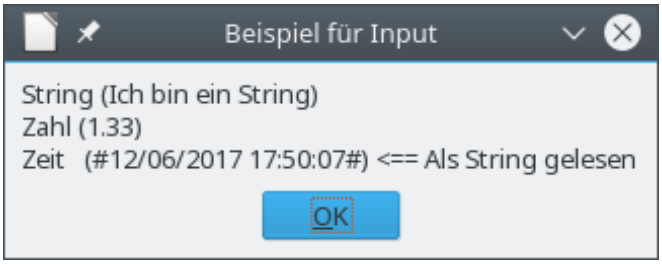
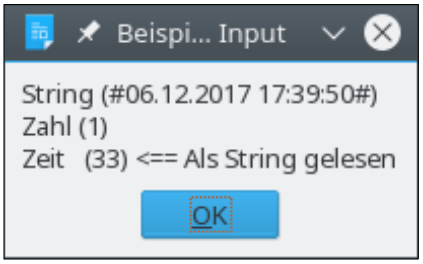


Bild 63. *Input kann # nicht als Trenner von Datums- und Zeitangaben erkennen.*

Achtung

Da die Anweisung Write Dezimalzahlen gemäß dem eingestellten Gebietsschema konvertiert, wird die Zahl 1.33 in der Form 1,33 in die Datei geschrieben, wenn das aktuelle Gebietsschema ein Dezimalkomma vorsieht. Für die Anweisung Input ist das Komma allerdings eine Datengrenze. Die Ausgabe des Listing 166 sieht folglich für das Gebietsschema Deutsch so aus:



In der Datei steht: 1,33,#14.08.2011 14:05:09#,"Ich bin ein String". Input liest bis zum ersten Komma und weist die 1 der Variablen d zu. Die nächste Datengruppe bis zum Komma, 33, wird der Variablen t zugewiesen, und schließlich die dritte Datengruppe bis zum Komma der Variablen s. Der von Write als dritte Datengruppe geschriebene String wird als nun vierte Gruppe ignoriert.

Die Anweisung Input ist für Dateien ungeeignet, in denen Zahlen mit Dezimalkomma vorkommen.

Leider werden die von Write eingefügten Trennzeichen nicht von der Anweisung Input erkannt. Zahlen und einfache Strings werden problemlos gelesen. Zu Zahlen mit Dezimalkommas beachten Sie aber den roten Tipp über diesem Absatz. Datums- und Zeitangaben sowie boolesche Werte, die mit dem Zeichen # eingefasst sind, werden nicht als solche erkannt. Diese Werte müssen in Stringvariablen eingelesen und dann zergliedert werden.

Vermeiden Sie die Anweisung Input, wenn Sie nicht ganz genau wissen, was in der Textdatei steht. Doppelte Anführungszeichen und Kommas werden als Datengrenzen betrachtet. Das Ergebnis kann sein, dass der Text beim Auslesen nicht korrekt zergliedert ist. Wenn Ihre Daten diese Zeichen enthalten, verwenden Sie die Anweisung Line Input und zergliedern Sie den Text gezielt. Falls Sie auch die Zeilentrenner einlesen müssen, sollten Sie die Datei im Modus Binary öffnen.

Eine Binärdatei ist eine Datei mit wahlfreiem Zugriff und der Datensatzlänge null. Zum Schreiben in eine Datei im Modus Random oder Binary verwenden Sie die Anweisung Put. Im einfachsten Fall fügen Sie eine einfache Variable direkt in die Datei ein:

```
int_var = 4 : long_var = 2
Put #n,,int_var    '04 00       (es werden zwei Bytes geschrieben)
Put #n,,long_var   '02 00 00 00 (es werden vier Bytes geschrieben)
```

Das erste Argument ist die Dateinummer, und das dritte Argument ist der zu schreibende Wert. Das zweite Argument ist die Position innerhalb der Datei, an der die Daten eingefügt werden sollen. Wenn Sie wie im Beispiel diese Positionsangabe weglassen, müssen Sie dennoch das Komma setzen.

```
Put #n,,variable             'Schreibt an der nächsten Datensatz- oder Byteposition
Put #n, position, variable   'Spezifiziert die nächste Datensatz- oder Byteposition
```

Im Modus Random wird die Position als Datensatznummer interpretiert, im Modus Binary als absolute Byteposition. Ohne Angabe einer Position werden die Daten am aktuellen Dateizeiger eingefügt, der dann mit den geschriebenen Daten weiter wandert.

Wenn die Datenvariable vom Typ Variant ist, wird den Daten eine den Datentyp beschreibende Integer-Zahl vorangesetzt. Es ist dieselbe Zahl, die auch von der Funktion VarType, wie später beschrieben, zurückgegeben wird.

```
v = 4            'Eine Variantvariable
Put #n,,v        '02 00 04 00  (die ersten beiden Bytes beschreiben den Typ als 2)
Put #n,,4        '02 00 04 00  (die ersten beiden Bytes beschreiben den Typ als 2)
Put #n,,CInt(4)  '02 00 04 00  (die ersten beiden Bytes beschreiben den Typ als 2)
```

Einem als Variant gespeicherten String wird der VarType vorangesetzt, wenn er mit Put in eine Datei geschrieben wird, die in irgendeinem Modus außer Binary geöffnet wurde. Wenn Put ein Array schreibt, wird jedes Element des Arrays geschrieben. Wenn das Array einen String-Variant enthält, wird der VarType auch dann geschrieben, wenn der Dateimodus Binary ist. Wenn Put einen String als Variant einfügt, wird konkret der VarType, die Stringlänge und dann der String geschrieben.

```
v() = Array("ABCD") 'ASCII ist hexadezimal 41 42 43 44
Put #n,,v()         '08 00 04 00 41 42 43 44 (08 00 = Typ)(04 00 = Länge)
```

Wenn Put Daten in eine Datei schreibt, wird erst der aktuelle Dateizeiger gesichert, danach werden alle Daten geschrieben. Falls die Blocklänge größer ist als null, wird der Dateizeiger eine Blocklänge hinter die zuletzt gesicherte Position verschoben, ohne Rücksicht darauf, wie viele Bytes tatsächlich geschrieben wurden. Wenn zum Beispiel die Blocklänge 32 ist und die aktuelle Zeigerposition 64, wird der Dateizeiger nach dem Einfügen der Daten auf das Byte 96 gesetzt. Wenn mehr als 32 Bytes geschrieben wurden, wird ein Teil des nächsten Datensatzes überschrieben. Wurden weniger als 32 Bytes geschrieben, bleibt ein Teil des vorherigen Datensatzes unverändert. Aus diesem Grunde initialisieren manche Leute jeden zu schreibenden Datensatz, wenn die Datei erstellt wird. Numerische Werte werden gewöhnlich zu null initialisiert, String-Variablen zu Leerzeichen.

Andererseits können Sie – auch wenn ich es nicht empfehlen kann – die Anweisung Put auf eine Datei anwenden, die im sequenziellen Modus geöffnet ist. Gleichermaßen können Sie die Anweisungen Write, Line Input und Input auf Dateien anwenden, die im Modus Binary oder Random geöffnet sind. Welche Bytes tatsächlich von den Schreibmethoden in Dateien mit der „falschen“ Struktur geschrieben werden, ist nicht dokumentiert, und das Ergebnis war schwierig zu analysieren. Schließlich habe ich den Quellcode gelesen, um festzustellen, was in jedem einzelnen Fall geschrieben wird, aber ein auf diesem Wege ermitteltes undokumentiertes Basic-Verhalten sollte man nicht als stabil und verlässlich ansehen. Wenn Sie diese Methoden für andere als die dokumentierten Dateistrukturen verwenden wollen, rate ich Ihnen, zuvor die Ergebnisse mit Ihren spezifischen Daten zu testen. Wenn ein Datenteil in eine Datei geschrieben wird, bestimmt der spezifische Kontext, was konkret eingefügt wird, s. Tabelle 68.

Tabelle 68. *Überblick über die von der Anweisung Put geschriebenen Daten.*

Typ	Bytes	Kommentar
Boolean	1	Basic speichert einen booleschen Wert in einem Integer. Im Falle True sind alle Bits gesetzt. Aber da der Wert zu einem Byte herunter konvertiert wird, entsteht ein Laufzeitfehler. Der Wert False wird problemlos geschrieben.

Typ	Bytes	Kommentar
Byte	3	Obwohl Byte-Variablen nur mit einem Byte geschrieben werden, sind sie nur als Variant verfügbar, so dass dem Wert zwei Bytes Typinformation vorangehen.
Currency	8	Intern als Double gespeichert.
Date	8	Intern als Double gespeichert.
Double	8	
Integer	2	
Long	4	
Object	Error	Laufzeitfehler. Es werden nur die Basistypen unterstützt.
Single	4	
String	Len(s)	Jedes Zeichen besteht aus einem Byte. Zeichen mit einem ASCII-Wert größer als 255 werden unvollständig geschrieben. Im Modus Binary schreibt die Anweisung Put keine Zeichen mit dem ASCII-Wert null. Im Modus Random funktioniert das aber, und dem String wird die Stringlänge vorangesetzt.
Variant	Unterschiedlich	Zuerst wird die Typinformation in zwei Bytes geschrieben, danach die Daten. Ein String enthält auch die Stringlänge in zwei Bytes.
Empty	4	Für eine leere Variant-Variable werden zwei Bytes als Typinformation für einen Integer-Wert geschrieben, danach zwei Null-Bytes.
Null	Error	Nur ein Objekt kann den Wert Null haben. Die Anweisung Put arbeitet nicht mit Objekten.

Tipp Get und Put arbeiten in Basic nur mit den Standard-Datentypen.

Tipp Zahlen werden in Binärdateien in umgekehrter Bytefolge geschrieben.

Achtung Die Anweisung Put kann den booleschen Wert True nicht schreiben. Außerdem schreibt sie unkorrekte Strings mit Unicode-Werten größer als 255.

Achtung Die Anweisung Get versagt, wenn die Position in Binärdateien nicht angegeben ist.

Mit Get werden Daten aus Dateien im Modus Binary und Random gelesen. Die Syntax ist der der Anweisung Put ähnlich.

```
Get #n,,variable              'Liest von der nächsten Datensatz- oder Byteposition
Get #n, position, variable    'Spezifiziert die nächste Datensatz- oder Byteposition
```

Wenn das Argument zur Anweisung Get eine Variable vom Typ Variant ist, wird die Typinformation immer gelesen, unabhängig vom Kontext. Wird ein String gelesen, wird vorausgesetzt, dass an seinem Anfang ein Integer-Wert die Stringlänge anzeigt. Diese benötigte Stringlänge wird nicht automatisch in Dateien im Modus Binary geschrieben, wohl aber im Modus Random. Listing 167 zeigt ein Beispiel, wie in eine Binärdatei geschrieben und aus ihr gelesen wird, s. auch Bild 64. Unter Apples macOS funktioniert das Listing nur, wenn OOo aus einem Terminal heraus gestartet wurde.

Listing 167. *Erstellt eine Binärdatei und liest Daten daraus.*

```
Sub ExampleReadWriteBinaryFile
  Dim sFileName As String 'Name der Datei
  Dim n As Integer        'Dateinummer
  Dim i As Integer        'Provisorische Integer-Variable
  Dim l As Long           'Provisorische Long-Variable
  Dim s As String         'Provisorische String-Variable
  Dim s2 As String        'Noch eine provisorische String-Variable
  Dim v                   'Provisorische Variant-Variable
```

```
  If IsRootDir Then Exit Sub
  sFileName = ConvertToURL(CurDir) & "/WegMitMir.txt"
  If FileExists(sFileName) Then
    Kill sFileName
  End If
  n = FreeFile()
  Open sFileName For Binary As #n
  i = 10  : Put #n, , i       '0A 00
  i = 255 : Put #n, , i       'FF 00
  i = -2  : Put #n, , i       'FE FF
  l = 10  : Put #n, , l       '0A 00 00 00
  l = 255 : Put #n, , l       'FF 00 00 00
  l = -2  : Put #n, , l       'FE FF FF FF

  REM Schreibt Stringdaten mit Put, setzt ihnen die Länge voran
  i = 8   : Put #n, , i       '08 00 (es sollen 8 Zeichen geschrieben werden)
  s = "ABCD"
  Put #n, , s                 '41 42 43 44 (ASCII für ABCD)
  Put #n, , s                 '41 42 43 44 (ASCII für ABCD)

  REM Schreibt Daten mit Put, die aus einer Variant-Variablen stammen
  Put #n, , CInt(10)          '02 00 0A 00
  i = -2 : Put #n, , CInt(i) '02 00 FE FF       (Funktionen geben Typ Variant zurück)
  Put #n, , CLng(255)         '03 00 FF 00 00 00 (Funktionen geben Typ Variant zurück)
  v = 255  : Put #n, , v      '02 00 FF 00       (Dies IST ein Variant)
  v = "ABCD" : Put #n, , v    '41 42 43 44       (Ist nicht Element eines Arrays)
  v = Array(255, "ABCDE")     'Der String enthält Typinformation und Länge
  Put #n, , v()               '02 00 FF 00 08 00 05 00 41 42 43 44 45
  Close #n

  REM Nun lesen wir die Datei
  s = ""
  n = FreeFile()
  Open sFileName For Binary Access Read As #n
  Get #n,  1, i  : s = s & "Gelesen: Integer " & i & Chr$(10)
  Get #n,  3, i  : s = s & "Gelesen: Integer " & i & Chr$(10)
  Get #n,  5, i  : s = s & "Gelesen: Integer " & i & Chr$(10)
  Get #n,  7, l  : s = s & "Gelesen: Long " & l & Chr$(10)
  Get #n, 11, l  : s = s & "Gelesen: Long " & l & Chr$(10)
  Get #n, 15, l  : s = s & "Gelesen: Long " & l & Chr$(10)
  Get #n, 19, s2 : s = s & "Gelesen: String " & s2 & Chr$(10)
  Close #n
  MsgBox s, 0, "Read Write mit Binärdatei"
  Kill sFileName       'Löscht die Datei; wir brauchen sie nicht mehr
End Sub
```

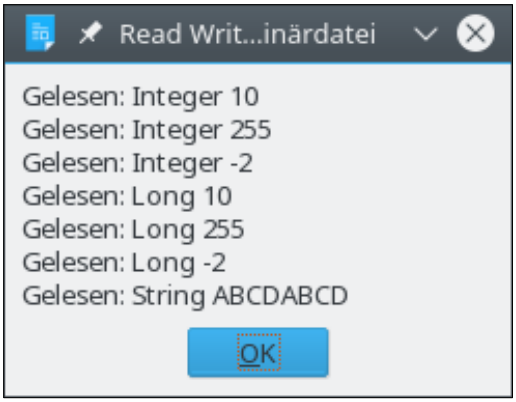


Bild 64. *Mit Get wird aus Binärdateien gelesen, mit Put wird in sie geschrieben.*

Dateien im Modus Random werden gerne zum Speichern benutzerdefinierter Datentypen verwendet, was leider nicht von Basic unterstützt wird. Zum wahlfreien Zugriff öffnen Sie eine Datei mit „Open DateiName For Random“. Der Code im Listing 168 schreibt mehrere Dateitypen und -größen mit der Blocklänge 8 in die Datei. Wenn die Anweisung Put keinen Bug hätte, würde der Dateizeiger, nachdem der erste Block geschrieben ist, auf den zweiten Block positioniert. Stattdessen aber wird er ans Dateiende verschoben. Diesen Bug vermeiden Sie, wenn Sie in jeder Put-Anweisung explizit die Position angeben. Wenn die Position auf einen Punkt jenseits des Dateiendes zeigt, wird der Dateizeiger an das Ende gesetzt. Aus diesem Grunde initialisiert der Code im Listing 168 die Datei vor dem Start zu lauter Nullen. Es liefert Positionen für die nachfolgenden Aktionen. Beachten Sie, dass vor dem Schreiben des Strings die Stringlänge eingefügt wird. Die Ausgabe ist im Wesentlichen identisch mit der im Bild 64 aus dem Listing 167. Unter Apples macOS funktioniert das Listing nur, wenn OOo aus einem Terminal heraus gestartet wurde.

Listing 168. *Schreiben und Lesen mit einer im Modus Random geöffneten Datei.*

```
Sub ExampleReadWriteRandomFile
  Dim sFileName As String 'Name der Datei
  Dim n As Integer        'Dateinummer
  Dim i As Integer        'Provisorische Integer-Variable
  Dim l As Long           'Provisorische Long-Variable
  Dim s As String         'Provisorische String-Variable
  Dim s2 As String        'Noch eine provisorische String-Variable
  If IsRootDir Then Exit Sub
  sFileName = ConvertToURL(CurDir) & "/WegMitMir.txt"

  REM Zuerst muss die Datei initialisiert werden!
  REM Soll die Datei nicht neu angelegt werden, muss man Access Read nutzen.
  REM Ich kann nicht den Modus Binary nehmen, weil dann
  REM keine ASCII-Nullen geschrieben werden.
  n = FreeFile()
  Open sFileName For Random As #n Len = 8
  REM Zuerst wird eine Datei mit reichlich Nullen erstellt.
  REM Ausreichend für 20 Datensätze mit je 8 Bytes.
  s = String(8 * 20 - 2, 0)  'String aus 158 Zeichen mit dem ASCII-Wert 0
  Put #n, 1, s               'Als Random geschrieben, daher wird Len(s) vorangesetzt.
  i = 0 : Put #n, 1, i       'Die Längenangabe wird mit Nullen überschrieben.

  REM Und nun die Daten.
  i = 10  : Put #n, 1, i      '0A 00
  i = 255 : Put #n, 2, i      'FF 00
  i = -2  : Put #n, 3, i      'FE FF
  l = 10  : Put #n, 4, l      '0A 00 00 00
```

```
    l = 255 : Put #n, 5, l       'FF 00 00 00
    l = -2  : Put #n, 6, l       'FE FF FF FF

    REM Jetzt der String. Die Länge (Integer) wird automatisch eingefügt.
    s = "ABCD" : Put #n, 7, s  '04 00 41 42 43 44 (Erst die Länge, dann die ASCII-Werte).
    Close #n
    REM Die Datei wird nun gelesen.
    s = ""
    n = FreeFile()
    Open sFileName For Random Access Read As #n Len = 8
    Get #n, 1, i  : s = s & "Gelesen: Integer " & i & Chr$(10)
    Get #n, 2, i  : s = s & "Gelesen: Integer " & i & Chr$(10)
    Get #n, 3, i  : s = s & "Gelesen: Integer " & i & Chr$(10)
    Get #n, 4, l  : s = s & "Gelesen: Long " & l & Chr$(10)
    Get #n, 5, l  : s = s & "Gelesen: Long " & l & Chr$(10)
    Get #n, 6, l  : s = s & "Gelesen: Long " & l & Chr$(10)
    Get #n, 7, s2 : s = s & "Gelesen: String " & s2 & Chr$(10)
    Close #n
    MsgBox s, 0, "Read Write mit Random-Datei"
    Kill sFileName       'Löscht die Datei; wir brauchen sie nicht mehr
End Sub
```

8.9. Fazit

Mit den Datei- und Verzeichnisfunktionen in Basic kann man Verzeichnisse und Dateien manipulieren. Mit Ausnahme der Lese- und Schreibvorgänge in Binary- und Random-Dateien arbeiten die Funktionen zur Manipulation von Dateien und Verzeichnissen ohne große Überraschungen. Andererseits sind manche der Funktionen beschädigt, und das schon seit Jahren. Sie werden fortgeschrittenere Methoden benötigen, wie Streams und SimpleFileAccess, für alles, was über das Lesen und Schreiben einfacher Dateien hinausgeht (s. Abschnitt 10.12. Services für Dateien und Verzeichnisse).

9. Diverse weitere Routinen

Dieses Kapitel behandelt die Basic-Subroutinen und -Funktionen, die nicht gut in eine größere Kategorie passen – das sind zum Beispiel Routinen zum Programmfluss, zur Dateneingabe durch den Nutzer, zur Datenausgabe auf dem Bildschirm, zur Fehlerbehandlung, zu Informationen über Variablen, zur Farbe und zur Bildschirmdarstellung. Es sind **auch ein paar** Funktionen dabei, die Sie nicht verwenden sollten.

Ich war zuerst geneigt, dieses Kapitel „Sonstiges“ zu nennen, weil es Routinen enthält, die übrig geblieben sind, nachdem ich die anderen in Kapitel gruppiert hatte. Auch wenn das Wort „Sonstiges“ oft einen abschätzigen Beigeschmack hat, so gilt das keineswegs für die Routinen dieses Kapitels. Der eklektische Mix enthält einige sehr interessante und nützliche Routinen, deren Vielfalt keine einschläfernde Langeweile aufkommen lassen wird.

9.1. Bildschirm und Farbe

In der Tabelle 69 finden Sie die Basic-Funktionen zur Ermittlung der Bildschirmmaße und zur Farbdarstellung. Die Bildschirmmaße beziehen sich auf die generelle Pixelgröße, so dass Sie Makros schreiben können, die Objekte in einer bestimmten Größe erstellen und präzise positionieren können.

Tabelle 69. *Basic-Funktionen, die sich auf den Bildschirm und die Farben beziehen.*

Funktion	Beschreibung
Blue(Farbwert)	Ermittelt den Blauanteil.
GetGuiType	Ermittelt den GUI-Typ: Mac, Windows, Unix.
Green(Farbwert)	Ermittelt den Grünanteil.
QBColor(DOS_Farbwert)	Gibt RGB für Standardfarben zurück.
Red(Farbwert)	Ermittelt den Rotanteil.
RGB(Rot, Grün, Blau)	Rechnet RGB in eine Farbnummer um.
TwipsPerPixelX	Pixelbreite in Twips.
TwipsPerPixelY	Pixelhöhe in Twips.

9.1.1. Bestimmung des GUI-Typs

Die Funktion GetGuiType gibt einen Integerwert zurück, der sich auf die grafische Benutzeroberfläche (GUI) bezieht. Damit können Sie herausfinden, auf welch einem Rechner das Makro läuft ... na ja, irgendwie so. Diese Funktion nennt nur den GUI-Typ, nicht das Betriebssystem – zum Beispiel nur Windows, nicht Windows 98 oder Windows XP. Die Funktion ist nur zur Kompatibilität mit früheren Versionen von Basic enthalten.

Einer meiner Kollegen betrieb OOo als Server auf seinem Heimrechner. Von seinem Arbeitsplatz verband es sich als Client zu seinem Heimrechner. Der von GetGuiType zurückgegebene Wert war unbestimmt, wenn OOo in einer Client-Server-Umgebung läuft.

Tabelle 70 zeigt die Rückgabewerte, einerseits aus den OOo-Hilfetexten, andererseits aus dem Quellcode der Version 3.2.1.

Tabelle 70. *Rückgabewerte von GetGuiType.*

#	OOo-Hilfetext	Quellcode
1	Windows	Windows (manchmal OS/2, unter dem Windows läuft)
2	Nicht beschrieben	OS/2
3	Nicht beschrieben	War früher mal Mac, wird nicht zurückgegeben

#	OOo-Hilfetext	Quellcode
4	Unix	Unix, also auch Mac
-1	Als nicht definiert beschrieben	Nicht unterstütztes OS

Das Makro im Listing 169 demonstriert die Funktion GetGuiType.

Listing 169. *Gibt den GUI-Typ als String aus.*

```
Sub DisplayGUIType()
  Dim s As String
  Select Case GetGUIType()
    Case 1
      s = "Windows"   ' Dokumentiert und im Code.
    Case 2
      s = "OS/2"      ' Nicht mehr dokumentiert und nicht im Code.
    Case 3
      s = "Mac OS"    ' Nicht mehr dokumentiert und nicht im Code.
                      ' Für macOS wird 4 zurückgegeben (LO).
    Case 4            ' Dokumentiert und im Code.
      s = "UNIX"
    Case Else         ' -1 : Nicht dokumentiert, steht aber so im Code.
      s = "Unbekannter Wert " & CStr(GetGUIType()) & Chr$(10) &_
          "Möglicherweise im Client-Server-Modus"
  End Select
  MsgBox "Der GUI-Typ ist " & s, 0, "GetGUIType()"
End Sub
```

Der Wert -1 wird dann zurückgegeben, wenn der Typ unbekannt ist. Das ist aber nicht ausdrücklich dokumentiert. Möglicherweise bedeutet das, dass Sie sich im Client-Server-Modus befinden, aber ich habe das nicht geprüft.

9.1.2. Ermittlung der Pixelgröße (in Twips)

Basic hat zwei Funktionen zur Ermittlung der Pixelgröße (Dot) in Twips: TwipsPerPixelX und TwipsPerPixelY. Das Wort Twip ist eine Abkürzung für „Zwanzigstel eines PostScript-Punkts (twentieth of a PostScript point)". Auf einen Zoll gehen 72 PostScript-Punkte, also 1440 Twips pro Zoll.

Im Jahre 1886 hat die American Typefounders Association ein Maß für den Schriftsatz mit dem Namen „American Printer's Point" eingeführt. Auf einen Zoll gehen ungefähr 72,27000072 Printer's Points. Jahre später haben Jim Warnock und Charles Geschke bei der Entwicklung der Seitenbeschreibungssprache PostScript für Adobe Systems den PostScript-Punkt auf genau 72 Punkt pro Zoll festgelegt. Als die Nadeldrucker auf den Markt kamen, konnten sie mit jeweils 10 oder 12 Zeichen pro Zoll drucken. Als Maßeinheit, die sowohl für Nadeldrucker als auch für PostScript-Punkte gleichermaßen praktisch war, wurden Twips erfunden.

Tipp Auf einen Zoll gehen 1440 Twips. Diese Zahl ist deswegen wichtig, weil OOo für viele Maße Twips verwendet.

Alle Grafikroutinen von Microsoft Windows basieren auf dem Standard Twips. Im Rich Text Format, in Druckertreibern, in Bildschirmtreibern und vielen anderen Produkten und Plattformen werden Twips verwendet – inklusive OOo. Das Makro im Listing 170 ermittelt die Anzahl der Twips pro Pixel jeweils in der X- und Y-Richtung (horizontal und vertikal) und gibt die Anzahl als Pixel pro Zoll aus.

Listing 170. *Ermittlung der Pixel pro Zoll.*

```
Sub DisplayPixelSize
  Dim s As String
  s = s & TwipsPerPixelX() & " Twips pro X-Pixel oder " &_
      CStr(1440 \ TwipsPerPixelX()) & " X-Pixels pro Zoll" & Chr$(10)
  s = s & TwipsPerPixelY() & " Twips pro Y-Pixel oder " &_
      CStr(1440 \ TwipsPerPixelY()) & " Y-Pixels pro Zoll"
  MsgBox s, 0, "Pixel-Größe"
End Sub
```

Bild 65. *Pixels pro Zoll auf meinem Rechner.*

Leider ist mir nicht klar, was bei mehreren Monitoren mit unterschiedlichen Werten zurückgegeben wird. Laut Quellcode sind es Werte vom „Standardgerät“.

9.1.3. Der Gebrauch der Farbfunktionen

Auf Computermonitoren, Digitalkameras, Scannern – und auch im menschlichen Auge – werden Farben durch die Addition der drei primären Lichtfarben erzeugt: Rot, Grün und Blau (RGB). Beim Drucken oder Malen wird die Farbe dadurch erzeugt, dass manche Farben absorbiert und andere reflektiert werden. Farbdrucker verwenden einen anderen Farbsatz, die Primärpigmentfarben: Zyan, Magenta, Gelb und Schwarz (CMYK für die englischen Bezeichnungen Cyan, Magenta, Yellow und Key bzw. BlacK). Diese beiden verschiedenen Systeme basieren auf realen physikalischen Modellen. Das RGB-Modell basiert darauf, wie sich Licht zu Farben mischt. Das CMYK-Modell basiert darauf, was beim Mischen von Farbstoffen verschiedener Farben geschieht.

Basic verwendet das RGB-Modell mit 256 verschiedenen Abstufungen in jeder Primärfarbe. Die Zahl wird als Ganzzahl Long gespeichert. Mit den Funktionen Red, Green und Blue werden entsprechend die roten, grünen und blauen Anteile einer Farbe in OOo extrahiert. Mit der Funktion RGB werden die individuellen Farbanteile kombiniert und ein Long-Integer für OOo erzeugt. RGB akzeptiert drei Argumente, je eines für die Primärfarben. Jeder Farbanteil muss im Bereich 0 bis 255 liegen. Die Funktion RGB verfügt über keine Plausibilitätskontrolle, also sind die Ergebnisse nicht vorhersehbar, wenn Sie die Regeln brechen.

```
Dim nRed As Integer           'Darf nur Werte von 0 bis 255 haben
Dim nGreen As Integer         'Darf nur Werte von 0 bis 255 haben
Dim nBlue As Integer          'Darf nur Werte von 0 bis 255 haben
Dim nOOoColor As Long         'Darf nur Werte von 0 bis 16.581.375 haben
nOOoColor = RGB(128, 3, 101)  '8.389.477
nRed   = Red(nOOoColor)       '128
nGreen = Green(nOOoColor)     '3
nBlue  = Blue(nOOoColor)      '101
```

In den Frühzeiten von DOS arbeitete BASIC mit 16 Farben. In der Tabelle 71 gibt es Spalten für die Farbnamen und eine Spalte für die DOS-Farbe mit der Zahl, die von DOS für diese Farbe verwendet wurde. In der Spalte OOo-Farbe steht die entsprechende von OOo verwendete Zahl. Die Spalten Rot, Grün und Blau enthalten die von den entsprechenden Basic-Funktionen zurückgegebenen Werte. Die

Funktion QBColor dient dazu, mit dem Argument einer DOS-Farbe die dazu gehörende OOo-Farbe zurückzugeben:

```
QBColor(dos_color)
```

Listing 171. *DOS-Farben in OOo.*

```
Sub DisplayQBColor
  Dim i%
  Dim s$
  For i = 0 To 15
    s = s & i & " = " & QBColor(i) & " = ("
    s = s & Red(QBColor(i)) & ", "
    s = s & Green(QBColor(i)) & ", "
    s = s & Blue(QBColor(i)) & ")"
    s = s & Chr$(10)
  Next
  MsgBox s
End Sub
```

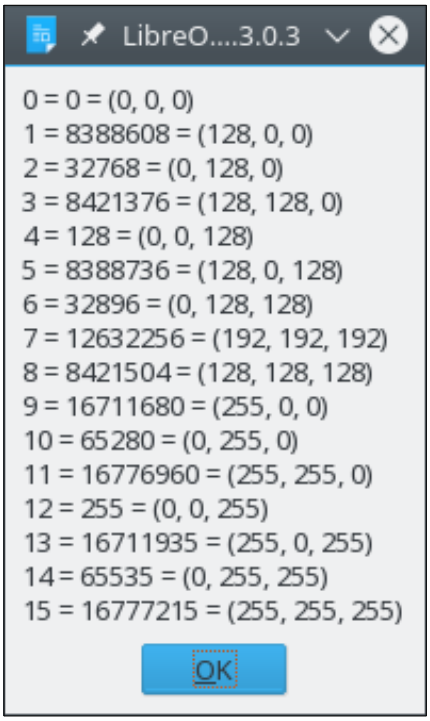


Bild 66. *DOS-Farben in OOo.*

Tabelle 71. *DOS-Farben in OOo.*

DOS-Farbe	OOo-Farbe	Rot	Grün	Blau
0	0	0	0	0
4	128	0	0	128
2	32768	0	128	0
6	32896	0	128	128
1	8388608	128	0	0
5	8388736	128	0	128
3	8421376	128	128	0
8	8421504	128	128	128
7	12632256	192	192	192
12	255	0	0	255

DOS-Farbe	OOo-Farbe	Rot	Grün	Blau
10	65280	0	255	0
14	65535	0	255	255
9	16711680	255	0	0
13	16711935	255	0	255
11	16776960	255	255	0
15	16777215	255	255	255

9.2. Makroausführung verzögern und abbrechen

Tabelle 72. *Basic-Funktionen zum Abbruch und zur Verzögerung der Makroausführung.*

Funktion	Beschreibung
Stop, End	Hält die Makroausführung sofort an.
Wait(Millisekunden)	Unterbricht die Makroausführung für eine kurze Zeitdauer.
WaitUntil(eineZeit)	Unterbricht die Makroausführung bis zu einem Zeitpunkt.

Der Befehl Stop bzw. End bricht die Ausführung des Makros ab. Ende, Schluss, Finito! Sie müssen es neu starten. Die Anweisung Wait unterbricht jedoch nur die Ausführung (s. Listing 172). Nach der angegebenen Dauer (in Millisekunden) läuft das Makro wieder weiter. Ein negativer Wert für die Millisekunden erzeugt einen Laufzeitfehler.

Listing 172. *Beispiel für die Funktion Wait.*

```
Sub ExampleWait
  On Error Goto BadInput      'Falsche Eingabe
  Dim nMillis As Long
  Dim nStart As Long
  Dim nEnd As Long
  Dim nElapsed As Long        'Verstrichene Zeit
  nMillis = CLng(InputBox("Wie viele Millisekunden Pause?"))
  nStart = GetSystemTicks()
  Wait(nMillis)
  nEnd = GetSystemTicks()
  nElapsed = nEnd - nStart
  MsgBox "Sie wollten ein Pause von " & nMillis & " Millisekunden." &_
     Chr$(10) & "Die Pause dauerte " & nElapsed & " Millisekunden.", 0, _
     "Beispiel für Wait"
BadInput:
  If Err <> 0 Then
    Print Error() & " in der Zeile " & Erl
  End If
  On Error Goto 0
End Sub
```

In all meinen Experimenten funktionierte die Anweisung Wait völlig korrekt. Das Makro pausiert und startet wieder zur angegebenen Zeit.

WaitUntil ist neu und bietet Kompatibilität mit VB. Der folgende Code bewirkt eine Pause von zwei Sekunden:

```
WaitUntil Now + TimeValue("00:00:02")
```

9.3. Externe Anwendungen

In der Tabelle 73 sind die Funktionen aufgelistet, die Sie benötigen, um Prozeduren aus Laufzeitbibliotheken (DLL) aufzurufen, um Befehle über die Kommandozeile (Shell) auszuführen, und um den dynamischen Datenaustausch (DDE) zwischen verschiedenen Programmen zu organisieren.

Tabelle 73. *Basic-Funktionen für externe Anwendungen.*

Funktion	Beschreibung
Declare	Deklariert eine DLL-Bibliothek, die Sie aufrufen wollen.
FreeLibrary(BibName)	Gibt eine DLL-Bibliothek frei.
Shell	Führt einen Befehl über die Kommandozeile aus.

9.3.1. Laufzeitbibliotheken (DLL = Dynamic Link Libraries)

Während seiner Ausführung kann ein Makro zahlreiche Subroutinen und Funktionen direkt aufrufen. Es kann sogar Routinen und Anwendungen aufrufen, die nichts mit OOo zu tun haben. Eine Laufzeitbibliothek (DLL) ist eine Sammlung von Routinen oder Programmen, die bei Bedarf aufgerufen werden können. Jede DLL ist ein Paket in einer Datei mit dem Namenssuffix „DLL“ – das Suffix „.dll“ ist nicht immer erforderlich, aber weitestgehend üblich. DLL-Dateien haben zwei feine Eigenschaften: Viele Programme können sich eine DLL teilen, und sie werden erst in den Arbeitsspeicher geladen, wenn sie gebraucht werden. Die Verwendung von DLLs fördert die Wiederverwendung von Programmcode und spart Arbeitsspeicher. Um Basic über eine Routine in einer DLL zu informieren, verwenden Sie die Anweisung Declare.

Achtung DLLs werden unter Linux und macOS nicht unterstützt.

```
Declare Sub Name Lib "LibName" (arguments)
Declare Function Name Lib "LibName" (arguments) As Type
```

LibName ist der Name der DLL, in der es die Routine mit dem Namen Name gibt. Normalerweise nutzen Sie eine DLL, die Sie nicht selber geschrieben haben. Sie werden den Namen der aufzurufenden Routine also nicht beeinflussen können. Die Namensgebung kann zum Problem werden, wenn Ihr Makro schon eine Routine mit demselben Namen enthält oder wenn Sie eine weitere Routine mit demselben Namen in einer anderen DLL aufrufen. Die Lösung des Problems besteht in dem Schlüsselwort „Alias“, das Sie der Declare-Anweisung hinzufügen können. In folgenden Fall ist RealName der in der DLL verwendete Name, und myName wird von Ihrem Makro als Name benutzt.

```
Declare Sub myName Lib "LibName" Alias "RealName" (arguments)
Declare Function myName Lib "Libname" Alias "RealName" (arguments) As Type
```

Für Funktionen sollte die Typdeklarierung nur Standardtypen verwenden. Sie müssen selbstverständlich wissen, welchen Typ Sie für die Deklarierung brauchen. Die Argumentliste enthält die Argumente, die an die externe Routine übergeben werden. Falls Sie ein Argument mit seinem Wert und nicht als Referenz angeben, müssen Sie das mit dem Schlüsselwort ByVal tun. Listing 173 ruft eine DLL auf.

Tipp Basic übergibt Argumente standardmäßig als Referenz. Das heißt, dass ein Argument, das von der aufgerufenen Routine geändert wird, anschließend auch in dem aufrufenden Programmteil geändert ist. Das Schlüsselwort ByVal bewirkt, dass ein Argument als Wert und nicht als Referenz übergeben wird.

Listing 173. *Ruft eine DLL auf. Nur unter Windows (und wenn die DLL vorhanden ist).*

```
Declare Sub MyMessageBeep Lib "user32.dll" _
                          Alias "MessageBeep" (Long)
Declare Function CharUpper Lib "user32.dll" _
                           Alias "CharUpperA" (lpsz As String)
```

```
Sub ExampleCallDLL
  REM Konvertiert einen String in Großbuchstaben.
  Dim sToBeUpper As String  'In Großbuchstaben umzusetzender String
  Dim sMessage As String    'Text der Meldung

  sToBeUpper  = "ich Habe große und Kleine Buchstaben"
  sMessage = "Konvertiert:" & Chr$(10) & sToBeUpper & Chr$(10)

  CharUpper(sToBeUpper)     'Der String ist nun verändert.

  sMessage = sMessage & "Zu:" & Chr$(10) & sToBeUpper
  MsgBox sMessage, 0, "Aufruf einer DLL-Funktion"

  REM Auf meinem Rechner wird ein Systemklang ausgegeben
  Dim nBeepLen As Long      'Dauer des Tons
  nBeepLen = 5000           '5 Sekunden
  MyMessageBeep(nBeepLen)   'Piiieps oder Bling
  FreeLibrary("user32.dll") 'Die DLL wird wieder freigegeben.
End Sub
```

Eine DLL wird erst geladen, wenn eine Routine in der DLL aufgerufen wird. Die Anweisung FreeLibrary entfernt die DLL aus dem Arbeitsspeicher. Sie benötigt ein Argument: den Namen der zu entfernenden DLL.

9.3.2. Befehle über die Systemkommandozeile

Die Anweisung Shell ruft eine externe Anwendung auf. Der Befehl steht einem Nutzer nicht zur Verfügung, der über ein virtuelles Portal verbunden ist, es sei denn, er ist derselbe Nutzer, der zuerst OOo gestartet hat. Shell erhält keine Informationen von der Anwendung. Shell führt schlicht und einfach eine andere Anwendung oder einen Befehl aus.

```
Shell(Pfadname, Fensterstil, Param, bSync)
```

Achtung Die Anweisung Shell ist eine potenzielle Sicherheitslücke.

Nur das erste Argument ist obligatorisch, der Rest ist optional. Das erste Argument ist der absolute Pfadname der externen Anwendung. Der Pfad kann in URL-Notation sein, muss es aber nicht. Die Anweisung Shell hat ein Problem mit Leerzeichen im Pfad- oder Anwendungsnamen. Sie können das Problem genauso lösen, wie es der Webbrowser macht: Ersetzen Sie jedes Leerzeichen durch „%20“. Der ASCII-Wert eines Leerzeichens ist 32, hexadezimal 20. Mit dieser Technik können Sie auch andere Zeichen ersetzen, die womöglich ein Problem darstellen.

```
Shell("file:///C|/Andy/My%20Documents/oo/tmp/h.bat",2) 'Die URL-Notation verwendet /
Shell("C:\Andy\My%20Documents\oo\tmp\h.bat",2)     'Die Windows-Notation verwendet \
```

Das zweite Argument (optional) bestimmt den Fensterstil der aufgerufenen Anwendung. Tabelle 74 listet die gültigen Werte für das Argument Fensterstil auf.

Tabelle 74. *Fensterstil für die Anweisung Shell.*

Stil	Beschreibung
0	Der Fokus liegt auf dem verborgenen Programmfenster.
1	Der Fokus liegt auf dem Programmfenster in der Standardgröße.
2	Der Fokus liegt auf dem minimierten Programmfenster.
3	Der Fokus liegt auf dem maximierten Programmfenster.
4	Das Programmfenster wird in Standardgröße angezeigt, ohne Fokus.

Stil	Beschreibung
6	Das Programmfenster wird minimiert, der Fokus bleibt auf dem aktuellen Fenster.
10	Vollbilddarstellung.

Das dritte Argument (optional) ist ein String, der der Anwendung übergeben wird. Jedes Leerzeichen in diesem String wird von der aufgerufenen Anwendung als Trenner zwischen einzelnen Argumenten betrachtet. Wenn Sie Argumente mit eingebetteten Leerzeichen übergeben wollen, umschließen Sie sie mit einem zusätzlichen Satz doppelter Anführungszeichen.

```
Shell("/home/andy/foo.ksh", 10, """one argument"" another")   ' zwei Argumente
```

Tipp Der String """one argument"" another" ist korrekt und genau so gemeint. Denken Sie darüber nach!

Das letzte, optionale Argument bestimmt, ob die Anweisung Shell die Kontrolle sofort, noch während die externe Anwendung läuft, zurückgibt (das Standardverhalten) oder ob sie abwartet, bis die Anwendung beendet ist. Steht das letzte Argument auf True, wird das Makro auf das Ende der Shell-Anwendung warten.

```
Sub ExampleShell
  Dim rc As Long
  rc = Shell("C:\andy\TSEProWin\g32.exe", 2, "c:\Macro.txt", True)
  Print "Gerade bin ich wieder zurück. Der Rückgabewert ist " & rc   ' rc = 0
  REM Diese beiden Aufrufe haben Leerzeichen in ihren Namen.
  Shell("file:///C|/Andy/My%20Documents/oo/tmp/h.bat", 2)
  Shell("C:\Andy\My%20Documents\oo\tmp\h.bat", 2)
End Sub
```

Die Anweisung Shell gibt einen Long-Integer mit dem Wert null zurück. Wenn das Programm nicht existiert, wird ein Laufzeitfehler ausgelöst und nichts zurückgegeben. Manche Anwendungen geben einen Wert zurück, der als Fehlercode behandelt werden kann. Dieser Wert ist jedoch von Shell nicht zu erhalten. Es leuchtet auch unmittelbar ein, dass man kein Endresultat einer Anwendung erwarten darf, wenn Shell die Kontrolle zurückgibt, bevor die Anwendung beendet ist.

Tipp In Visual Basic sind die Argumente für die Funktion Shell anders: Shell(Pfad, Fensterstil, bsync, Timeout). Der Wert Timeout bestimmt, wie lange auf das Ende der externen Anwendung gewartet wird. Die Argumente zur Anwendung folgen dem Namen im selben String, durch Leerzeichen getrennt.

VB verwendet kein eigenes Argument für die Argumente, die der von Shell gestarteten Anwendung beigegeben werden. Stattdessen folgen die Argumente dem Namen der Anwendung, durch Leerzeichen getrennt, innerhalb desselben Satzes von Anführungszeichen, die den Anwendungsnamen mit Pfad umschließen. Diese Methode funktioniert auch in Basic als Alternative für die Kommandoargumente. Wenn nur die Anwendung mit Argumenten und der Fensterstil erforderlich sind, erlaubt Ihnen dieser alternative Weg, Shell-Kommandos auf identische Weise für VB und StarBasic zu schreiben. Wenn Sie die Argumente bsync oder Timeout benötigen, sind die Umgebungen VB und StarBasic nicht kompatibel.

```
Shell("/home/andy/foo.ksh Hallo zusammen") 'zwei Argumente, "Hallo" und "zusammen"
```

9.4. Dynamischer Datenaustausch

Tipp Alle DDE-Befehle sind veraltet und das schon seit über 10 Jahren. Ich kann daher nur davon abraten, sie zu verwenden (s. https://bz.apache.org/ooo/show_bug.cgi?id=39574).

Dynamischer Datenaustausch (DDE = Dynamic Data Exchange) ist ein Mechanismus zum Datenaustausch zwischen Programmen. Daten können in Echtzeit oder auf Anforderung aktualisiert werden.

DDE ist nur unter Windows verfügbar. Unter Linux wird die Kanalnummer 1 zurückgegeben, aber keine Daten und auch kein Laufzeitfehler.

Obwohl die DDE-Befehle, die von einem DDE-Server entgegengenommen werden, für diesen Server spezifisch sind, ist die generelle Syntax doch immer gleich. Die meisten DDE-Befehle erwarten einen Server, eine Datei und einen Bereich. Der Server ist der DDE-Name der Anwendung, die die Daten bereitstellt. Die Datei, das heißt der Dateiname, bezeichnet den Speicherort des referenzierten Elements. Das Beispiel im Listing 174 verwendet die Funktion DDE in einer Calc-Tabelle, um den Inhalt der Zelle A1 aus einer Excel-Tabelle auszulesen.

Listing 174. *DDE als Calc-Funktion: liest den Inhalt der Zelle A1 aus einem Dokument.*

```
=DDE("soffice";"/home/andy/tstdoc.xls";"a1") 'DDE in Calc zum Zugriff auf eine Zelle
='file:///home/andy/TST.ods'#$Tabelle1.A1    'Direkter Zellbezug
```

In der zweiten Zeile sehen Sie, wie ohne DDE ein direkter Bezug zu einer anderen Calc-Tabelle hergestellt werden kann. Basic unterstützt DDE-Befehle direkt.

Tabelle 75. *Basic-Funktionen für DDE (veraltet).*

Funktion	Beschreibung
DDEExecute(nDDEKanal, Befehl)	Führt einen DDE-Befehl aus.
DDEInitiate(Server, Datei)	Öffnet einen DDE-Kanal.
DDEPoke(nDDEKanal, Bereich, Daten)	Schreibt Daten über den DDE-Kanal auf den Server.
DDERequest(nDDEKanal, Bereich)	Setzt eine DDE-Anforderung über den offenen Kanal ab.
DDETerminate(nDDEKanal)	Schließt den Kanal.
DDETerminateAll()	Schließt alle DDE-Verbindungen.

Zuerst stellt die Funktion DDEInitiate eine Verbindung zum DDE-Server her. Das erste Argument enthält den Namen des Servers – zum Beispiel „soffice“ oder „excel“. Das zweite Argument legt den zu nutzenden Kanal fest. Ein üblicher Wert für einen Kanal ist ein Dateiname. Der geöffnete Kanal wird durch eine Zahl gekennzeichnet, die vom Befehl DDEInitiate zurückgegeben wird. Eine Kanalnummer 0 zeigt an, dass der Kanal nicht geöffnet werden konnte. Der Versuch einer DDE-Verbindung zu einer nicht geöffneten OOo-Datei führt zur Kanalnummer 0, s. Listing 175.

Listing 175. *Zugriff auf ein Calc-Dokument mit DDE.*

```
Sub ExampleDDE
  Dim nDDEChannel As Integer          'Kanalnummer
  Dim s As String
  REM Die Datei muss in OOo geöffnet sein, sonst wird der Kanal nicht geöffnet.
  nDDEChannel = DDEInitiate("soffice", "c:\TST.ods")
  If nDDEChannel = 0 Then
    Print "Entschuldigung, der DDE-Kanal konnte nicht geöffnet werden"
  Else
    Print "Wir nutzen den Kanal " & nDDEChannel & ", um die Zelle A1 zu lesen"
    s = DDERequest(nDDEChannel, "A1")  'Anforderung
    Print "Erhalten: " & s
    DDETerminate(nDDEChannel)          'Schließt den Kanal.
  End If
End Sub
```

Die verfügbaren Kommandos sowie die Syntax sind serverabhängig, eine detaillierte Beschreibung des DDE-Protokolls geht daher über den Rahmen dieses Buches hinaus.

Achtung Beim letzten Test lief Listing 175 und gab einen Wert zurück, dann aber stürzte OOo bei mir ab.

9.5. Benutzereingabe und Bildschirmausgabe

Basic stellt sehr einfache Mechanismen zur Ausgabe von Informationen an den Nutzer und zum Erhalt von Informationen vom Nutzer bereit (s. Tabelle 76). Mit diesen Routinen kann man auf keine Dateien zugreifen, sie liefern einfach Nutzereingaben von der Tastatur oder eine Ausgabe auf dem Bildschirm.

Tabelle 76. *Funktionen zur Nutzereingabe und zu Bildschirmausgaben.*

Funktion	Beschreibung
InputBox(Meldung, Titel, Standard, x_pos, y_pos)	Eingabeaufforderung zu einem String.
MsgBox (Text, Typ, Titel)	Ausgabe einer Meldung in einem gestalteten Dialog.
Print Ausdruck1; Ausdruck2, Ausdruck3;...	Ausgabe einzelner Strings.

9.5.1. Einfache Ausgabe

Die Anweisung Print produziert eine einfache, einzeilige Ausgabe auf dem Bildschirm. Der Anweisung folgt eine Liste von Ausdrücken. Sind diese Ausdrücke durch ein Semikolon getrennt, werden sie in der Ausgabe direkt aneinander gesetzt. Wenn sie durch ein Komma getrennt sind, wird zwischen die Ausdrücke ein Tabulatorschritt gesetzt. Ein Tabulatorschritt ist 4 Leerzeichen lang und kann nicht geändert werden.

```
Print Ausdruck1, Ausdruck2, ...  ' Ausgabe mit Tabs zwischen den Ausdrücken
Print Ausdruck1; Ausdruck2; ...  ' Ausgabe ohne irgendwas zwischen den Ausdrücken
Print 1, Now; "Hallo", "erstmal"; 34.3 ' Wechsel zwischen Komma und Semikolon ist ok
```

Die Argumente werden vor der Ausgabe zu Strings entsprechend dem lokalen Gebietsschema konvertiert. Mit anderen Worten, Datumsangaben und Zahlen erscheinen so, wie Sie es in Ihrer Konfiguration eingestellt haben (**Extras | Optionen | Spracheinstellungen | Sprachen**). Boolesche Werte erscheinen jedoch immer als der Text „True“ oder „False“.

Der OOo-Hilfetext listet zwei spezielle Ausdrücke auf, die mit der Anweisung Print verwendbar sind: Spc und Tab. Die Funktion Spc arbeitet genauso wie die Funktion Space. Sie akzeptiert ein numerisches Argument und gibt einen String aus, der aus lauter Leerzeichen besteht. Auch die Funktion Tab benötigt ein numerisches Argument für die Anzahl der Tabulatorschritte. Diese Funktion steht für AOO leider nicht zur Verfügung. Im Listing 176 finden Sie ein Beispiel für Print.

Achtung Obwohl die Funktion Tab seit der OOo-Version 1 dokumentiert ist, existiert sie in AOO immer noch nicht, jedenfalls bis zur jetzigen Version 4.1.4.

Listing 176. *Ein Beispiel für die Funktionen Spc() und Tab().*

```
Sub ExamplePrint
  On Error Goto AOO
  'AOO löst bei der Tab-Funktion einen Fehler aus.
  Print Tab(0);"LO:";Tab(1);"Datum:";Spc(2);Date();Tab(2);"Uhrzeit:";Spc(2);Time()
  Exit Sub
  AOO:
  Print "Spc(0);AOO:";Spc(4);"Datum:";Spc(2);Date();Spc(8);"Uhrzeit:";Spc(2);Time()
End Sub
```

Die Anweisung Print wird gerne beim Debuggen für eine einfache einzeilige Ausgabe verwendet, weil man mit ihr durch einen Klick auf die Schaltfläche Abbrechen ein Makro beenden kann, s. Bild 67, fürs Debuggen eine große Hilfe. Platzieren Sie eine Print-Anweisung vor oder hinter den potenziell problematischen Zeilen. Wenn die Werte nicht korrekt aussehen, können Sie mit einem Klick das Makro beenden.

Tipp Das Nette an der Anweisung Print ist, dass man ein Makro über die Schaltfläche Abbrechen stoppen kann, mit dem Vorteil – oder Nachteil –, dass der Fokus auf die IDE springt, auf die Zeile mit der Print-Anweisung.

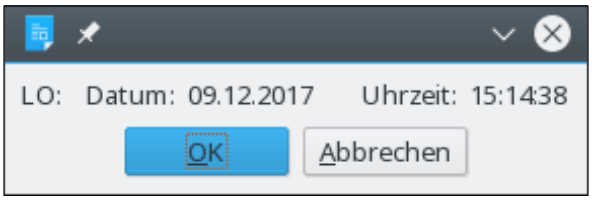


Bild 67. *Die Funktionen Spc und Tab positionieren die einzelnen Stringausgaben.*

Wenn Sie viele Print-Anweisungen zum Debuggen brauchen, dann versehen Sie die Daten mit Erläuterungen, um sich selbst an die Zusammenhänge zu erinnern.

```
Print "Vor der Schleife, x = ";x
For i = 0 To 10
  Print "In der Schleife, i = "; i; " und x = "; x
Next
```

Wenn Sie einen String ausgeben, der ein Zeilenumbruchzeichen (ASCII 10 oder 13) enthält, wird für jede neue Zeile ein neuer Dialog angezeigt. Der Code im Listing 177 gibt drei Dialoge in Folge aus, mit dem jeweiligen Text „eins“, „zwei“ und „drei“. Der Print-Dialog ist zu mehrzeiliger Ausgabe durchaus in der Lage. Wenn eine Textzeile zu lang wird, wird sie auf mehrere Zeilen umbrochen. Mit anderen Worten, obwohl Print selbständig Zeilen umbricht, so hat der Nutzer aber keine Möglichkeit, einen Zeilenumbruch im Dialog zu erzwingen.

Listing 177. *Print: Neue Zeile im String führt zu weiterem Dialog.*

```
Sub PrintDisplayThreeDialogs
  Print "eins" & Chr$(10) & "zwei" & Chr$(13) & "drei"  ' Gibt drei Dialoge aus
End Sub
```

Die Print-Anweisung hat einfache, klare Regeln zur Zahlenformatierung. Positiven Zahlen geht ein Leerzeichen, negativen Zahlen ein Minuszeichen voraus. Zahlen mit Dezimalanteil werden in Exponentialnotation dargestellt, wenn sie zu lang werden.

Die Print-Anweisung gibt den Dialog jedes Mal direkt aus, es sei denn, die Anweisung endet mit einem Semikolon oder einem Komma. In diesem Fall wird der Text jeder Print-Anweisung gespeichert und kumuliert, bis eine Print-Anweisung ohne Semikolon oder Komma am Ende gefunden wird.

```
Sub PrintMultiLineExample
  Print "eins",  'Noch keine Ausgabe, endet mit einem Komma
  Print "zwei"   'Gibt „eins    zwei“ aus
  Print "drei",  'Noch keine Ausgabe, endet mit einem Komma
  Print "vier"; 'Noch keine Ausgabe, endet mit einem Semikolon
  Print          'Gibt „drei    vier“ aus
End Sub
```

9.5.2. Mehrzeilige Ausgabe

Die Anweisung MsgBox bietet mehr Möglichkeiten der Dialoggestaltung als die Anweisung Print, kann aber nur einen einzigen String ausgeben. Stringausdrücke mit einem Zeilenumbruchzeichen (ASCII 10 oder 13) werden im selben Dialog dargestellt. Mit jedem Zeilenumbruchzeichen wird im Dialog eine neue Zeile gestartet.

Listing 178. *Gibt einen einfachen Meldungsdialog mit Zeilenumbruch aus.*

```
Sub ExampleMsgBoxWithReturn
  MsgBox "eins" & Chr$(10) & "zwei"
End Sub
```

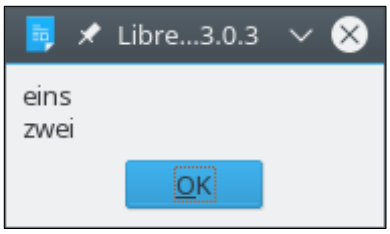


Bild 68. *Ein einfacher MsgBox-Dialog mit nur einer OK-Schaltfläche.*

Der Dialog im Bild 68 ist sehr einfach. Die Funktion MsgBox akzeptiert zwei weitere Argumente, s. Listing 179. Das Argument DialogTitel ist die Titelzeile des Dialogs. Die Tabelle 77 zeigt die gültigen Werte für das Argument DialogTyp. Der Dialogtyp bestimmt, welche Schaltflächen es im Dialog gibt, welches die Standardschaltfläche ist und welche Symbole im Dialog angezeigt werden.

Listing 179. *Die Anweisung (Funktion) MsgBox kann einen Typ und einen Dialogtitel erhalten.*

```
MsgBox(Meldung)
MsgBox(Meldung, DialogTyp)
MsgBox(Meldung, DialogTyp, DialogTitel)
```

Tabelle 77. *Gültige Werte für den Dialogtyp.*

Wert	Beschreibung	Benannte Konstante (nur LO)
0	Zeigt nur die OK-Schaltfläche.	MB_OK
1	Zeigt die Schaltflächen OK und Abbrechen.	MB_OKCANCEL
2	Zeigt die Schaltflächen Abbrechen, Wiederholen und Ignorieren.	MB_ABORTRETRYCANCEL
3	Zeigt die Schaltflächen Ja, Nein und Abbrechen.	MB_YESNOCANCEL
4	Zeigt die Schaltflächen Ja und Nein.	MB_YESNO
5	Zeigt die Schaltflächen Wiederholen und Abbrechen.	MB_RETRYCANCEL
16	Fügt dem Dialog das Stoppschild-Symbol hinzu.	MB_ICONSTOP
32	Fügt dem Dialog das Fragezeichen-Symbol hinzu.	MB_ICONQUESTION
48	Fügt dem Dialog das Ausrufezeichen-Symbol hinzu.	MB_ICONEXCLAMATION
64	Fügt dem Dialog das Informations-Symbol hinzu.	MB_ICONINFORMATION
128	Die Standardschaltfläche im Dialog ist die erste Schaltfläche. Das ist die Standardeinstellung.	
256	Die Standardschaltfläche im Dialog ist die zweite Schaltfläche.	MB_DEFBUTTON2
512	Die Standardschaltfläche im Dialog ist die dritte Schaltfläche.	MB_DEFBUTTON3

Listing 180. *Das Verhalten der MsgBox-Typen.*

```
Sub MsgBoxExamples()
  Dim i%
  Dim values     'Werteliste
  values = Array(0, 1, 2, 3, 4, 5)
  For i = LBound(values) To UBound(values)
    MsgBox ("Dialogtyp: " & values(i), values(i))
  Next
  values = Array(16, 32, 48, 64, 128, 256, 512)
  For i = LBound(values) To UBound(values)
    MsgBox ("Ja, Nein, Abbrechen, mit Typ: " & values(i), values(i) + 3)
```

```
    'Mit LO auch möglich:
    'MsgBox ("Ja, Nein, Abbrechen, mit Typ: " & values(i), values(i) + MB_YESNOCANCEL)
  Next
End Sub
```

Man kann gleichzeitig mehrere Dialogtypen angeben, um die gewünschten Symbole und Schaltflächen mit der Standardschaltfläche darzustellen. Die Wahl der Schaltflächen wird in den ersten vier Bits kodiert (die Werte 0-15 sind binär 0000-1111), die Wahl der Symbole und der Standardschaltfläche in den höheren Bits (64 ist zum Beispiel binär 01000000). Zur Kombination der Attribute verwenden Sie Or oder die Addition der Werte. (Das ist wie bei der Behandlung von Dateiattributen).

Obwohl Sie einen Dialog mit der Schaltfläche „Abbrechen“ anzeigen können, so wird die Makroausführung dadurch nicht wie bei der Print-Anweisung gestoppt. Stattdessen gibt die Funktion MsgBox einen Integer-Wert zurück, der die gewählte Schaltfläche identifiziert (s. Tabelle 78). Ein Klick auf „Abbrechen“ gibt den Wert 2 zurück (3 beim Dialogtyp 2). Diesen Wert können Sie in Ihrem Code prüfen und je nach Lage entscheiden, wie das Makro darauf reagieren soll.

Tabelle 78. *Von der Funktion MsgBox zurückgegebene Werte.*

Wert	Beschreibung	Benannte Konstante (nur LO)
1	OK	IDOK
2	Abbrechen	IDCANCEL
3	Abbruch	IDABORT
4	Wiederholen	IDRETRY
5	Ignorieren	IDIGNORE
6	Ja	IDYES
7	Nein	IDNO

Wenn Sie also wollen, dass nach einem Klick auf die Schaltfläche „Abbrechen“ das Makro beendet wird, dann müssen Sie den Rückgabewert prüfen, wie es im Listing 181 geschieht. Die Meldung enthält ein Zeilenumbruchzeichen, so dass zwei Textzeilen ausgegeben werden. Der Dialogtyp fordert drei Schaltflächen und ein Symbol an und bestimmt die zweite Schaltfläche als Standard (s. Bild 69). Je nach der vom Nutzer gewählten Schaltfläche macht das Makro unterschiedliche Dinge.

Listing 181. *Darstellung der Arbeitsweise von MsgBox.*

```
Sub ExampleMsgBox
  Dim nReturnCode As Integer   'Rückgabewert
  Dim nDialogType As Integer
  Dim sMessage As String
  sMessage = "Ein Fehler ist aufgetreten!" & Chr$(10) _
             & "Trotzdem mit der wichtigen Arbeit weitermachen?"
  REM   3 (LO auch: MB_YESNOCANCEL) heißt Ja, Nein, Abbrechen
  REM  48 (LO auch: MB_ICONEXCLAMATION) zeigt das Symbol Ausrufezeichen
  REM 256 (LO auch: MB_DEFBUTTON2) heißt, dass die zweite Schaltfläche Standard ist.
  nDialogType = 3 Or 48 Or 256
  nReturnCode = MsgBox(sMessage, nDialogType, "Huch, ein Fehler")
  If nReturnCode = 2 Then     'LO auch: If ReturnCode = IDCANCEL Then
    Print "Das Makro wird umgehend abgebrochen!"
    Stop       'Abbruch. Die letzte Zeile des Makros wird nicht mehr ausgeführt.
  ElseIf nReturnCode = 6 Then  'LO auch: If ReturnCode = IDYES Then
    Print "Sie haben sich für Ja entschieden."
  ElseIf nReturnCode = 7 Then  'LO auch: If ReturnCode = IDNO Then
    Print "Sie haben sich für Nein entschieden."
  Else
```

```
    Print "Hierhin werde ich nie gelangen!", nReturnCode
  End If
  Print "Die Subroutine wird nun verlassen."
End Sub
```

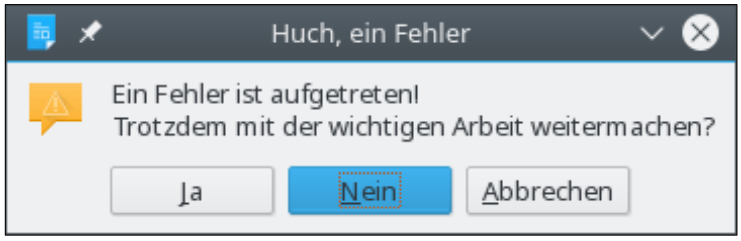


Bild 69. *Gestaltung einer MsgBox mit einem Symbol und mehreren Schaltflächen.*

9.5.3. Eingabeaufforderung

Mit der Funktion InputBox fordern Sie den Nutzer zu einer Eingabe auf. Sie können dem Dialog einen Titel geben. Wenn Sie einen Standardtext angeben, wird er in der Eingabezeile angezeigt. Der Dialog enthält eine Eingabezeile und die Schaltflächen OK und Abbrechen. Die Funktion InputBox gibt der ausführenden Anweisung einen String zurück. Ein Klick auf Abbrechen erzeugt einen String mit der Länge 0.

```
InputBox(Meldung)
InputBox(Meldung, Titel)
InputBox(Meldung, Titel, Standard)
InputBox(Meldung, Titel, Standard, x_pos, y_pos)
```

Die Positionsangaben sind in Twips und relativ zur oberen linken Ecke des aktuellen Fensters (1 Zoll sind 1440 Twips). Ohne spezifische Positionsangaben wird der Dialog horizontal und vertikal über dem aktuellen Fenster zentriert. Das Beispiel im Listing 182 positioniert den Eingabedialog zwei Zoll vom linken Rand des Fensters und vier Zoll vom oberen Rand. Die Größe des Dialogs wird automatisch aus der Meldung und den Schaltflächen festgelegt. Das Layout wird auch hier wie bei den anderen elementaren Eingabe- und Ausgabedialogen von OOo bestimmt.

Listing 182. *Beispiel für InputBox.*

```
Sub ExampleInputBox
  Dim sReturn As String      'Rückgabewert
  Dim sMsg As String         'Eingabeaufforderung
  Dim sTitle As String       'Fenstertitel
  Dim sDefault As String     'Standardtext
  Dim nXPos As Integer       'Twips vom linken Rand
  Dim nYPos As Integer       'Twips vom oberen Rand
  nXPos = 1440 * 2           'Zwei Zoll Abstand vom linken Fensterrand
  nYPos = 1440 * 4           'Vier Zoll Abstand vom oberen Fensterrand
  sMsg  = "Bitte geben Sie einen sinnvollen Text ein:"
  sTitle = "Sinnvoller Text"
  sDefault = "Hallo"
  sReturn = InputBox(sMsg, sTitle, sDefault, nXPos, nYPos)
  If sReturn <> "" Then
    REM Gibt den eingegebenen Text aus, eingeschlossen in doppelten Anführungszeichen
    Print "Sie haben """; sReturn; """ eingegeben."
  Else
    Print "Sie haben entweder einen leeren String eingegeben " & _
          "oder die Schaltfläche Abbrechen gewählt."
  End If
End Sub
```

Bild 70 zeigt den Dialog, wie er auf dem Bildschirm erscheint. Jeder Tastendruck ersetzt den Standardtext, weil dieser beim Dialogstart markiert ist. Das Makro im Listing 182 untersucht den Rückgabewert und prüft, ob der String leer ist, das heißt die Länge null hat. Ein leerer String könnte bedeuten, dass der Nutzer den Dialog über die Schaltfläche Abbrechen beendet hat oder dass der Nutzer einen leeren String eingegeben und dann die Schaltfläche OK ausgelöst hat. Diese beiden Fälle sind nicht auseinander zu halten.

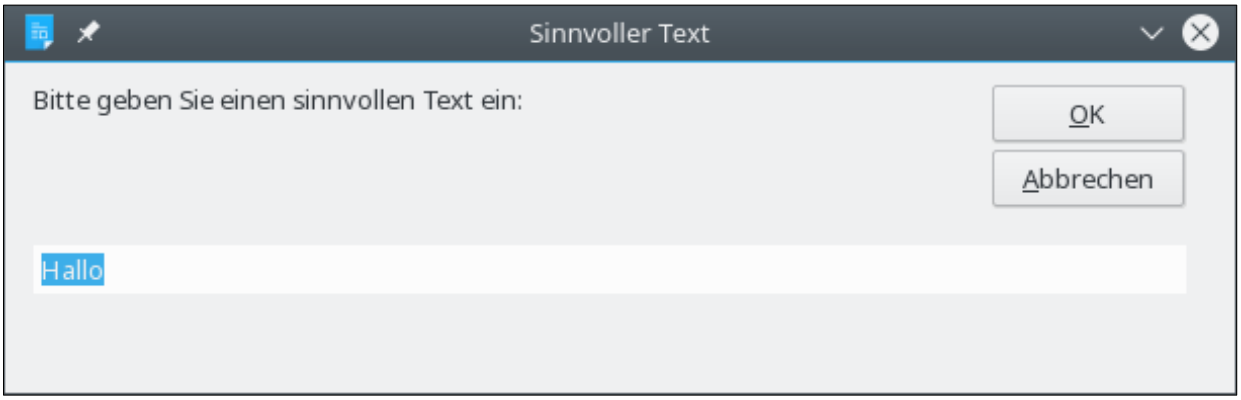


Bild 70. *InputBox mit markiertem Standardtext.*

9.6. Vermischte Routinen

Die vermischten Routinen in diesem Abschnitt sind Universalroutinen, die in keinem unmittelbaren Zusammenhang stehen (s. Tabelle 79).

Tabelle 79. *Vermischte Basic-Funktionen.*

Funktion	Beschreibung
Beep	Gibt einen systemabhängigen Ton aus.
CBool(Ausdruck)	Konvertiert einen Integer-Wert oder einen String zu einem booleschen Wert.
Environ(String)	Gibt den Wert einer Umgebungsvariablen zurück.
GetSolarVersion	Die interne aktuelle Version.
CreateObject(Objekt_Typ)	Dynamische Version von „Dim As New“.
Erase(Objekt)	Löscht ein Objekt aus dem Arbeitsspeicher.

Die Anweisung Beep erzeugt einen systemabhängigen Ton. Man kann weder die Tonhöhe noch die Tondauer beeinflussen. Auf manchen Systemen wird über den eingebauten Lautsprecher eine konfigurierbare Tondatei abgespielt, auf anderen wird ein Systemereignis erzeugt, das einen systemdefinierten Ton über die systemnahe interne Hardware ausgibt.

```
Beep          'Erzeugt einen Ton
Wait(500)     'Wartet 1/2 Sekunde
Beep          'Erzeugt einen Ton
```

Mit der Funktion CBool wird ein String oder eine Zahl zu einem booleschen Wert konvertiert. Jeder numerische Ausdruck mit dem Wert null ergibt False. Numerische Ausdrücke mit einem Wert abweichend von null ergeben True. Strings mit der Zeichenfolge „true“ oder „false“ ergeben True beziehungsweise False, die Groß- und Kleinschreibung spielt keine Rolle. Ein String, der nicht genau True oder False ergibt, wird als Zahl behandelt. Wenn der String weder „true“ oder „false“ noch eine Zahlenfolge darstellt, entsteht ein Laufzeitfehler.

```
Print CBool(False)      'False
Print CBool(13)         'True
Print CBool("13")       'True
```

```
Print CBool("trUe")     'True
Print CBool("&h1")      'True
Print CBool("13xx")     'Laufzeitfehler
Print Cbool("Truee")    'Laufzeitfehler
```

Umgebungsvariablen können Sie mit der Funktion Environ abfragen. Wenn es die Umgebungsvariable nicht gibt, wird ein leerer String zurückgegeben. Es gibt keine Methode, Umgebungsvariablen zu setzen oder zu ändern.

```
Print Environ("PATH")
Print Environ("TEMP")
```

Mit GetSolarVersion erhalten Sie die interne Versionsnummer von OOo. Sie können in Ihrem Makro Hilfskonstruktionen (so genannte Workarounds) um bekannte Bugs verwenden, wenn Sie wissen, in welchen OOo-Versionen sie auftreten.

```
Print GetSolarVersion
```

Mit der Funktion CreateObject können Objekte dynamisch erstellt werden. Ein Objekt, das mit „Dim v As New“ zu erzeugen ist, kann man auch mit der Funktion CreateObject erzeugen. Das heißt, Sie können damit Struktur-Objekte erstellen, also auch für benutzereigene Datentypen. Die in OOo zugrunde liegenden speziellen Datenobjekte heißen Universal Network Objects (UNO) – zu den Einzelheiten s. Kapitel 10. Universal Network Objects (UNO) Diese Objekte können nicht mit CreateObject erzeugt werden. OOo definiert aber auch Strukturen, die nicht UNO-Objekte sind. Diese Objekte können wiederum mit „Dim v As New“ oder mit CreateObject erzeugt werden, s. Listing 183.

Listing 183. *Erzeugt ein Objekt mit CreateObject oder mit Dim As New.*

```
'Entweder so
Dim oProp As New com.sun.star.beans.PropertyValue      'OOo-Struktur
'Oder so
Dim o As Object
o = CreateObject("com.sun.star.beans.PropertyValue")
```

Listing 183 demonstriert, wie man eine Variable mit einem von OOo definierten Variablentyp erzeugt, der wie ein benutzereigener Typ strukturiert ist. Der Typname dieses Objekts lautet „com.sun.star.beans.PropertyValue“. Viele Objekte in OOo haben ähnlich lange und lästige Namen. Wenn man über solche Variablentypen schreibt oder diskutiert, kürzt man üblicherweise den Typnamen auf den letzten Namensteil ab. Zum Beispiel: Die Eigenschaft Name wird in der PropertyValue-Variablen gesetzt (s. Listing 184). Objekte vom Typ PropertyValue haben zwei Eigenschaften: Name als String und Value als Variant.

Listing 184. *Erstellt Strukturen und Array-Variablen und löscht sie mit Erase().*

```
Dim aProp As New com.sun.star.beans.PropertyValue
aProp.Name = "Vorname"          'Setzt die Eigenschaft Name
aProp.Value = "Kunibert"        'Setzt die Eigenschaft Value
Erase aProp
Print IsNull(aProp)             'True
Print IsEmpty(aProp)            'False

REM Eine neue Variable!
Dim aPropr
aPropr = CreateObject("com.sun.star.beans.PropertyValue")
Erase aProp
Print IsNull(aProp)             'True
Print IsEmpty(aProp)            'False

Dim a
a = Array("Hallo", 2)
Erase a
```

```
Print IsNull(a)                 'False
Print IsEmpty(a)                'True

Dim b() As String
ReDim b(0 To 1) As String
b(0) = "Hallo" : b(1) = "du"
'b() = "Hallo"                  'Laufzeitfehler, Variable nicht belegt (Erwartet)
'Print b()                      'Laufzeitfehler, Variable nicht belegt (Erwartet)
'Erase b()                      'Syntaxfehler, nicht sehr überraschend
Erase b                         'Ich hätte nicht erwartet, dass das geht.
Print IsNull(b())               'False
Print IsEmpty(b())              'False
Print IsArray(b())              'False, das ist sicher schlimm.
'Print LBound(b())              'Laufzeitfehler, Variable nicht belegt.
b() = "Hallo"                   'Komisch, jetzt kann ich b() als Stringvariable nutzen.
Print b()                       'Hallo
```

Listing 184 veranschaulicht die Anweisung Erase, die es seit OOo 2.0 gibt. Mit Erase wird Platz im Arbeitsspeicher geschaffen anstelle der Daten, die nicht mehr benötigt werden. Verwenden Sie Erase erst, wenn Sie die Variable wirklich nicht mehr brauchen.

Mit der Funktion CreateObject erstellen Sie ein Objekt dynamisch – wenn Sie es also nicht schon bei der Deklarierung nutzen wollen. Mit CreateObject können Sie in einer Anweisung nur ein Objekt erstellen. Für ein Array eines bestimmten Typs wählen Sie die Konstruktion „Dim As New“, s. Listing 185. Sie können sogar die Array-Dimensionen ändern und dabei die vorhandenen Daten erhalten. Mühseliger ist es, ein Array zu deklarieren und es dann einzeln mit den passenden Werten zu füllen, s. Listing 186.

Listing 185. *Beispiel für ReDim mit Preserve.*

```
Sub ExampleReDimPreserveProp
  REM Auf diese Art leicht zu erzeugen
  Dim oProps(2) As New com.sun.star.beans.PropertyValue
  oProps(0).Name = "Vorname"  : oProps(0).Value = "Hans"
  oProps(1).Name = "Nachname" : oProps(1).Value = "Fasel"
  oProps(2).Name = "Alter"    : oProps(2).Value = 53
  ReDim Preserve oProps(3) As New com.sun.star.beans.PropertyValue
  oProps(3).Name = "Gewicht"  : oProps(3).value = 97
  Print oProps(2).Name   'Alter
End Sub
```

Listing 186. *Man kann einem deklarierten Array PropertyValue-Variablen hinzufügen.*

```
REM Dies ist mühseliger, aber es geht...
Dim oProps(2)
oProps(0) = CreateObject("com.sun.star.beans.PropertyValue")
oProps(1) = CreateObject("com.sun.star.beans.PropertyValue")
oProps(2) = CreateObject("com.sun.star.beans.PropertyValue")
oProps(0).Name = "Vorname"  : oProps(0).Value = "Hans"
oProps(1).Name = "Nachname" : oProps(1).Value = "Fasel"
oProps(2).Name = "Alter"    : oProps(2).Value = 53
```

Bei der Zuweisung eines Arrays zu einem anderen wird eine Referenz zugewiesen, so dass beide Arrays dasselbe Array-Objekt referenzieren. Variablentypen wie Integer und PropertyValue werden jedoch als Kopie zugewiesen. Nicht zu beachten, welche Typen mit ihrem Wert und welche Typen als Referenz kopiert werden, ist eine häufige Fehlerquelle. Strukturen und ganzzahlige Typen (wie Integer und String) werden mit ihrem Wert kopiert, aber Arrays und UNO-Variablen (sehen Sie später) als Referenz. Im Listing 187 wird gezeigt, wie ganz offensichtlich der Wert kopiert wird.

Listing 187. *PropertyValue wird als Wert kopiert.*

```
Sub ExampleCopyAsValue
  Dim aProp1
  Dim aProp2
  aProp1 = CreateObject("com.sun.star.beans.PropertyValue")
  aProp1.Name = "Alter"    'Setzt die Eigenschaft Name in der ersten Variablen
  aProp1.Value = 27        'Setzt die Eigenschaft Value in der ersten Variablen
  aProp2 = aProp1          'Eine Kopie wird erzeugt
  aProp2.Name = "Gewicht" 'Setzt die Eigenschaft Name in der zweiten Variablen
  aProp2.Value = 97        'Setzt die Eigenschaft Value in der zweiten Variablen
  Print aProp1.Name, aProp2.Name   'Alter  Gewicht
End Sub
```

Tipp Standardobjekte werden als Wert, UNO-Variablen als Referenz kopiert.

Wird eine Integer-Variable einer anderen zugewiesen, wird der Wert kopiert und sonst nichts. Die beiden Variablen sind und bleiben unabhängig voneinander. Das gilt auch für Strukturen. Später werden Sie sehen, dass Textcursors die Eigenschaft CharLocale besitzen, mit den Werten für Land und Sprache des Textes unter dem Textcursor. Üblich, aber falsch ist es, die Gebietseinstellung direkt in der Variablen zu setzen. Dadurch werden Sprache und Land in einer Kopie der Eigenschaft CharLocale gesetzt statt in der Kopie, die der Textcursor nutzt. Ich sehe diesen Fehler sehr oft.

```
oCursor.CharLocale.Language = "fr"  'Setzt die Sprache Französisch in einer Kopie
oCursor.CharLocale.Country =  "CH"  'Setzt das Land Schweiz in einer Kopie
```

Eine korrekte Methode, die Gebietseinstellung zu setzen, besteht darin, eine neue Locale-Struktur zu erstellen, diese neue Struktur zu modifizieren, um dann die neue Struktur in den Textcursor zu kopieren.

```
Dim aLocale As New com.sun.star.lang.Locale
aLocale.Language = "fr"        'Setzt Locale auf die Sprache Französisch
aLocale.Country  = "CH"        'Setzt Locale auf das Land Schweiz
oCursor.CharLocale = aLocale   'Weist den Wert neu zu
```

Sie können auch eine Kopie der Struktur erstellen, die kopierte Struktur modifizieren, um dann die modifizierte Struktur in den Textcursor zu kopieren.

```
Dim aLocale
aLocale = oCursor.CharLocale   'Es kann auch eine Kopie sein
aLocale.Language = "fr"        'Setzt Locale auf die Sprache Französisch
aLocale.Country  = "CH"        'Setzt Locale auf das Land Schweiz
oCursor.CharLocale = aLocale   'Weist den Wert neu zu
```

9.7. Partition

Partition ist nicht dokumentiert und wurde wahrscheinlich zur Kompatibilität mit VB hinzugefügt. Partition gibt einen Variant-String zurück, der anzeigt, in welchem einer Reihe von berechneten Bereichen eine Zahl vorkommt.

```
Partition(Zahl, Startwert, Endewert, Intervall)
```

Betrachten wir die folgenden Werte:

```
Startwert = 0
Endewert = 17
Intervall = 5
```

Die folgenden „Partitionen“ werden angenommen:

1. „ :-1“ Alles unter 0
2. „.0:.4“ Fünf Zahlen von 0 bis 4.

3. „.5:.9“ Fünf Zahlen von 5 bis 9.
4. „10:14“ Fünf Zahlen von 10 bis 14.
5. „15:17“ Drei Zahlen von 15 bis 17.
6. „18: “ Alles über 17.

Das Beispiel im Listing 188 prüft Zahlen unter und über dem Intervall. Wie erwartet befinden sich Werte vor dem ersten Intervall in der Partition „ :-1“. Werte, die in ein Intervall fallen, werden sauber identifiziert. Ein wenig heikel ist nur, dass das letzte Intervall behauptet, die Werte „15:19“ einzuschließen, obwohl 18 und 19 über das Intervall hinaus gehen.

Listing 188. *Partition mit einer Reihe von Werten.*

```
Sub ExamplePartition
  Dim i%
  Dim s$
  For i = -2 To 20
    s = s & "Partition(" & i & ", 0, 17, 5) = " & _
        Partition(i, 0, 17, 5) & Chr$(10)
  Next
  MsgBox s
End Sub
```

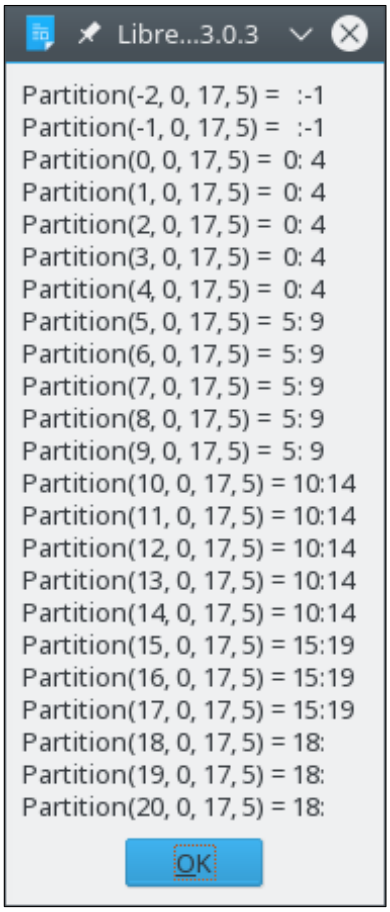


Bild 71. *Partition zeigt das Intervall, das eine bestimmte Zahl enthält.*

Die von Partition zurückgegebenen Werte sind sorgsam formatiert. Der untere und der obere Wert haben dieselbe Anzahl an Zeichen. Somit können sie korrekt sortiert werden, wenn man das wünscht. Es hilft auch beim Zergliedern der Rückgabewerte.

9.8. Inspizierung und Erkennung von Variablen

Basic verfügt über eine Reihe von Funktionen zur Inspizierung und Erkennung von Variablen, s. Tabelle 80. Diese Routinen sind häufig dann nützlich, wenn man eine Funktion aufruft und sich nicht sicher über den Rückgabetyp ist. Sie werden auch beim Debuggen gebraucht. Sie könnten diese Funktionen zum Beispiel zur Absicherung verwenden, ob ein Rückgabetyp gültig ist.

Tabelle 80. *Funktionen zur Inspizierung von Basic-Variablen.*

Funktion	Beschreibung
IsArray	Ist die Variable ein Array?
IsDate	Enthält der String ein gültiges Datum?
IsEmpty	Ist die Variable eine leere Variant-Variable?
IsMissing	Ist die Variable ein ausgelassenes optionales Argument?
IsNull	Ist die Variable ein nicht initialisiertes Objekt?
IsNumeric	Enthält der String eine gültige Zahl?
IsObject	Ist die Variable ein Objekt?
IsUnoStruct	Ist die Variable eine UNO-Struktur?
TypeName	Gibt den Typnamen des Objekts als String zurück.
TypeLen	Gibt die Anzahl der Bytes zurück, die der Typ belegt (Strings = Länge).
VarType	Gibt den Typ der Variablen als Integer zurück.

Mit IsArray sehen Sie, ob eine Variable ein Array ist, s. Listing 189. Wenn IsArray True zurückgibt, heißt das noch nicht, dass die Variable Daten enthält oder ob sie überhaupt dimensioniert ist – es heißt nur, dass sie existiert und als Array definiert ist. Mit den Funktionen UBound und LBound ermittelt man, wie schon erläutert, die obere beziehungsweise untere Dimensionsgrenze.

Listing 189. *Mit IsArray sehen Sie, ob eine Variable ein Array ist.*

```
Dim n As Long       'Dies ist KEIN Array
Dim a() As String   'Dies ist ein Array
Dim b(5)            'Dies ist ein Array
Dim v As Variant    'Dies ist noch kein Array
Print IsArray(v)    'False
Print IsArray(n)    'False
Print IsArray(a)    'True
Print IsArray(b())  'True
ReDim v(3)          'Nun ist es ein Array!
Print IsArray(v())  'True
```

Die Funktion IsDate prüft, ob ein String ein gültiges Datum darstellt, s. Listing 190. Das Argument wird vor der Überprüfung zu einem String konvertiert, wodurch ein numerisches Argument immer False ergibt. Es wird nicht nur die Syntax geprüft, sondern auch, ob das Datum überhaupt gültig ist. Diese Kontrolle erstreckt sich nicht auf die Uhrzeitkomponente des Strings.

Listing 190. *IsDate überprüft, ob ein String ein gültiges Datum enthält.*

```
Print IsDate("1. Dezember 1582 2:13:42") 'True
Print IsDate("2:13:42")                  'True
Print IsDate("1.12.1582")                'True
Print IsDate(Now)                        'True
Print IsDate("26:61:112")                'True
Print IsDate(True)                       'False, wird erst zu einem String konvertiert
Print IsDate(32686,22332)                'False, wird erst zu einem String konvertiert
Print IsDate("29.02.2003")               'False, nur 28 Tage im Februar 2003
```

Wie die Funktion IsDate schaut sich auch die Funktion IsNumeric Strings an, s. Listing 191. Wenn das Argument nicht gänzlich aus einer einzigen gültigen Zahl besteht, abgesehen von Leerzeichen am Anfang oder Ende beziehungsweise von Anführungszeichen, wird False zurückgegeben.

Listing 191. *IsNumeric ist sehr pingelig mit der Form des Arguments.*

```
Print IsNumeric(" 123")    'True
Print IsNumeric(" 12 3")   'False
Print IsNumeric(1.23)      'True
Print IsNumeric(1,23)      'True
Print IsNumeric("123abc")  'False
Print IsNumeric(True)      'False
Print IsNumeric(Now)       'False
```

Variant-Variablen starten ohne jeglichen Wert, sie sind als Empty (leer) initialisiert. Objektvariablen werden mit dem Wert Null initialisiert. Die Funktionen IsEmpty und IsNull prüfen genau diese Zustände. Mit der Funktion IsObject erfahren Sie, ob eine Variable ein Objekt ist.

```
Dim v As Variant  'Ist beim Start nicht initialisiert, leer (Empty)
Dim o As Object   'Initialisiert zu Null

Print IsObject(v) 'False  Nein, Variant ist kein Objekt
Print IsObject(o) 'True   Ja, dies ist ein Objekt

Print IsEmpty(v)  'True   Variants starten als Empty, nicht initialisiert
Print IsNull(v)   'False  Um Null zu sein, muss ein Variant etwas enthalten

Print IsEmpty(o)  'False  Variants starten als Empty, nicht als Objekte
Print IsNull(o)   'True   Objekte starten als Null

v = 0
Print IsObject(v) 'True   Variant wurde gerade ein Objekt
Print IsEmpty(v)  'False  Variant enthält nun einen Wert (ein Objekt)
Print IsNull(v)   'True   Variant enthält ein Null-Objekt
```

Mit der Funktion IsMissing finden Sie heraus, ob ein optionales Argument fehlt. Üblicherweise wird ein Standardwert angenommen, wenn ein Argument nicht angegeben ist.

```
Sub TestOptional
  Print "Arg ist "; ExampleOptional()         'Arg ist nicht da
  Print "Arg ist "; ExampleOptional("Hallo")  'Arg ist Hallo
End Sub

Function ExampleOptional(Optional x) As String
  ExampleOptional = IIF(IsMissing(x), "nicht da", CStr(x))
End Function
```

Die Funktion IsUnoStruct prüft, ob eine Variable eine von OOo definierte Struktur enthält.

```
  Dim v
  Print IsUnoStruct(v)                                 'False
  v = CreateUnoStruct("com.sun.star.beans.Property") 'Erstellt eine UNO-Struktur
  Print IsUnoStruct(v)                                 'True
```

Die Funktion TypeName gibt den Typ einer Variablen als String zurück, wohingegen die Funktion VarType den Typ als Ganzzahl angibt. Die Tabelle 81 enthält eine Liste der verfügbaren Typen. Die erste Spalte, Basic, zeigt, ob der Typ zu Basic gehört und daher wohl erkannt wird. Die anderen Werte repräsentieren OOo-interne Typen. Basic bildet normalerweise interne Typen auf Basic-Typen ab, so dass Sie diese anderen Typen eher nicht zu Gesicht bekommen werden. Sie tauchen jedoch im Quellcode auf und sind hier der Vollständigkeit halber aufgeführt.

Tabelle 81. *Variablentypen und ihre Namen.*

Basic	VarType	TypeName	Länge	Beschreibung
Ja	0	Empty	0	Variant, nicht initialisiert
Ja	1	Null	0	Objekt, keine konkreten Daten
Ja	2	Integer	2	Integer (kurze Ganzzahl)
Ja	3	Long	4	Long Integer (lange Ganzzahl)
Ja	4	Single	4	Single (kurze Fließkommazahl)
Ja	5	Double	8	Double (lange Fließkommazahl)
Ja	6	Currency	8	Currency (Ganzzahl mit 4 Dezimalstellen)
Ja	7	Date	8	Date (Datum und Uhrzeit)
Ja	8	String	strlen	String (Zeichenkette)
Ja	9	Object	0	Objekt
Nein	10	Error	2	Interner OOo-Typ
Ja	11	Boolean	1	Boolesche Variable
Ja	12	Variant	0	Variant-Variablen verhalten sich wie jeder andere Typ.
Nein	13	DataObject	0	Interner OOo-Typ
Nein	14	Unknown Type	0	Interner OOo-Typ
Nein	15	Unknown Type	0	Interner OOo-Typ
Nein	16	Char	1	Interner OOo-Typ, ein einfaches Textzeichen
Ja	17	Byte	1	Interner OOo-Typ, kann aber mit CByte erzeugt werden
Nein	18	UShort	2	Interner OOo-Typ, Integer (16 Bits), vorzeichenlos
Nein	19	ULong	4	Interner OOo-Typ, Long (32 Bits), vorzeichenlos
Nein	20	Long64	8	Interner OOo-Typ, Long (64 Bits)
Nein	21	ULong64	8	Interner OOo-Typ, Long (64 Bits), vorzeichenlos
Nein	22	Int	2	Interner OOo-Typ, Integer (16 Bits)
Nein	23	UInt	2	Interner OOo-Typ, Integer (16 Bits), vorzeichenlos
Nein	24	Void	0	Interner OOo-Typ, kein Wert
Nein	25	HResult	0	Interner OOo-Typ
Nein	26	Pointer	0	Interner OOo-Typ, Zeiger auf irgendetwas
Nein	27	DimArray	0	Interner OOo-Typ
Nein	28	CArray	0	Interner OOo-Typ
Nein	29	Userdef	0	Interner OOo-Typ, benutzerdefiniert
Nein	30	Lpstr	strlen	Interner OOo-Typ, Langzeiger auf einen String
Nein	31	Lpwstr	strlen	Interner OOo-Typ, Langzeiger auf „breiten“ Unicodestring
Nein	32	Unknown Type	0	Interner Kerntyp String
Nein	33	WString	strlen	Interner OOo-Typ, „Breiter“ Unicodestring
Nein	34	WChar	2	Interner OOo-Typ, „Breites“ Unicodezeichen
Nein	35	Int64	8	Interner OOo-Typ, Integer (64 Bits)
Nein	36	UInt64	8	Interner OOo-Typ, Integer (64 Bits), vorzeichenlos
Nein	37	Decimal	16	OLE Automatisierungstyp, in VB verfügbar

Die Funktion TypeLen gibt an, wie viele Bytes eine Variable belegt. Der Wert ist fest eingebaut für jeden Wert außer für Strings, deren Länge zurückgegeben wird. Für Array-Variablen ist es immer die

Länge null. Das Makro im Listing 192 erzeugt alle Basic-Typen, fügt sie in ein Array und gibt deren Typ, Länge und Typnamen in einem String aus, zu sehen im Bild 72.

Listing 192. *Darstellung der Typinformationen für Standardtypen.*

```
Sub ExampleTypes
  Dim b As Boolean
  Dim c As Currency
  Dim t As Date
  Dim d As Double
  Dim i As Integer
  Dim l As Long
  Dim o As Object
  Dim f As Single
  Dim s As String
  Dim v As Variant
  Dim n As Variant
  Dim ta()
  Dim ss$
  n = Null
  ta() = Array(v, n, i, l, f, d, c, t, s, o, b, CByte(3) _
    CreateUnoValue("unsigned long", 10)))
  For i = LBound(ta()) To UBound(ta())
    ss = ss & ADPTypeString(ta(i))
  Next
  MsgBox ss, 0, "Typ, Länge und Name"
End Sub

Function ADPTypeString(v) As String
  Dim s As String               'Ausgabestring
  Dim i As Integer              'Arbeitswert
  s = s & "Typ = "              'Beginn des Ausgabestring
  i = VarType(v)                'Typnummer
  If i < 10 Then s = s & "0"    'Falls erforderlich, mit führender Null
  s = s & CStr(i)               'Danach die Typnummer
  If IsArray(v) Then
    s = s & " ("
    i = i And Not 8192
    If i < 10 Then s = s & "0"  'Falls erforderlich, mit führender Null
    s = s & CStr(i) & ")"       'Danach die Typnummer
  Else
    s = s & "   Länge = "       'Beginn des Strings für die Längenangabe
    i = TypeLen(v)              'Stringlänge
    If i < 10 Then s = s & "0"  'Falls erforderlich, mit führender Null
    s = s & CStr(i)             'Danach die Länge
  End If
  s = s & "  Name = "           'Beginnt des Strings für den Namen
  s = s & TypeName(v) & Chr$(10)'Danach der Name und ein Zeilenumbruch
  ADPTypeString = s             'Rückgabewert der Funktion
End Function
```

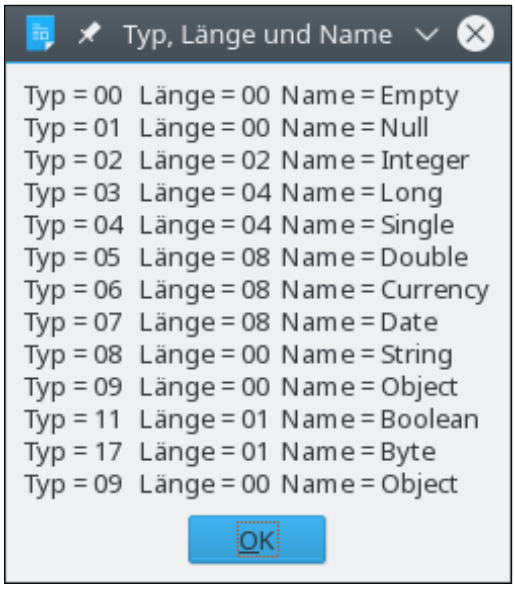


Bild 72. *Variablentypen, -längen und -namen.*

Die Funktion ADPTypeString erledigt die Aufgabe, den Ausgabestring zu erzeugen. Sie behandelt Arrays auf eigene Weise, weil sich die für Arrays zurückgegebenen Typnummern total von den Typnummern für Standardvariablen unterscheiden. Wenigstens erscheint es so, bis man einen genaueren Blick auf die Nummern wirft. Wenn Sie bei dem Wort Bits ins Zittern geraten, dann überspringen Sie am besten den Rest des Absatzes. Bei dem von VarType für ein Array zurückgegebenen Wert ist immer das Bit 14 gesetzt – binär eine 1 gefolgt von 13 Nullen. Das ist 2000 hexadezimal oder 8192 dezimal. Die Funktion IsArray ist so aufgebaut, dass sie Bit 14 des VarType prüft. Wenn man Bit 14 löscht, repräsentiert der Rest der Zahl den numerischen Typ des Arrays. Der Operator Not löscht jedes gesetzte Bit und setzt jedes Null-Bit, somit wird mit Not 8192 jedes Bit in der Zahl gesetzt mit Ausnahme von Bit 14. Wenn man diesen Wert dann mit mit dem Typ And-et, wird Bit 14 gelöscht, wobei der Rest der Bits intakt bleibt.

```
i = i And Not 8192
```

Die Länge eines Arrays wird immer als null zurückgegeben, also habe ich sie mit dem VarType im Listing 192 nicht aufgeführt. Der Code im Listing 193 ähnelt dem im Listing 192, doch die Typen sind Arrays. Beachten Sie, dass dem Typnamen für Arrays runde Klammern folgen, s. Bild 73.

Listing 193. *Beispiel für Informationen über Arraytypen.*

```
Sub ExampleTypesArray
  Dim b() As Boolean
  Dim c() As Currency
  Dim t() As Date
  Dim d() As Double
  Dim i() As Integer
  Dim l() As Long
  Dim o() As Object
  Dim f() As Single
  Dim s() As String
  Dim v() As Variant
  Dim ta(), j%
  Dim ss$
  ta() = Array(i, l, f, d, c, t, s, o, b, v)
  For j% = LBound(ta()) To UBound(ta())
    ss = ss & ADPTypeString(ta(j%))
  Next
  MsgBox ss, 0, "Typ und Name"
End Sub
```

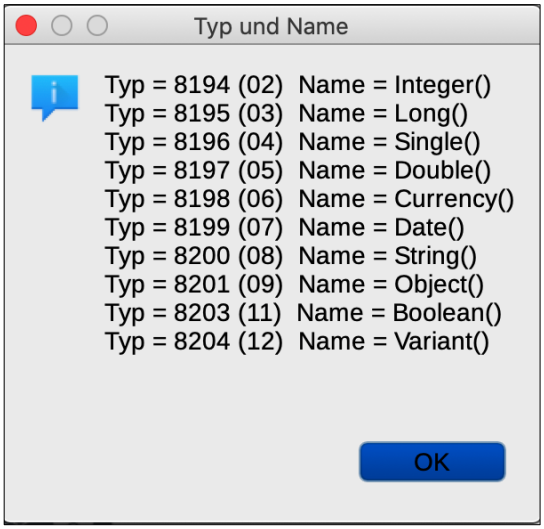


Bild 73. *Typ und Name für Array-Variablen.*

9.9. Nicht zu empfehlende Routinen und andere Kuriositäten

Ich höre noch meinen guten Freund Manfred sagen, dass ein verbotenes Buch aus einem Leser drei macht. Wenden Sie diese Logik aber nicht auf die in diesem Abschnitt vorgestellten Routinen an. Sie sind nicht offiziell dokumentiert. Auch wenn sie jetzt funktionieren, so mögen sie es in der nahen Zukunft wohl nicht mehr tun. Veraltete (engl.: deprecated) Routinen mögen immer noch existieren und funktionieren. Aber es kann jederzeit geschehen, dass sie entfernt werden. Dann bleiben noch leere Hülsen, die zwar kompilieren und laufen, aber nichts tun. In älterem Code werden Sie wohl den Gebrauch dieser Altlasten noch finden (s. Tabelle 82). In dieser Liste taucht auch CDec auf, weil es nur in der Windowsversion vorkommt, was eigentlich ganz dumm ist.

Tabelle 82. *Veraltete und fragwürdige Routinen.*

Routine	Kommentar
AboutStarBasic	Veraltete leere Hülse, die einmal einen Informationsdialog bot.
SendKeys	Veraltet, erzeugt einen Fehler.
DumpAllObjects(Pfad, boole)	Interne Debugging-Routine, keine praktische Verwendung bekannt.
Load(Objekt)	Veraltet.
Unload(Objekt)	Veraltet.
LoadPicture(Pfad)	Veraltet, lädt eine Bilddatei.
SavePicture(Objekt, Pfad)	Veraltet, eine Bilddatei zu speichern schlägt fehl.
CreatePropertySet(Objekt)	Veraltet, frühere Funktion zur UNO-Unterstützung.
CDec(Ausdruck)	Generiert den Typ Decimal, nur unter Windows.
GetGUIVersion()	Veraltet, gibt -1 zurück.

DumpAllObjects ist eine interne Debugging-Routine, die zwei Argumente akzeptiert. Das erste ist der Name einer Datei für die Textausgabe. Das zweite ist ein boolescher Wert, der bestimmt, ob jedes Objekt vor dem Abspeichern vollständig geladen werden soll. Aus Effizienzgründen werden manche Eigenschaften erst erzeugt, wenn zum ersten Mal auf sie zugegriffen wird. Das gilt auch für Basic-Objekte, die UNO-Objekte umhüllen.

```
DumpAllObjects("C:\foo.txt", True)
REM Alle Objekte werden vor dem Abspeichern vollständig geladen
```

Die Routinen LoadPicture und SavePicture beziehen sich auf die alten Basic-Dialoge, die zugunsten von UNO nicht mehr eingebunden sind. Das von LoadPicture zurückgegebene Objekt wurde zur Darstellung eines Bildes in einem Dialogkontrollelement gebraucht. Heute wird das Bild über eine Bildkontrolle mit Hilfe der UNO-Schnittstelle dargestellt.

```
Dim v
v = LoadPicture("C:\test1.jpg") 'Das Bild wird scheinbar geladen
SavePicture(v, "C:\test2.jpg")  'Es wird eine Datei mit der Größe 0 geschrieben
```

Die Funktion CreatePropertySet akzeptiert ein Objekt als Argument und gibt eine Property-Struktur zurück, die allerdings leer und nicht gerade von Nutzen ist. Diese Funktion wurde geschaffen, als die UNO-Funktionalität eingeführt wurde, und wird in Zukunft aus dem Quelltext verschwinden. Wenn Sie auf Code stoßen, der diese Funktion verwendet, so kann ich nur den Rat geben, das zu ändern.

```
v = CreatePropertySet(ThisComponent.Text)
```

9.10. Routinen, die ich nicht verstehe

Beim Lesen des Quellcodes finde ich Funktionen, von denen ich manche verstehe, andere aber nicht. Dieser Abschnitt stellt Funktionen vor, die ich nicht zufriedenstellend zu dokumentieren vermag.

Tabelle 83. *Funktionen, die ich nicht wirklich verstehe.*

Funktion	Beschreibung
EnableReschedule(boole)	Erlaubt wohl oder verhindert, dass Basic Threads oder Prozesse umleitet.
Me	Visual Basic. Benötigt CompatibilityMode.
RTL	Visual Basic. Benötigt CompatibilityMode.
GetDialogZoomFactorX()	Der Faktor, mit dem ein Dialog in der X-Richtung ausgedehnt wird. Wird verwendet, wenn ein Bild zur Vorschau skaliert wird.
GetDialogZoomFactorY()	Der Faktor, mit dem ein Dialog in der Y-Richtung ausgedehnt wird.

EnableReschedule akzeptiert ein boolesches Argument. Da ich weder eine Verwendung noch eine Dokumentation dieser Methode gefunden habe, kann ich nur spekulieren, wozu sie dient. Ich rate mal, dass OOo manchmal Dinge umorganisiert, vielleicht auch Ereignisse, und dass dabei manchmal, sagen wir während eines Callback, das Umorganisieren nicht erlaubt sein sollte. Mit EnableReschedule wird das interne Umorganisierungsflag ein- oder ausgeschaltet. Das Problem ist, dass es keine Methode gibt, den aktuellen Stand abzufragen, also kann man ihn auch nicht setzen und dann wiederherstellen. Es scheint aber so zu sein, dass die Standardeinstellung auf eingeschaltet steht.

Me gibt es in der .NET-Welt, um die Klasse oder Struktur der aktuellen Codeausführung zu referenzieren. Me ist nur im Compatiblity Mode nutzbar. Wenn der Kontext für Me nicht stimmt, wird ein Fehler generiert.

GetDialogZoomFactorX und GetDialogZoomFactorY scheinen den Skalierungsfaktor zu bestimmen, damit Vorschaubilder passend dargestellt werden. In den sehr wenigen Beispielen, die ich gefunden habe, sah der Code immer so ähnlich aus wie der folgende:

```
widthToUse  = GetDialogZoomFactorX(imageWidth) * imageWidth
heightToUse = GetDialogZoomFactorY(imageHeight) * imageHeight
```

Lesen Sie einmal hier nach: http://www.herger.net/staroffice/sbinteam/os/preview2.htm. Ich persönlich glaube ja, dass man diese Methoden nicht braucht, aber ich kann mich auch irren.

9.11. Neue Kompatibilitätsfunktionen

Ich habe Funktionen gefunden, die früher nicht dokumentiert waren. Bei all diesen muss CompatibilityMode auf True gesetzt sein, wenn Sie nicht ohnehin modulweit Option VBASupport 1 verwenden. Ich habe sie hier nicht weiter untersucht, da sie sehr spezialisiert sind.

Es handelt sich um eine Reihe von finanzmathematischen Funktionen: DDB, FV, IRR, MIRR, NPer, NPV, Pmt, PPmt, PV, Rate, SLN, SYD. Der LO-Hilfetext liefert detaillierte Beschreibungen.

9.12. Fazit

Die Informationen in diesem Kapitel sind sehr subtil. Wenn Sie Basic zum ersten Mal benutzen, sollten Sie das Kapitel noch einmal lesen, wenn Sie mehr Erfahrung gewonnen haben. Die Routinen zur Variablenüberprüfung sind nützlich, wenn zurückgegebene Objekte auszuwerten sind. Twips muss man kennen, wenn man Abmessungen und Bildgrößen bestimmt. Achten Sie darauf, veraltete (deprecated) Routinen zu vermeiden.

10. Universal Network Objects (UNO)

Intern basiert OOo auf Universal Network Objects (UNO). Dieses Kapitel behandelt die Subroutinen und Funktionen, die Basic im Zusammenhang mit UNO zur Verfügung stellt. Es geht dabei um Methoden zur Erzeugung und Überprüfung von Objekten, die für den Kern von OOo unerlässlich sind.

Bis hierher habe ich mich mit einfachen Einzelwerten wie Strings und Integers befasst. Grundsätzlich kann ein Objekt jedoch mehrere Datenelemente und Methoden enthalten. Zum Beispiel können Sie ein Dokument als eine Variable behandeln und Informationen über das Dokument (Daten oder Eigenschaften) abrufen sowie Methoden aufrufen, um das Dokument zu manipulieren.

In diesem Kapitel beginnen wir, über Dinge zu reden, die mit der eigentlichen Funktionalität von OOo zu tun haben – Dinge, die es Ihnen erlauben, OOo‘s internes Potenzial auszuschöpfen. Sie werden auch Einblicke in weitere Details über die Arbeitsweise von OOo gewinnen – all das, was Sie für die wirklich coolen Sachen brauchen.

Eine solche Schnittstelle zwischen den Funktionalitäten einer Anwendung und dem Zugriff darauf über eine Programmiersprache heißt API (englisch Application Programming Interface, wörtlich Anwendungsprogrammierschnittstelle).

Sind Sie Programmierer oder ansonsten technisch versiert? Wenn nicht, dann sparen Sie sich den Rest dieses Absatzes und lesen gleich den nächsten Tipp.

Sie lesen noch? Ausgezeichnet, also UNO...

- Ist das schnittstellenbasierte Komponentenmodell für OOo.
- Ermöglicht die Interoperabilität zwischen Programmiersprachen, Objektmodellen und Hardwarearchitekturen, entweder im Prozess oder über die Prozessgrenzen hinweg, sowohl im Intranet als auch im Internet.
- Unterstützt neue Sprachen durch das Hinzufügen von Sprachanbindungen, auch „Bridge“ oder „Adapter“ genannt. Ein solches Modell erleichtert die Unterstützung mehrerer Sprachen.
- Erlaubt die Einbindung von und den Zugriff auf UNO-Komponenten in jeder Programmiersprache mit kompletter Sprachanbindung.

Tipp Sie können auch ohne tiefes Verständnis der Universal Network Objects mächtige Makros schreiben. Betrachten Sie UNOs einfach als irgendwelche Objekte, die intern von OOo genutzt werden.

Ganz einfach gesagt: OOo verwendet für die inneren Abläufe Universal Network Objects. Mit UNOs ist es möglich, auf eine OOo-Instanz zuzugreifen, die auf einem anderen Rechner mit einem anderen Betriebssystem läuft. So benötigt man schon ein vages Verständnis der Universal Network Objects, weil die meisten internen Abläufe von OOo mit UNO eingebunden sind.

Tabelle 84 zeigt die Funktionen, die Basic für den Umgang mit UNO bereitstellt.

Tabelle 84. *Basic-Funktionen mit Bezug auf Universal Network Objects.*

Funktion	Beschreibung
BasicLibraries	Zugriff auf in einem Dokument gespeicherte Basic-Bibliotheken.
CreateObject(Objekttyp)	Kann jeden Standardtyp erzeugen, flexibler als CreateUnoStruct und CreateUnoService.
CreateUnoDialog()	Erzeugt einen definierten Dialog.
CreateUnoListener()	Erzeugt einen Beobachter (Fachbegriff Listener).
CreateUnoService()	Erzeugt einen Universal-Network-Object-Service.
CreateUnoStruct()	Erzeugt eine Universal-Network-Object-Struktur.
CreateUnoValue()	Erzeugt einen Universal-Network-Object-Wert.
DialogLibraries	Zugriff auf in einem Dokument gespeicherte Dialog-Bibliotheken.

Funktion	Beschreibung
EqualUNOObjects()	Prüft, ob zwei UNO-Objekte dieselbe Instanz referenzieren.
FindObject()	*Sucht ein benanntes Objekt. Nicht verwenden.*
FindPropertyObject()	*Findet eine namensbasierte Objekteigenschaft. Nicht verwenden.*
GetDefaultContext()	Kopiert den Standardkontext.
GetProcessServiceManager()	Referenz auf den Service-Manager.
GlobalScope	Bibliotheken auf Anwendungsebene.
HasUnoInterfaces()	Unterstützt ein Objekt bestimmte Schnittstellen?
IsUnoStruct()	Ist diese Variable ein Universal Network Object?
StarDesktop	Besondere Variable als Referenz auf das Desktop-Objekt.
ThisComponent	Besondere Variable als Referenz auf das aktuelle Dokument

10.1. Grundlegende Typen

Die von OpenOffice genutzten einfachen UNO-Datentypen unterscheiden sich von denen, die Basic verwendet. Es gibt aber Überschneidungen und Ähnlichkeiten.

10.1.1. Einfache UNO-Datentypen

Basic ist exzellent darin, zwischen den nativen Basic-Typen und den von OOo intern benötigten Typen zu konvertieren. Wenn Sie jedoch eine Methode eines Universal Network Objects aufrufen und Basic nicht weiß, welcher Typ es sein muss, könnte der Typ nicht passend konvertiert werden. Zum Beispiel erwartet die Methode setPropertyValue im Interface XPropertySet zwei Argumente – einen String als Namen der Eigenschaft und einen zu setzenden Wert. Der Typ des zu setzenden Werts hängt von der zu setzenden Eigenschaft ab. Wenn es ein Problem sein sollte, den korrekten Typ zu erzeugen, verwenden Sie die Funktion CreateUnoValue (s. Listing 194), um eine Referenz zu einem Universal Network Object zu erzeugen, das den passenden Typ enthält. Mir ist ein solches Problem noch nie untergekommen, also machen Sie sich keinen Kopf über Ihre Argumente. Sie können sich mit an Sicherheit grenzender Wahrscheinlichkeit darauf verlassen, dass Basic das Richtige tut.

Listing 194. *Mit CreateUnoValue wird eine Referenz zu einem internen UNO-Wert erzeugt.*

```
Dim v
v = CreateUnoValue("unsigned long", 10)
v = CreateUnoValue("string", "Hallo")
v = CreateUnoValue("byte", 10)                'Ein Byte ist im Bereich von -128 bis 127
v = CreateUnoValue("[]byte", Array(3, 2, 1)) 'Man kann sogar Arrays erzeugen
'v = CreateUnoValue("Byte", 10)               'Fehler: "No such element"
'v = CreateUnoValue("byte", 1000)             'Fehler: "Value is out of range"
'v = CreateUnoValue("uint64", 10)             'Fehler: "No such element"
```

Das erste Argument zu CreateUnoValue ist der zu erzeugende Datentyp. Es muss einer der einfachen UNO-Datentypen sein (s.a. https://wiki.openoffice.org/wiki/Documentation/DevGuide/ProUNO/-Simple_Types).

Tabelle 85. *Einfache UNO-Typen.*

TypeName	Beschreibung
void	Kein Wert.
char	Einfaches Unicode-Textzeichen (UTF-16).
boolean	Boolescher Wert (True oder False).
byte	8-Bit-Ganzzahl mit Vorzeichen (von -128 bis 127, inklusive).

TypeName	Beschreibung
short	16-Bit-Ganzzahl mit Vorzeichen (von -32768 bis 32767, inklusive).
unsigned short	*Veraltet*: 16-Bit-Ganzzahl ohne Vorzeichen.
long	32-Bit-Ganzzahl mit Vorzeichen (von -2147483648 bis 2147483647, inklusive).
unsigned long	*Veraltet*: 32-Bit-Ganzzahl ohne Vorzeichen.
hyper	64-Bit-Ganzzahl mit Vorzeichen (von -9223372036854775808 bis 9223372036854775807, inklusive).
unsigned hyper	*Veraltet*: 64-Bit-Ganzzahl ohne Vorzeichen.
float	32-Bit-Fließkommazahl (einfache Genauigkeit gemäß IEC 60559).
double	64-Bit-Fließkommazahl (doppelte Genauigkeit gemäß IEC 60559).
string	String von Unicode-Textzeichen.
type	Metatyp, der alle UNO-Typen beschreibt.
any	Spezieller Typ, der Werte aller anderen Typen aufnehmen kann (außer any).

Bei der Verwendung der Namen ist zu beachten, dass sie alle klein geschrieben werden müssen. Ihnen darf ein Paar eckiger Klammern vorangehen als Zeichen für ein Array. Der von CreateUnoValue zurückgegebene Wert ist für Basic nicht nutzbar, er wird nur für die Interna von OOo gebraucht. Mit anderen Worten, erzeugen Sie keinesfalls einen Typ „byte“, um ihn dann als Zahl benutzen zu wollen.

Listing 195. *Test der von CreateUnoValue unterstützten Typen.*

```
Sub TestCreateUnoValues
  Dim typeNames
  Dim v
  Dim i%

  typeNames = Array("void", "boolean", "char", "byte", "string", _
      "short", "unsigned short", "long", "unsigned long", _
      "hyper", "unsigned hyper", "float", "double", "any")
  ' Die Liste der Namen stammt von
  ' cppu/source/typelib/typelib.cxx
  For i = LBound(typeNames) To UBound(typeNames)
    v = CreateUnoValue(typeNames(i), 65)
    ' Man kann den Wert nicht direkt nutzen, weil es ein UNO-Typ ist,
    ' der nur dazu taugt, Argumente zu UNO zu übergeben.
    ' Dinge wie "Print v" oder "CStr(v)" gehen schief.
  Next
End Sub
```

10.1.2. Konstanten

In UNO ist eine Konstante ein benannter Wert eines gültigen einfachen UNO-Typs. Da die Vielfalt an Werten und Typen groß ist, werden Konstanten häufig in Gruppen von Varianten für bestimmte Einstellungen zusammenfasst.

Zum Beispiel: Innerhalb des API-Moduls com.sun.star.awt ist die Konstantengruppe ImageAlign angelegt, in der die vier Möglichkeiten der Bildausrichtung als Konstanten definiert sind. Der komplette Gruppenname ist also com.sun.star.awt.ImageAlign, und die einzelnen Konstanten haben jeweils einen Namen und einen Wert.

Gruppenname:	com.sun.star.awt.ImageAlign
Konstantendefinitionen:	LEFT = 0 TOP = 1 RIGHT = 2 BOTTOM = 3

Sie kennen eine solche Strukturierung schon von den benutzerdefinierten Typen in Basic, bei denen der Abruf eines Wertes auf dieselbe Weise erfolgt wie bei den UNO-Konstanten. Dem String des kompletten Gruppennamens com.sun.star.awt.ImageAlign fügen Sie erst einen Punkt, dann den gewünschten Namen an. Zum Beispiel verwenden Sie die RIGHT-Konstante in der Form com.sun.star.awt.ImageAlign.RIGHT, s. Listing 196. Der Name einer Konstanten sagt dem Leser des Basic-Codes, um was es geht. Die direkte Verwendung der numerischen Werte würde, wenn auch erlaubt, Ihren Code verschleiern, ihn also weniger lesbar machen.

Achtung Die Groß- und Kleinschreibung der Namen von Konstanten muss bei ihrem ersten Gebrauch beachtet werden. Wenn die beiden Zeilen im Listing 196 vertauscht würden, entstünde ein Laufzeitfehler, weil zuerst die Version mit der falschen Schreibung gesucht würde, aber nicht gefunden werden konnte.

Listing 196. *Groß-/Kleinschreibung der Namen von Konstanten spielt bei ihrem zweiten(!) Gebrauch keine Rolle mehr.*

```
Print com.sun.star.awt.ImageAlign.RIGHT '2
Print COM.SUN.STAR.awt.ImageALIGN.rigHt '2 - Funktioniert, weil bereits bekannt
```

Wenn Sie innerhalb Ihrer Makros die Werte einer Konstantengruppe mehrfach verwenden müssen, können Sie den Gruppennamen einer Objektvariablen zuweisen, um den Schreibaufwand zu verringern und den Code übersichtlicher zu gestalten:

```
Dim oPropConcept
oPropConcept = com.sun.star.beans.PropertyConcept
MsgBox oPropConcept.DANGEROUS   '1
MsgBox oPropConcept.PROPERTYSET '2
```

10.1.3. Enumerationen

Eine Enumeration hat den UNO-Typ Enum. Es ist eine geordnete Liste einer oder mehrerer Bezeichner (Enumeratoren), die Long-Werte des Enum-Typs repräsentieren. Im Allgemeinen sind die Werte aufsteigend gezählt, beginnend mit 0 und plus 1 für jeden weiteren Wert. Verwenden Sie jedoch immer nur die Enumeratoren, niemals deren Werte direkt.

Enumerationen sind so aufgebaut wie Konstantengruppen. Zum Beispiel: Innerhalb des API-Moduls com.sun.star.uno ist die Enumeration TypeClass angelegt.

Enumerationsname:	com.sun.star.uno.TypeClass
Enumerationen:	VOID CHAR BOOLEAN BYTE ...

Einen einzelnen Wert referenzieren Sie also durch com.sun.star.uno.TypeClass.BOOLEAN. Ebenso wie bei Konstantengruppen können Sie auch den Enumerationstyp in einer Variablen referenzieren, um Ihren Code schlanker zu gestalten.

10.1.4. Strukturen

Strukturen (so genannten Structs) verhalten sich wie benutzerdefinierte Datentypen. Structs wiederum werden zu komplexeren UNO-Objekten zusammengefasst. Ein Struct enthält Eigenschaften, aber keine Methoden.

Ein Struct bietet einen Weg, mehr als einen Wert in einer einzigen Variablen zu speichern. Der Zugriff auf einen Wert in einem Struct geschieht über einen Namen, der vom Programmierer beim Entwurf des Structs vergeben wird. Ein vielfach verwendetes Struct ist PropertyValue, dessen Hauptzweck es ist, einen String als Namen (Name) und einen Variant als Wert (Value) zu haben. Das folgende Listing zeigt, wie ein PropertyValue-Struct erzeugt und dann Name und Value festgelegt werden. Auf die Eigenschaften greift man dadurch zu, dass man einen Punkt zwischen den Variablen- und den Eigenschaftsnamen setzt.

Listing 197. *Mit Dim As New wird ein UNO-Struct erzeugt.*

```
Dim aProp As New com.sun.star.beans.PropertyValue
aProp.Name = "Vorname"        'Setzt die Eigenschaft Name
aProp.Value = "Kunibert"      'Setzt die Eigenschaft Value
```

Tipp OOo-Objekte haben lange Namen wie com.sun.star.beans.PropertyValue. Im laufenden Text kürze ich für gewöhnlich die Namen ab und schreibe PropertyValue. In Ihren Makros müssen Sie aber den vollen Namen verwenden!

Vor dem Gebrauch eines UNO-Struct muss man es erzeugen (oder referenzieren). Die üblichste Methode, ein UNO-Struct zu erzeugen, geht über Dim As New, s. Listing 197. Man kann mit Dim auch ein Struct-Array erzeugen.

Listing 198. *Mit Dim As New wird ein Array von UNO-Structs erzeugt.*

```
Dim aProp(4) As New com.sun.star.beans.PropertyValue
aProp(0).Name  = "Vorname"       'Setzt die Eigenschaft Name
aProp(0).Value = "Hildegunde"    'Setzt die Eigenschaft Value
```

Mit der Funktion CreateUnoStruct wird ein UNO-Struct erst zu dem Zeitpunkt erzeugt, wenn es gebraucht wird, und nicht schon viel früher zum Zeitpunkt der Deklarierung. Wenn man ein UNO-Struct dynamisch erzeugt, hat man die Möglichkeit, den Namen des Structs zur Laufzeit zu bestimmen statt zur Kompilierungszeit. Wie man zur Laufzeit einen Namen vergibt, zeigen Listing 199 und Listing 202. Im Listing 197 wird der Name zur Kompilierungszeit bestimmt.

Listing 199. *Mit CreateUnoStruct wird ein UNO-Struct zur Laufzeit erzeugt.*

```
Dim aProp
aProp = CreateUnoStruct("com.sun.star.beans.PropertyValue")
aProp.Name = "Vorname"        'Setzt die Eigenschaft Name
aProp.Value = "Andrew"        'Setzt die Eigenschaft Value
```

Die Anweisung With vereinfacht den Prozess, die Eigenschaften eines Struct zu setzen.

Listing 200. *Mit With wird das Setzen der Eigenschaften eines Structs vereinfacht.*

```
Dim aProp(4) As New com.sun.star.beans.PropertyValue
With aProp(0)
  .Name = "Vorname"         'Setzt die Eigenschaft Name
  .Value = "Kunibert"       'Setzt die Eigenschaft Value
End With
```

Früher war es nur mit der Funktion CreateUnoStruct möglich, ein UNO-Struct zu erzeugen. Seit der Einführung der „Dim As New“-Syntax wird CreateUnoStruct weniger gebraucht. Mit der umfassenderen Funktion CreateObject kann man Instanzen aller Typen erzeugen, die Basic intern mit dem Factory-Mechanismus unterstützt. Dazu gehören auch benutzerdefinierte Typen.

Listing 201. *Erzeugt einen benutzerdefinierten Typ mit CreateObject oder Dim As.*

```
Type PersonType
  FirstName As String
  LastName As String
End Type

Sub ExampleCreateNewType
  Dim Person As PersonType
  Person.FirstName = "Andrew"
  Person.LastName  = "Pitonyak"
  PrintPerson(Person)
  Dim Me As Object
  Me = CreateObject("PersonType")
  Me.FirstName = "Andy"
  Me.LastName  = "Pitonyak"
  PrintPerson(Me)
End Sub

Sub PrintPerson(x)
  Print "Person = " & x.FirstName & " " & x.LastName
End Sub
```

Tipp Für einen benutzerdefinierten Typ funktionieren sowohl „Dim As New“ als auch „Dim As“. Für ein UNO-Struct muss man allerdings „Dim As New“ nehmen.

Die Funktion CreateObject erwartet dieselben Argumente wie CreateUnoStruct, funktioniert aber mit allen unterstützten Typen, CreateUnoStruct hingegen nur mit UNO-Structs. Es gibt daher keinen Grund mehr, CreateUnoStruct anstelle von CreateObject zu verwenden.

Listing 202. *Ein UNO-Struct wird mit CreateObject erzeugt.*

```
Dim aProp
aProp = CreateObject("com.sun.star.beans.PropertyValue")
aProp.Name = "Vorname"       'Setzt die Eigenschaft Name
aProp.Value = "Kunibert"     'Setzt die Eigenschaft Value
```

Tipp CreateObject bietet größere Flexibilität als CreateUnoStruct bei der dynamischen Erzeugung von benannten Objekten.

Ich habe ein kleines Testprogramm geschrieben, das 20000 Structs erzeugte. CreateUnoStruct und CreateObject brauchten etwa gleich viel Zeit. Dim As New war jedoch 500 Systemticks schneller. Das ist vielleicht nützlich, wenn Sie ein Makro so schnell wie möglich laufen lassen wollen.

Die Funktion TypeName zeigt an, dass ein UNO-Struct ein Objekt ist. Mit IsUnoStruct ermitteln Sie, ob eine Variable ein UNO-Struct ist.

Listing 203. *Mit IsUnoStruct wird geprüft, ob ein Objekt ein UNO-Struct ist.*

```
Sub TestTypeName()
  Dim aProp As New com.sun.star.beans.PropertyValue
  Print TypeName(aProp)      'Object
  Print IsUnoStruct(aProp)   'True
End Sub
```

10.2. Services, Interfaces und Co.

Die Zugriffe von Programmiersprachen wie Basic auf die OOo-Funktionalitäten werden über Objekte gesteuert, die nach den Prinzipien der objektorientierten Programmierung strukturiert sind. In der

API ist eine gewaltige Anzahl an Objekten definiert, beispielsweise ein Dokument, der gesamte Text mit all seinen Eigenheiten, ein Tabellenblatt, eine Zelle oder ein Dialogfenster. Solche Objekte können Eigenschaften besitzen oder über spezielle Funktionen, sogenannte Methoden, gesteuert werden. Außerdem können Eigenschaften und Methoden hierarchisch an untergeordnete Objekte vererbt werden. Um eine Eigenschaft oder eine Methode aufzurufen, wird dem Objektnamen ein Punkt und der Name eben dieser Eigenschaft oder Methode angehängt, zum Beispiel „oDialog.execute()“.

StarBasic ist im Prinzip keine objektorientierte, sondern eine prozedurale Programmiersprache, mit der Ausnahme von UNO-Objekten. Diese lassen sich auch in Basic mit der typischen Punktsyntax verwenden.

Die UNO-Objekte existieren in der API als abstrakte Definitionen. In der praktischen Anwendung braucht Basic konkreten Speicherplatz. Ein Objekt muss also unter einem Namen als Variable deklariert werden. Für Objekte, deren Speicherplatz nicht von vornherein festgelegt werden kann, ist der Variablentyp „Variant“ vorgesehen. Vielfach sieht man auch Makros, in denen UNO-Objekte als Typ „Object“ deklariert sind. Das ist zwar nicht so gedacht, aber in der Praxis völlig risikolos (s. Abschnitt 10.6. . Typdefinition Object oder Variant). Die konkrete Variable eines abstrakten Objekts nennt man eine Instanz. Die Anzahl der möglichen Instanzen eines Objekts in einem Makro ist im allgemeinen nicht begrenzt. Allein Singletons sind auf eine einzige konkrete Instanz innerhalb der Laufzeit einer Anwendung limitiert.

Die wesentlichen Objekte zum Zugriff auf OOo-Inhalte sind Interfaces und Services.

10.2.1. UNO-Interfaces

Ein Interface definiert, wie etwas mit seiner Umgebung interagiert. Ein UNO-Interface gleicht einer Gruppe von Subroutinen- und Funktionen-Deklarierungen. Parameter und Rückgabetypen werden zusammen mit der Funktionalität spezifiziert. UNO-Interfaces sind unabhängig von Objekten. Viele werden von einer Reihe von Objekten eingebunden, die gleiche oder ähnliche Funktionalitäten benötigen.

Sie können über ein Interface Daten aus einem Objekt holen, Daten in einem Objekt setzen oder einem Objekt sagen, es solle etwas tun. Das Interface zeigt, wie ein Objekt genutzt wird, aber es sagt nichts darüber aus, wie es strukturiert ist. Wenn ein Interface zum Beispiel eine Methode getHeight hat, die eine Ganzzahl zurückgibt, so ist es nur natürlich anzunehmen, dass das Objekt eine ganzzahlige Eigenschaft namens Height hat. Es kann jedoch sein, dass die Höhe ein abgeleiteter Wert ist oder aus anderen Eigenschaften berechnet wird. Das Interface gibt nicht an, wie die Höhe ermittelt wird, nur dass sie verfügbar ist. Ein UNO-Struct hingegen enthält Eigenschaften, auf die direkt zugegriffen werden kann. Die interne Struktur ist nicht verborgen.

Einer Interface-Methode muss die Argumentliste in runden Klammern angefügt werden. Wenn die Liste leer ist, sind die Klammern dennoch erforderlich.

Methoden, die auf Eigenschaften zugreifen, beginnen mit get… zum Lesen und set… zum Schreiben der Eigenschaft

```
t = ObjectVarName.getText()   'Lesen
ObjectVarName.setText(t)      'Schreiben
```

Basic bietet wie auch Python das Bonbon, diese Methoden wie Eigenschaften zu verwenden, und zwar immer dann, wenn die Methode getSomeProperty() ohne Parameter definiert ist und die Methode setSomeProperty(SomeType aValue) genau ein Argument erfordert:

```
t = ObjectVarName.Text        'Lesen
ObjectVarName.Text = t        'Schreiben
```

Tipp UNO-Interface-Namen starten mit dem Großbuchstaben X.
Wenn Sie wissen wollen, welche Methoden ein Objekt unterstützt, prüfen Sie die Interfaces.

UNO-Interface-Namen starten mit dem Großbuchstaben X, woran sie leicht zu erkennen sind. Zum Beispiel spezifiziert das Interface com.sun.star.text.XTextRange einen Textbereich mit einer Start- und einer Endposition. Objekte, die das Interface XTextRange einbinden, werden auch verwendet, um die Positionen eines Objekts in einem Textdokument zu ermitteln. Die Start- und Endpositionen dürfen auch identisch sein. In der Tabelle 86 finden Sie die vom Interface XTextRange definierten Methoden.

Tabelle 86. *Methoden im Interface com.sun.star.text.XTextRange.*

Methode	Beschreibung
getText()	Gibt das Interface com.sun.star.text.XText zurück, das dieses XTextRange enthält.
getStart()	Gibt ein com.sun.star.text.XTextRange zurück, das nur die Startposition referenziert.
getEnd()	Gibt ein com.sun.star.text.XTextRange zurück, das nur die Endposition referenziert.
getString()	Gibt einen String zurück, der den reinen Text dieses Textbereichs enthält.
setString(str)	Setzt den String für diesen Textbereich. Ersetzt den vorhandenen Text und löscht alle Formate.

Tipp Ein UNO-Interface kann von einem anderen abgeleitet sein. Jedes UNO-Interface muss im Endeffekt von com.sun.star.uno.XInterface abgeleitet sein.

Ein neues UNO-Interface kann von einem anderen abgeleitet sein. Das machen nicht Sie, sondern das hat der Entwickler des Interface getan. Das abgeleitete Interface unterstützt alle Methoden, die in dem Interface definiert sind, von dem es abstammt. Zum Beispiel erweitert com.sun.star.text.XTextCursor das Interface XTextRange dahin, dass es die Bereichsänderung ermöglicht, was ja auch sinnvoll ist, wenn man bedenkt, was man alles mit mit einem Cursor machen kann. Jedes Objekt, in dem das Interface XTextCursor eingebunden ist, unterstützt automatisch die Methoden der Tabelle 86 und zusätzlich die neuen Methoden des Interface XTextCursor.

Die wesentlichen Punkte bisher in Bezug auf Interfaces sind:

- Ein Interface definiert Methoden, das ist das, was ein Objekt tun kann. Das schließt das Lesen und Setzen interner Eigenschaften ein.
- Ein Interface kann aus einem anderen Interface abgeleitet sein.
- Der letzte Bestandteil eines kompletten Interfacenamens beginnt mit einem X.

Der Zugriff auf UNO-Objektmethoden geschieht über die eingebundenen Interfaces („exported interfaces“ im Sprachgebrauch der API). In vielen Programmiersprachen wie Java und C++ benötigen Sie ein wenig UNO-Magie, um das korrekte Interface hervorzuholen, bevor Sie die in diesem Interface definierten Methoden aufrufen können. Basic verbirgt diese Details vor Ihnen, so dass Sie wie die Eigenschaften so auch die Methoden direkt aufrufen können, indem Sie sie mit einem Punkt an den Objektnamen schreiben.

Viele Interfaces werden nicht nur in einem Objekt verwendet.

Tipp Basic verbirgt viele der komplizierten Details. Somit ist es in den meisten Fällen einfacher, ein Basic-Programm zu schreiben als ein Java-Script.

10.2.2. UNO-Services

Ein Service ist die abstrakte Definition eines Objekts, die durch eine Kombination von Methoden und Eigenschaften eine nützliche Funktionalität verkapselt. Die Methoden werden von UNO-Interfaces bereitgestellt. Sie definieren, wie ein Objekt mit der Außenwelt interagiert. Ein UNO-Struct definiert eine Sammlung von Daten. Ein UNO-Service kombiniert beides. Genau wie ein UNO-Inter-

face spezifiziert ein UNO-Service nicht die konkrete Ausgestaltung. Es legt nur fest, wie man mit dem Objekt umgeht.

Fast jedes UNO-Objekt wird von einem Service definiert, daher werden UNO-Objekte Services genannt. Streng genommen ist ein „Service“ jedoch die Objektdefinition. Das wirkliche Objekt eines UNO-Objekts ist die Instanz, die durch die Definition des Service erzeugt wurde. Ein Service kann mehrere Services und Interfaces enthalten. Ein Interface definiert gewöhnlich ein Aspektbündel eines Service und hat daher normalerweise einen geringeren Umfang.

Die Namen vieler Services gleichen denen von Interfaces. Beispielsweise wird vom Service TextCursor unter anderen ein Interface mit dem Namen XTextCursor exportiert. Ein Interface oder eine Eigenschaft können für einen Service als optional deklariert sein. Das Interface XWordCursor ist als optional für den Service TextCursor gekennzeichnet. Daraus folgert, dass nicht alle Textcursors das Interface XWordCursor unterstützen. In der Praxis werden Sie lernen, welche Textcursors welche Interfaces unterstützen, und sie dann einfach nutzen.

Es gibt im Prinzip zwei Wege, einen Service zu erzeugen.

- Erstellen Sie eine Referenz auf ein existierendes Objekt, nehmen Sie zum Beispiel die erste Texttabelle des aktuellen Dokuments.
- Fordern Sie eine Service-Factory (factory = Fabrik) auf, eine Instanz eines Objekts zu erzeugen. Wenn Sie zum Beispiel in ein Dokument eine neue Texttabelle einfügen wollen, fordern Sie vom Dokument eine neue leere Tabelle an, die Sie dann konfigurieren und in das Dokument einfügen.

Eine Service-Factory gibt ein Objekt gemäß dem Servicenamen zurück. Die Haupt-Objekt-Factory für OOo ist der Prozess-Servicemanager. Der Factory wird der Servicename mitgeteilt, und sie entscheidet, was zurückgegeben wird. Eine Factory mag eine ganz neue Objektinstanz oder eine schon existierende zurückgeben. Mit GetProcessServiceManager() erhalten Sie eine Referenz auf den Prozess-Servicemanager. Mit der Methode CreateInstance des Prozess-Servicemanagers wird ein Service erzeugt, s. Listing 204.

Listing 204. *Über den Prozess-Servicemanager einen Service erzeugen.*

```
Sub ManagerCreatesAService
  Dim vFileAccess          'Objekt Dateizugriff
  Dim s As String
  Dim vManager
  vManager = GetProcessServiceManager()
  vFileAccess = vManager.createInstance("com.sun.star.ucb.SimpleFileAccess")
  s = vFileAccess.getContentType("http://www.pitonyak.org/AndrewMacro.odt")
  Print s
End Sub
```

Der Code im Listing 204 referenziert den Prozess-Servicemanager, erzeugt eine Instanz des Service SimpleFileAccess und nutzt diesen Service. Die Funktion CreateUnoService ist eine Abkürzung für das Erstellen eines UNO-Service, s. Listing 205. Dieser Code demonstriert die Funktion CreateUnoService und zeigt, dass es damit einfacher geht, als erst einen Servicemanager zu erzeugen. Listing 205 zeigt darüber hinaus eine nützliche Funktionalität, nämlich die Dateiauswahl über einen Dialog.

Listing 205. *Dateiauswahl vom Plattenspeicher.*

```
Sub PrintChosenFileName
  Dim sFileNameURL$
  sFileNameURL = ChooseAFileName
  MsgBox "URL: " & sFileNameURL & Chr(10) & Chr(10) & _
         "Pfad: " & ConvertFromURL(sFileNameURL)
End Sub
```

```
Function ChooseAFileName() As String
  Dim vFileDialog         'Instanz des  FilePicker
  Dim vFileAccess         'Instanz des Service SimpleFileAccess
  Dim iAccept As Integer  'Rückgabe vom FilePicker
  Dim sInitPath As String 'Der Startpfad
  'Achtung: Die folgenden Services müssen in dieser Reihenfolge
  'aufgerufen werden, sonst wird Basic den vFileDialog nicht wieder entfernen.
  vFileDialog = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")
  vFileAccess = CreateUnoService("com.sun.star.ucb.SimpleFileAccess")
  'Jetzt wird der Startpfad gesetzt.
  sInitPath = ConvertToUrl(CurDir)
  If vFileAccess.exists(sInitPath) Then
    vFileDialog.setDisplayDirectory(sInitPath)
  End If

  iAccept = vFileDialog.execute()       'Der Dateiauswahldialog wird ausgeführt.
  If iAccept = 1 Then                   'Prüfung des Rückgabewerts des Dialogs.
    ChooseAFileName = vFileDialog.Files(0) 'Rückgabe des Dateinamens, falls
                                           'der Dialog nicht abgebrochen wurde.
  End If
  vFileDialog.dispose()                 'Der Dialog wird entfernt.
End Function
```

Die Auswahl eines Verzeichnisses funktioniert ähnlich. Der hier verwendete DefaultContext wird im Abschnitt 10.2.3. Kontext erläutert, Singletons im Abschnitt 10.2.4. Singletons.

Listing 206. *Verzeichnisauswahl vom Plattenspeicher.*

```
REM sInPath benennt das Startverzeichnis. Wenn kein Startverzeichnis
REM angegeben ist, wird das Standardarbeitsverzeichnis des Nutzers genommen.
REM Das ausgewählte Verzeichnis wird als URL zurückgegeben.
Function ChooseADirectory(Optional sInPath$) As String
  Dim oDialog         'Instanz des Service FolderPicker - Verzeichnisauswahl
  Dim oSFA............'Instanz des Service SimpleFileAccess - Dateiinformationen
  Dim s As String
  Dim oPathSettings   'Instanz des Service PathSettings - Ooo-Pfade
  Dim oContext

  oDialog = CreateUnoService("com.sun.star.ui.dialogs.FolderPicker")
  oSFA = CreateUnoService("com.sun.star.ucb.SimpleFileAccess")

  If IsMissing(sInPath) Then
    oContext = GetProcessServiceManager().DefaultContext
    oPathSettings = _
           oContext.getValueByName("/singletons/com.sun.star.util.thePathSettings")
    If IsEmpty(oPathSettings) Then
      'AOO nutzt den älteren Singleton-Typ:
      oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")
    End If
    oDialog.setDisplayDirectory(oPathSettings.Work) 'Das Benutzerarbeitsverzeichnis
                                                    'als Ausgangspunkt.
  ElseIf oSFA.exists(sInPath) Then
    oDialog.setDisplayDirectory(sInPath)
  Else
    s = "Verzeichnis '" & sInPath & "' existiert nicht."
    If MsgBox(s, 33, "Fehler") = 2 Then Exit Function
  End If
```

```
  If oDialog.execute() = 1 Then
    ChooseADirectory() = oDialog.getDirectory()
  End If
End Function
```

Tipp

Die Dialoge FilePicker und FolderPicker können entweder mit den Standard-Dateiauswahldialogen des Betriebssystems oder mit spezifischen OOo-Dialogen eingesetzt werden. Die Dialoge des Betriebssystems bieten weniger Funktionalität, so kann man zum Beispiel nicht das Startverzeichnis vorwählen. Dazu kreuzen Sie über das Menü **Extras > Optionen > LibreOffice > Allgemein** „LibreOffice-Dialoge verwenden“ an.

Wenn Sie eine originale AOO-Version unter Linux verwenden, haben Sie keine Auswahl. Es werden nur die OOo-spezifischen Dialoge verwendet.

Das folgende Listing 207 demonstriert den UNO-Service TextSearch, mit dem Strings mit verschiedenen Modi durchsucht werden können, auf die direkte Art oder mit regulären Ausdrücken oder nach Ähnlichkeit, vorwärts oder rückwärts, mit oder ohne Teilstringersetzung. Für CJK-Schriften (chinesisch, japanisch, koreanisch) sind diverse Transliterationsmodi vorgesehen. Das Beispiel zeigt eine Suche mit einem einfachen regulären Ausdruck.

Listing 207. *Nutzung des Service TextSearch.*

```
Sub StringTextSearch
  Dim oTextSearch               'Service TextSearch
  Dim sStrToSearch As String    'Zu durchsuchender String
  Dim sMatchString As String    'Gefundener String
  Dim aSearchResult             'Suchergebnis als Variant
                                'Wird vom Suchprozess zurückgegeben als
                                'Struct com.sun.star.util.SearchResult
  Dim i As Long                 'Trefferzähler
  Dim iMatchStartPos As Long    'Startposition des jeweiligen Treffers
  Dim iMatchEndPos As Long      'Endposition des jeweiligen Treffers
  Dim iMatchLen As Long         'Anzahl der Zeichen im Treffer
  Dim aSearchOpt As New com.sun.star.util.SearchOptions
                                                'Struct für Suchoptionen
  Dim s As String               'Ausgabestring

  Dim deLocale As New com.sun.star.lang.Locale  'Struct für Gebietsschema
  deLocale.Language = "de"      'zur Absicherung der korrekten Umsetzung
  deLocale.Country = "DE"       'der Groß- und Kleinschreibung

  oTextSearch = CreateUnoService("com.sun.star.util.TextSearch")
  s = ""

  With aSearchOpt  'Suchoptionen
    .Locale = deLocale

    'algorithmType unterstützt ABSOLUTE (= buchstabengenau),
    'REGEXP (= regulärer Ausdruck) und APPROXIMATE (= Ähnlichkeitssuche
    'mit dem Algorithmus "Weighted Levenshtein Distance").
    'Wir suchen mit einem regulären Ausdruck:
    .algorithmType = com.sun.star.util.SearchAlgorithms.REGEXP

    .searchString = "a+"   'Gesucht wird ein a oder mehrere a's in Folge.

    'Zur Nichtbeachtung der Groß- und Kleinschreibung funktioniert
    'com.sun.star.i18n.TransliterationModulesNew.IGNORE_CASE
    'nur für den Service Transliteration.
    'Für den Service TextSearch funktioniert aber
```

```
    'die Umwandlung von Groß- zu Kleinbuchstaben:
    .transliterateFlags = _
             com.sun.star.i18n.TransliterationModulesNew.UPPERCASE_LOWERCASE
  End With

  oTextSearch.setOptions(aSearchOpt)

  'Sowohl für die Argumente Start- und Endposition der Suchmethoden als auch
  'bei den zurückgegebenen Arrays startOffset und endOffset gilt:
  'Alle Positionsangaben sind Offsets und sind 0-basiert:
  'Für die Vorwärtssuche searchForward:
  'Start = Position des ersten Zeichens der Suche bzw. des Treffers
  'Ende = Position hinter(!) dem letzten Zeichen der Suche bzw. des Treffers
  'Für die Rückwärtssuche searchBackward:
  'Start = Position hinter(!) dem letzten Zeichen der Suche bzw. des Treffers
  'Ende = Position des ersten Zeichens der Suche bzw. des Treffers

  sStrToSearch = "aaa hello AAA"
  'Vorwärtssuche:
  aSearchResult = oTextSearch.searchForward(sStrToSearch, 0, Len(sStrToSearch))

  With aSearchResult
    'Die Eigenschaft subRegExpressions enthält die Anzahl der Treffer,
    'ohne Treffer den Wert 0. Reguläre Ausdrücke bieten die Möglichkeit, in einem
    'Suchvorgang mehrere Treffer zu finden. Deren jeweilige Start- und Endpositionen
    'stehen in den Arrays startOffset und endOffset.
    'Mit unserer einfachen Suche wird es aber nur jeweils einen Treffer geben.
    Do While .subRegExpressions > 0
      For i = LBound(.startOffset) To UBound(.startOffset)
        iMatchStartPos = .startOffset(i)
        iMatchEndPos = .endOffset(i)
        iMatchLen = iMatchEndPos - iMatchStartPos
        sMatchString = Mid(sStrToSearch, iMatchStartPos + 1, iMatchLen)
        s = s & "Treffer " & i + 1 & " von " & _
              Ubound(.startOffset) + 1 & " => " & sMatchString & Chr$(10)
      Next
      aSearchResult = oTextSearch.searchForward(sStrToSearch, iMatchEndPos, _
                                                Len(sStrToSearch))
    Loop
  End With

  MsgBox s
End Sub
```

Der Code im Listing 205 erzeugt zwei UNO-Services mit Hilfe der Funktion CreateUnoService. Es gibt aber auch Situationen, in denen Sie den Servicemanager brauchen, denn der verfügt über Methoden, einen Service mit Argumenten zu erzeugen, CreateInstanceWithArguments, und eine Liste aller unterstützten Services zu erstellen, getAvailableServiceNames(). Der Code im Listing 208 gibt eine Liste der unterstützten Servicenamen aus. Auf meinem Rechner, in der LO-Version 6.4.5.2, sind es 998 Services, verglichen mit 562 der OOo-Version 1. Die Zugriffe auf das Textdokument werden im einzelnen im Kapitel 14. Textdokumente vorgestellt.

Listing 208. *Der Servicemanager unterstützt Services.*

```
Sub HowManyServicesSupported
  Dim oDoc          ' Dokument zur Aufnahme der Servicenamen.
  Dim oText         ' Das Textobjekt des Dokuments.
```

```
  Dim oSD           ' SortDescriptor (Sortieroptionen), erzeugt für das Dokument.
  Dim oCursor       ' Textcursor für die Sortierung.
  Dim i%            ' Indexvariable.
  Dim sServices     ' Array der Service-Namen.

  sServices = GetProcessServiceManager().getAvailableServiceNames()
  Print "Der Servicemanager unterstützt "; UBound(sServices); " Services"

  ' Erzeugt ein Textdokument.
  oDoc = StarDesktop.loadComponentFromURL("private:factory/swriter", _
                                          "_blank", 0, Array())
  oText = oDoc.getText()

  ' Gibt die Servicenamen Zeile für Zeile im Textdokument aus.
  For i = LBound(sServices) To UBound(sServices)
    oText.insertString(oText.getEnd(), sServices(i), False)
    ' Keine Leerzeile am Ende.
    oText.insertControlCharacter(oText.getEnd(), _
            com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
  Next
  ' Alphabetische Sortierung der Servicenamen.
  oCursor = oDoc.Text.createTextCursorByRange(oDoc.Text)
  oSD = oCursor.createSortDescriptor()
  oCursor.sort(oSD)

  oText.insertString(oText.getStart(), _
           "Der Servicemanager unterstützt " & UBound(sServices) & " Services", False)
End Sub
```

Es lohnt sich, einen Blick auf die unterstützten Services zu werfen. Sie erhalten einen Einblick in die verfügbare Funktionalität, die Sie erforschen können. Ein gutes Beispiel dafür finden Sie im Listing 209. Ein Objekt, das in ein Dokument eingefügt werden soll, muss von eben diesem Dokument erzeugt werden. Mit den Methoden aus dem Listing 208 können Sie erkennen, welche Objekttypen das Dokument erzeugen kann.

Listing 209. *Auflistung der Objekte, die ein Dokument erzeugen kann.*

```
REM Gibt die Objekt-/Servicetypen aus, die ein Dokument erzeugen kann.
REM Wenn das Dokument (oDoc) fehlt, wird das aktuelle Dokument verwendet.
REM Wenn nameFilter angegeben ist, werden die Servicenamen ausgegeben, die diesen
REM String enthalten, unabhängig von Groß- oder Kleinschreibung.
REM Die Servicenamen werden in ein neu erstelltes Textdokument geschrieben.
Sub TypesDocCanCreate(Optional oDoc, Optional nameFilter$)
  Dim allNames    ' Liste aller Namen.
  Dim oWriteDoc   ' Neu erstelltes Textdokument zur Aufnahme der Namen.
  Dim oText       ' Das Text-Objekt des Dokuments.
  Dim s : s = "private:factory/swriter"

  ' Ermittelt, was dieses Dokument erzeugen kann.
  ' Fehlerbehandlung für den Fall, dass oDoc die Methode getAvailableServiceNames
  ' nicht unterstützt.
  On Error Goto WrongDoc
  If IsMissing(oDoc) Then
    allNames = ThisComponent.getAvailableServiceNames()
  Else
    allNames = oDoc.getAvailableServiceNames()
  End If

  ' Erstellt ein neues Textdokument zur Aufnahme der Namensliste.
```

```
    oWriteDoc = StarDesktop.loadComponentFromURL(s, "_blank", 0, Array())
    oText = oWriteDoc.getText()
    If IsMissing(nameFilter) Then
      oText.insertString (oText.End, Join(allNames, Chr$(13)), False)
    Else
      Dim i%
      For i = LBound(allNames) To UBound(allNames)
        If (InStr(allNames(i), nameFilter) > 0) Then
          ' Fügt den Text ein.
          oText.insertString (oText.End, allNames(i), False)

          ' Fügt einen neuen Absatz ein.
          oText.insertControlCharacter(oText.getEnd(),_
             com.sun.star.text.ControlCharacter.APPEND_PARAGRAPH, False)
        End If
      Next
    End If
    Exit Sub

    WrongDoc:
      On Error Goto 0
      MsgBox "Dieses Dokument kann keine Services erzeugen.", 48, "Servicetypen"
End Sub
```

10.2.3. Kontext

Im Listing 206 wird der DefaultContext verwendet. OOo enthält „Dinge“, die über ihren Namen erreichbar sind, zum Beispiel den Typbeschreibungsmanager und den Servicemanager. Ein Kontext ist eine Sammlung von Name-Wert-Paaren, mit denen man diese „Dinge“ ansprechen kann. OOo führt einen Standardkontext, erreichbar mit der Basic-Funktion GetDefaultContext. Obwohl beim Erzeugen eines Service eigentlich ein Kontext erforderlich ist, nutzt Basic automatisch den Standardkontext beim Aufruf von CreateUnoService. Ein weiterer Grund dafür, dass Basic einfacher zu benutzen ist als andere Sprachen.

Wenn Sie den Service DefaultContext benötigen, rufen Sie ihn entweder über den ServiceManager auf (Basic-Funktion GetProcessServiceManager()) oder direkt mit der Basic-Funktion GetDefaultContext().

```
    oContext = GetProcessServiceManager().DefaultContext  'Entweder so
    oContext = GetDefaultContext()                        'oder so
```

Listing 210. *Auflistung der Namen der Kontext-Elemente.*

```
Sub DisplayContextNames
  MsgBox Join(GetDefaultContext().getElementNames(), Chr$(10))
End Sub
```

Das Listing 242 verwendet den Standardkontext.

10.2.4. Singletons

Ein Singleton ist ein UNO-Objekt, von dem im Leben des Kontexts einer UNO-Komponente nur genau *eine* Instanz existieren kann. Der Nutzer braucht das nicht weiter zu beachten, denn bei dem ersten Aufruf wird die Instanz angelegt, und alle weiteren Aufrufe erhalten automatisch genau diese eine Instanz.

Ein Singleton des älteren Typs ist ein Service, der mit CreateUnoService(Name) erzeugt wird.

Im Gegensatz dazu referenziert ein Singleton des neueren Typs ein üblicherweise mit einem führenden X thematisch gleich benanntes Interface.
(Quelle: https://wiki.openoffice.org/wiki/Documentation/DevGuide/WritingUNO/Singleton)

Ein solches Singleton wird mit seinem Namen aus dem Standardkontext der Anwendung aufgerufen. Für die Referenz auf die Pfadeinstellungen sieht das also folgendermaßen aus:

```
oContext = GetProcessServiceManager().DefaultContext
oPathSettings = _
            oContext.getValueByName("/singletons/com.sun.star.util.thePathSettings")
```

Beachten Sie, dass zum Namen der einleitende Text „/singletons/“ gehört.

10.3. Komplexere Strukturen (A. Heier)

Über Basic haben Sie Zugriff auf die API und damit auch auf die dort definierten komplexeren Datentypen. Ein einfacher Struct, wie zum Beispiel com.sun.star.beans.PropertyValue, wurde im Abschnitt 10.1. Grundlegende Typen vorgestellt. Dieser Abschnitt informiert Sie über polymorphe Structs und über die assoziativen Datenstrukturen com.sun.star.container.EnumerableMap und com.sun.star.beans.PropertyBag. Diese Services verbinden (assoziieren) einen Schlüssel mit einem Wert. Im Gegensatz zu einer klassischen arraybasierten Lösung ist ein assoziativer Speicher ressourcenschonend und auch bei großen Datenmengen immer noch relativ schnell in der Bearbeitung.

10.3.1. Pair

Außer den einfachen Structs gibt es die sogenannten polymorphen Structs, z. B. com.sun.star.beans.Pair<T,U>. Dabei sind die Datentypen frei wählbar, es müssen aber UNO-Datentypen sein (zur Verwendung einfacher UNO-Datentypen s. Abschnitt 10.1.1. . Einfache UNO-Datentypen). Sie können hier theoretisch alle UNO-Werte verwenden. Im praktischen Einsatz aber kommen Ihre Werte aus Basic. Wählen Sie also die UNO-Datentypen, die zu den genutzten Basic-Werten passen.

Ein Pair enthält zwei Elemente, auf die man mit „.First“ beziehungsweise „.Second“ zugreift. T und U sind Platzhalter für die Datentypen, die Sie bei der Variablendeklarierung in Winkelklammern konkret angeben müssen, s. Listing 211. Statt mit CreateObject können Sie die Variable auch mit der Anweisung „Dim As New“ deklarieren. In diesem Fall müssen Sie den Ausdruck in Anführungszeichen setzen:

```
Dim oPair As New "com.sun.star.beans.Pair<long,boolean>"
```

Achtung Leider wird in der API-Dokumentation von LO der Inhalt der Winkelklammern mit Leerzeichen dargestellt („< T, U >“). Das kann zu der Annahme führen, dass in Basic ebenso Leerzeichen zu beachten sind. Leerzeichen dürfen in der Variablendeklarierung nicht enthalten sein! In der API-Dokumentation von AOO (4.1.2) wird Pair gar nicht beschrieben. Es funktioniert aber auch dort!

Listing 211. *Polymorphes PropertyValue.*

```
Sub PolymorphicStruct
  Dim oPair As Object
  oPair = CreateObject("com.sun.star.beans.Pair<long,boolean>")
  oPair.First = 4711      'Setzt die erste Objekt-Eigenschaft; hier long
  oPair.Second = True     'Setzt die zweite Objekt-Eigenschaft; hier boolean
  MsgBox "erstes Objekt: " & oPair.First & ", zweites Objekt: " & oPair.Second
End Sub
```

10.3.2. EnumerableMap

Eine EnumerableMap enthält Elemente in strukturierter Form. Über einen eindeutigen Schlüssel wird ein Element in einer Map identifiziert. Ebenso kann man über die Map iterieren. Ein Beispiel erleichtert Ihnen den Einstieg:

Listing 212. *EnumerableMap erzeugen, befüllen und auslesen*

```
Sub ExampleEnumerableMap
  Dim oMap
  oMap = com.sun.star.container.EnumerableMap.create("string", "short")
  oMap.put("ExampleKey", CreateUnoValue("short", 4711))
  MsgBox oMap.get("ExampleKey")
End Sub
```

Der Service EnumerableMap kann nicht einfach wie andere Services mit der Basic-Funktion CreateUnoService (s. Abschnitt 10.2.2. . UNO-Services) erzeugt werden. Sie müssen einen der beiden verfügbaren Konstruktoren (create bzw. createImmutable) verwenden, wodurch Sie ein Map-Objekt erzeugen (s. Listing 212). Diese Map unterstützt das Interface XEnumerableMap, das seinerseits das Interface XMap erbt.

Tabelle 87. *Die Konstruktoren des Service com.sun.star.container.EnumerableMap.*

Methode	Beschreibung
create(KeyType, ValueType)	Erzeugt eine EnumerableMap mit den übergebenen Datentypen.
createImmutable(KeyType, ValueType, Values)	Erzeugt eine EnumerableMap mit den übergebenen Datentypen und einer Werteliste (Values) als Array von Pair-Structs. Diese Map ist danach nicht mehr änderbar.

Für eine detaillierte Übersicht über die möglichen Schlüssel- und Werttypen lesen Sie in der API-Dokumentation des Service com.sun.star.container.EnumerableMap nach. Auch hier bietet es sich an, die Typen zu wählen, die zu den verwendeten Basic-Werten passen.

Tabelle 88. *Die wichtigsten Methoden des Service com.sun.star.container.EnumerableMap.*

Methode	Beschreibung
put(Key, Value)	Speichert ein Schlüssel-Wert-Paar in der Map.
get(Key)	Gibt den Wert zurück, der unter dem übergebenen Schlüssel gespeichert ist.
remove(Key)	Löscht aus der Map das Schlüssel-Wert-Paar, das durch den übergebenen Schlüssel identifiziert wird.
containsKey(Key)	Prüft, ob der übergebene Schlüssel in der Map enthalten ist.
containsValue(Value)	Prüft, ob der übergebene Wert in der Map enthalten ist.
clear()	Entfernt alle Schlüssel-Wert-Paare aus der Map.
createElementEnumeration(Boolean)	Erzeugt eine Enumeration über alle Elemente. Der Parameter bestimmt, ob die neue Enumeration als Kopie isoliert von der Map sein soll.
createKeyEnumeration(Boolean)	Erzeugt eine Enumeration über alle Schlüssel. Der Parameter bestimmt, ob die neue Enumeration als Kopie isoliert von der Map sein soll.
createValueEnumeration(Boolean)	Erzeugt eine Enumeration über alle Werte. Der Parameter bestimmt, ob die neue Enumeration als Kopie isoliert von der Map sein soll.
hasElements()	Gibt True zurück, wenn noch Elemente in der Map enthalten sind, sonst False.

Iterieren können Sie nach folgender Technik:

```
'Über Keys oder Values iterieren
''XXX' durch 'Value' oder 'Key' ersetzen
oEnum = oMap.createXXXEnumeration(False)
Do While oEnum.hasMoreElements()
  oXXX = oEnum.nextElement()
  '...
Loop
```

```
'Über Element iterieren
oEnum = oMap.createElementEnumeration(False)
Do While oEnum.hasMoreElements()
  oPair = oEnum.nextElement()
  oKey = oPair.First
  oValue = oPair.Second
  '...
Loop
```

Das Listing 213 zeigt ein Beispiel für eine mit create erstellte EnumerableMap, in dem Sie sehen können, wie man die Anzahl der Elemente ermittelt und wie man ein sortiertes Array der verwendeten Schlüssel erhält. Die Reihenfolge der Schlüssel in einer EnumerableMap ist unbestimmt. Die Map selbst bietet weder Methoden zur Sortierung noch zur Zählung der Elemente. Man muss diese Funktionen über die Enumeration herstellen.

Listing 213. *Abiturienten mit ihren Noten in einer EnumerableMap*

```
REM Das folgende Makro erstellt eine EnumerableMap mit den Namen von Abiturienten
REM als Schlüssel und deren Abiturnoten als Wert und gibt eine nach Schlüssel
REM sortierte Liste der Elemente aus.
Sub ExampleMap
REM Autor: V. Lenhardt
  Dim oMap                    'Das EnumerableMap-Objekt
  Dim sKeysSorted() As String 'Array der sortierten Schlüsselstrings
  Dim n As Long               'Arrayindex
  Dim sMsg As String          'Ausgabestring

  REM Als Schlüsseltyp habe ich "string" gewählt. Als Werttyp ist "any" eine
  REM sichere und problemlose Wahl, vergleichbar mit dem Basic-Typ "Variant".
  REM Bei einem spezifischen Datentyp kann bei der Schlüssel/Wert-Speicherung
  REM unter Umständen eine gezielte Konvertierung mit CreateUnoValue nötig sein.
  oMap = com.sun.star.container.EnumerableMap.create("string", "any")
  oMap.put("Zoe", 1.2)
  oMap.put("Julia", 1.1)
  oMap.put("Kevin", 2.4)
  oMap.put("Anna", 1.8)
  oMap.put("Paul", 3.1)
  oMap.put("Ole", 3.9)

  sKeysSorted() = SortedKeys(oMap)
  For n = 0 To UBound(sKeysSorted())
    sMsg = sMsg & sKeysSorted(n) & ": " & oMap.get(sKeysSorted(n)) & Chr(10)
  Next
  MsgBox sMsg, 0, "Sortierte Map"
End Sub

REM Gibt die Anzahl der Elemente einer EnumerableMap zurück.
Function MapCount(oMap) As Long
  Dim oEnum
  Dim oDummy
  Dim n As Long

  oEnum = oMap.createElementEnumeration(False)
  Do While oEnum.hasMoreElements()
    oDummy = oEnum.nextElement()
    n = n + 1
  Loop
  MapCount = n
End Function
```

```
REM Gibt die Schlüssel einer EnumerableMap als sortiertes Array zurück.
REM Zur Sortierung wird die Funktion BubbleSortList in der
REM Basic-Bibliothek "Tools" verwendet.
Function SortedKeys(oMap)
  Dim oEnum
  Dim sKeys() As String
  Dim n As Long

  REM Die Rückgabeliste wird auf die Anzahl der enthaltenen Elemente dimensioniert.
  Redim sKeys(MapCount(oMap) - 1) As String

  oEnum = oMap.createKeyEnumeration(False)
  Do While oEnum.hasMoreElements()
    sKeys(n) = oEnum.nextElement()
    n = n + 1
  Loop

  GlobalScope.BasicLibraries.loadLibrary("Tools")
  SortedKeys() = BubbleSortList(sKeys())
End Function
```

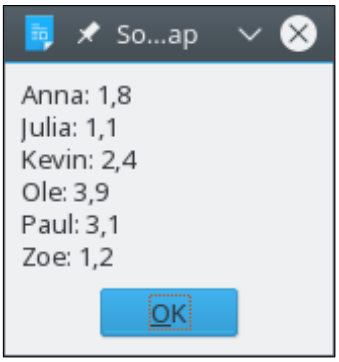


Bild 74. *Nach Schlüsseln sortierte EnumerableMap.*

Möchten Sie hingegen auf eine feste Anzahl an Elementen in der Map zurückgreifen, die später nicht mehr veränderbar sein sollen, dann instanziieren Sie eine EnumerableMap mit createImmutable.

Listing 214. *Beispiel für eine ImmutableMap*

```
Sub ImmutableMapTest()
  Dim oMap
  REM Die Variablendefinition eines polymorphen Structs muss in Anführungszeichen
  REM erfolgen. Das Pair-Array für den EnumerableMap-Konstruktor createImmutable
  REM muss die Datentypen <any,any> enthalten (ohne Leerzeichen), unabhängig von den
  REM konkret verwendeten Schlüssel- oder Werttypen.
  Dim oPairs(4) As New "com.sun.star.beans.Pair<any,any>"

  oPairs(0).First = "LOG_LEVEL_OFF"
  oPairs(0).Second = CreateUnoValue("short", 0)
  oPairs(1).First = "LOG_LEVEL_TRACE"
  oPairs(1).Second = CreateUnoValue("short", 1)
  oPairs(2).First = "LOG_LEVEL_WARN"
  oPairs(2).Second = CreateUnoValue("short", 2)
  oPairs(3).First = "LOG_LEVEL_ERROR"
  oPairs(3).Second = CreateUnoValue("short", 4)
  oPairs(4).First = "LOG_LEVEL_ALL"
  oPairs(4).Second = CreateUnoValue("short", 8)
```

```
  oMap = _
    com.sun.star.container.EnumerableMap.createImmutable("string", "short", oPairs())

  MsgBox oMap.get("LOG_LEVEL_WARN")       '2
  'MsgBox oMap.put("LOG_LEVEL_WARN", 16)  'Fehler
  'MsgBox oMap.put("LOG_LEVEL_NEW", 16)   'Fehler
End Sub
```

Achtung

In der LO-API-Dokumentation zum Service EnumerableMap ist zur Methode „createImmutable“ als Sequenztyp das polymorphe Struct „`com.sun.star.beans.Pair< any, any >`“ angegeben, mit Leerzeichen innerhalb der Winkelklammern. Das kann zu der Annahme führen, dass in Basic ebenso Leerzeichen zu beachten sind. In der Variablendeklarierung dürfen keine Leerzeichen enthalten sein!

In der entsprechenden AOO-API-Dokumentation ist der Service als „not published“ gekennzeichnet. Als Sequenztyp steht dort „`com.sun.star.beans.Pair< string, string >`“, auch mit Leerzeichen. Das ist doppelt irreführend. Denn es funktioniert nur mit dem Datentyp any: „`com.sun.star.beans.Pair<any,any>`“ (ohne Leerzeichen)!

10.3.3. PropertyBag

Ein PropertyBag ist nichts anderes als ein Container, der Eigenschaften mit Namen und Werten als Struct com.sun.star.beans.PropertyValue speichert. Ein PropertyBag ist kein assoziativer Speicher. Die assoziierten Schlüssel-Wert-Paare werden nicht von ihm verwaltet, wie etwa bei einer EnumerableMap. Es ist eher ein Container, in dem assoziative Datenstrukturen enthalten sind. Zur Identifizierung einer Eigenschaft ist immer ein Schlüssel vom Typ String erforderlich. Ein PropertyValue nimmt immer nur ein einziges Schlüssel-Wert-Paar auf. Ein PropertyBag enthält einen oder mehrere PropertyValues und stellt somit nur so etwas wie einen Behälter dar.

Listing 215. *PropertyBag erzeugen, befüllen und auslesen*

```
Sub ExamplePropertyBag
  Dim oBag
  oBag = CreateUnoService("com.sun.star.beans.PropertyBag")
  'erzielt die gleiche Wirkung wie obere Codezeile:
  'oBag = com.sun.star.beans.PropertyBag.createDefault()
  oBag.addProperty("Test", com.sun.star.beans.PropertyAttribute.BOUND, 666) 'anlegen
  oBag.setPropertyValue("Test", 667)   'schreiben
  MsgBox oBag.getPropertyValue("Test") 'lesen
End Sub
```

Beim Hinzufügen von Eigenschaften ist als zweiter Parameter ein PropertyAttribute-Wert notwendig (s. Tabelle 89), der darüber entscheidet, wie mit dem Wert der Eigenschaft verfahren werden kann.

Tabelle 89. *Einige Konstanten der Gruppe com.sun.star.beans.PropertyAttribute.*

Konstante	Beschreibung
MAYBEVOID = 1	Der Wert der Eigenschaft kann leer sein.
BOUND = 2	Bewirkt, dass bei der Änderung des Werts der Eigenschaft ein PropertyChangeEvent ausgelöst wird und alle registrierten Beobachter davon erfahren.
READONLY = 16	Bewirkt einen Schreibschutz der Eigenschaft.
REMOVABLE = 128	Die hinzugefügte Eigenschaft kann auch wieder gelöscht werden.

Der Name einer Eigenschaft ist immer ein String, der Wert jedoch kann beliebig sein, denn als Datenpyp ist „any“ festgelegt. Die folgende Tabelle 90 gibt Ihnen einen Überblick über die wichtigsten Methoden des Service.

Tabelle 90. *Die wichtigsten Methoden des Service com.sun.star.beans.PropertyBag.*

Methode	Beschreibung
setPropertyValue(String, Variant)	Bei bereits hinzugefügten Eigenschaften schreibt diese Methode einen neuen Wert in die durch den Namen spezifizierte Eigenschaft.
getPropertyValue(String)	Gibt den Wert der angefragten Eigenschaft zurück.
addPropertyChangeListener(String, Listener)	Fügt einen Beobachter für entsprechend maskierte Eigenschaften hinzu. Ein leerer Namensstring ("") registriert den Beobachter für alle BOUND-Properties. Bei Angabe eines Namens wird ausschließlich für diese Eigenschaft ein Beobachter hinzugefügt.
removePropertyChangeListener (String, Listener)	Entfernt den Beobachter.
addProperty(String, Short, Variant)	Fügt dem PropertyBag eine Eigenschaft neu hinzu. Der zweite Parameter ist ein Wert aus der Konstantengruppe com.sun.star.beans.PropertyAttribute (s. Tabelle 89).
removeProperty(String)	Entfernt eine hinzugefügte Eigenschaft aus dem Container.
getPropertyValues()	Gibt ein com.sun.star.beans.PropertyValue-Array aller im Container enthaltenen Eigenschaften zurück.
setPropertyValues(Array als com.sun.star.beans.PropertyValue)	Setzt die Werte aller als Array übergebenen Eigenschaften. Die Eigenschaften müssen vorher im Container angelegt gewesen sein.

Eine Besonderheit von PropertyBags ist die Möglichkeit, einer Eigenschaft einen Beobachter hinzuzufügen. Damit können Sie nun auf Änderungen an der „Variablen“ reagieren.

```
Global oPCL
Sub Foo()
  '...
  oPCL = CreateUnoListener("PCL_", "com.sun.star.beans.XPropertyChangeListener")
  '...
  oBag.addPropertyChangeListener("Test", oPCL)
  '...
End Sub
```

Den Listener können Sie erst nach dem Anlegen der Eigenschaften dafür registrieren. Abschließend sind dann noch die beiden Methoden für die Ereignisbehandlung zu implementieren:

```
Sub PCL_disposing()
  Print "Innerhalb PCL_disposing()"
End Sub

Sub PCL_propertyChange()
  Print " Innerhalb PCL_propertyChange()"
End Sub
```

Hinterher muss der Beobachter natürlich wieder entfernt werden.

10.4. Inspizierung von Universal Network Objects

Beim Schreiben eines Basic-Makros verstehe ich nicht immer die Werte, die von internen OOo-Funktionen zurückgegeben werden – zum Beispiel bei der Untersuchung des von GetDefaultContext zurückgegebenen Werts. Ich schreibe dann Testcode zur Untersuchung der Rückgabewerte, um geeignete Entscheidungen zu treffen. Oft füge ich noch weiteren Testcode für einen tieferen Einblick in das Objekt hinzu. Die Inspizierung startet mit Routinen, die Sie wahrscheinlich schon kennen, s. Tabelle 91.

Tabelle 91. *Grundlegende Inspizierungsroutinen.*

Routine	Kommentar
IsMissing(obj)	Zur Kontrolle, ob optional Argumente weggelassen wurden.
IsNull(obj)	Ein Null-Objekt kann nicht inspiziert werden, aber man weiß, dass es Null ist.
IsEmpty(obj)	Ein leeres Objekt kann nicht inspiziert werden, aber man weiß, dass es leer ist.
IsArray(obj)	Mit Methoden zur Array-Inspizierung kann man mehr über das Array erfahren.
TypeName(obj)	Findet heraus, ob es ein einfacher Typ ist wie String oder Integer. Wenn es der Typ Object ist, dann ist es wahrscheinlich ein UNO-Struct oder ein UNO-Service.
IsUnoStruct(obj)	Findet heraus, ob es ein UNO-Struct ist.

Der Code im Listing 216 demonstriert, wie mit den Funktionen der Tabelle 91 erste Erkenntnisse gewonnen werden.

Listing 216. *Inspizierung des Objekts TextTables im aktuellen Dokument.*

```
Dim v
v = ThisComponent.getTextTables()
Print IsObject(v)       'True
Print IsNull(v)         'False
Print IsEmpty(v)        'False
Print IsArray(v)        'False
Print IsUnoStruct(v)    'False
Print TypeName(v)       'Object
MsgBox v.dbg_methods    'Zu dieser Eigenschaft kommen wir später.
```

Wenn das zurückgegebene Objekt den Typnamen Object hat und kein UNO-Struct ist, dann ist es wahrscheinlich ein UNO-Service. Mit der Funktion HasUnoInterfaces stellen Sie fest, ob ein UNO-Objekt einen Satz von Interfaces unterstützt. Das erste Argument ist das zu prüfende Objekt. Die weiteren Argumente sind Interfacenamen. Wenn alle der gelisteten Interfaces unterstützt werden, gibt die Funktion True zurück, ansonsten False. So können mehrere Interfaces gleichzeitig geprüft werden.

```
HasUnoInterfaces(obj, interface1)
HasUnoInterfaces(obj, interface1[, interface2[, interface3[, ...]]])
```

Um zwischen einem UNO-Struct, einem beliebigen Objekt und einem UNO-Service zu unterscheiden, prüfen Sie zuerst, ob die Variable ein Objekt ist, leicht geschehen mit der Funktion TypeName. Wenn TypeName das Wort Object ausgibt, dann wissen Sie schon einmal, dass es sich um irgendein Objekt handelt. Als nächstes testen Sie, ob das Objekt ein UNO-Struct ist, mit Hilfe der Funktion IsUnoStruct. Wenn Sie schließlich herausfinden, dass das Objekt ein Interface unterstützt (überhaupt nur irgendeines), wissen Sie, dass es ein UNO-Service ist. Jedes Interface ist von com.sun.star.uno.XInterface abgeleitet, also reicht es zu prüfen, ob das Interface XInterface unterstützt wird. Der Code im Listing 217 verwendet die Basic-Variable ThisComponent, die das aktuelle Dokument referenziert.

Listing 217. *Mit HasUnoInterfaces und IsUnoStruct ermitteln Sie den UNO-Typ.*

```
Dim aProp As New com.sun.star.beans.PropertyValue
Print IsUnoStruct(aProp)                                          'True
Print HasUnoInterfaces(aProp, "com.sun.star.uno.XInterface")      'False
Print IsUnoStruct(ThisComponent)                                  'False
Print HasUnoInterfaces(ThisComponent, "com.sun.star.uno.XInterface")  'True
```

Tipp Wenn das erste Argument der Funktion HasUnoInterfaces kein Objekt ist, wird ein Laufzeitfehler generiert. Das Objekt darf ruhig Null sein, aber es muss ein Objekt sein. Wenn das Argument vom Typ Variant ist, muss es ein Objekt enthalten, es darf nicht leer sein.

Die meisten UNO-Services unterstützen auch das Interface com.sun.star.lang.XServiceInfo, mit dem Sie das Objekt fragen können, welche Services von ihm unterstützt werden, s. Tabelle 92.

Tabelle 92. *Methoden im Interface com.sun.star.lang.XServiceInfo.*

Methode	Beschreibung
getImplementationName()	Gibt einen String zurück, der den konkreten Service eindeutig identifiziert. Zum Beispiel ist SwXTextDocument der Name eines Writer-Textdokuments.
getSupportedServiceNames()	Gibt ein Stringarray der Services zurück, die vom Objekt unterstützt werden.
supportsService(serviceName)	Gibt True zurück, wenn das Objekt den angegebenen Service unterstützt.

OOo unterstützt verschiedene Dokumenttypen, zum Beispiel Writer (Textverarbeitung) oder Calc (Tabellenkalkulation). Jeder Dokumenttyp unterstützt wenigstens einen Service, der ausschließlich diesem Dokumenttyp zu Eigen ist, s. Tabelle 93. Sie können den Dokumenttyp bestimmen, indem Sie prüfen, ob er einen dieser Services unterstützt. Ein anderer Weg geht über die Methode getImplementationName(), s. auch Tabelle 92.

Der Code im Listing 218 zeigt, wie man sicherstellt, dass eine Variable ein Textdokument referenziert. Wenn das Argument vDoc nicht das Interface XServiceInfo unterstützt, tritt jedoch ein Laufzeitfehler auf, weil die Methode supportsService nicht eingebunden ist. Falls nötig, setzen Sie eine passende Fehlerbehandlung ein, wie Listing 219 zeigt (wird im Listing 228 aufgerufen). Sollte das Argument fehlen, wird das aktuelle Dokument angenommen. Im Falle eines Fehlers oder falls keiner der bekannten Services unterstützt wird, gibt die Funktion „Unbekannt“ zurück.

Tabelle 93. *Eindeutige Servicenamen für Dokumenttypen.*

Dokumenttyp	Service
Zeichnung	com.sun.star.drawing.DrawingDocument
Textdokument	com.sun.star.text.TextDocument
HTML-Dokument	com.sun.star.text.WebDocument
Tabellendokument	com.sun.star.sheet.SpreadsheetDocument
Formel	com.sun.star.formula.FormulaProperties
Präsentation	com.sun.star.presentation.PresentationDocument
Datenbank	com.sun.star.sdb.OfficeDatabaseDocument
Datenbanktabelle	com.sun.star.sdb.DataSourceBrowser

Listing 218. *Absichern, dass das Dokument ein Textdokument ist.*

```
Sub DoSomethingToWriteDocument(vDoc)
  If Not vDoc.supportsService("com.sun.star.text.TextDocument") Then
    MsgBox "Es wird ein Textdokument erwartet", 48, "Fehler"
    Exit Sub
  End If
  REM Hier beginnt der Rest der Subroutine
End Sub
```

Listing 219. *Ermittlung des Dokumenttyps.*

```
Function GetDocType(Optional vDoc) As String
  On Error GoTo Oops
  If IsMissing(vDoc) Then vDoc = ThisComponent
  If vDoc.supportsService("com.sun.star.sheet.SpreadsheetDocument") Then
    GetDocType = "Tabellendokument"
  ElseIf vDoc.supportsService("com.sun.star.text.TextDocument") Then
    GetDocType = "Textdokument"
```

```
    ElseIf vDoc.supportsService("com.sun.star.drawing.DrawingDocument") Then
      GetDocType = "Zeichnung"
    ElseIf vDoc.supportsService(_
                        "com.sun.star.presentation.PresentationDocuments") Then
      GetDocType = "Präsentation"
    ElseIf vDoc.supportsService("com.sun.star.formula.FormulaProperties") Then
      GetDocType = "Formel"
    ElseIf vDoc.supportsService("com.sun.star.sdb.OfficeDatabaseDocument") Then
      GetDocType = "Datenbank"
    Else
      GetDocType = "Unbekannt"
    End If
  Oops:
    If Err <> 0 Then GetDocType = "Unbekannt"
    On Error GoTo 0   'Abschaltung der Fehlerbehandlung NACH dem Fehlercheck
  End Function
```

Es gibt Dokumenttypen, die nicht so leicht erkannt werden können. Zum Beispiel unterstützen XForms-Dokumente und Masterdokumente dieselben Services wie reguläre Textdokumente. Alle Dokumente, die auf Writer basieren, unterstützen den Service com.sun.star.text.GenericTextDocument, und alle Dokumente, die einen Office-Typ haben, unterstützen den Service com.sun.star.document.OfficeDocument.

Wenn ich eine Frage zu einem Interface oder Service habe, schaue ich zuallererst online in der englischsprachigen offiziellen Dokumentation nach. AOO und LO führen unterschiedliche Dokumentationen als UNO-API-Referenz. Dort finden Sie auch die Abkürzung IDL – als IDL Reference oder UNO IDL API. Das ist die Abkürzung von Interface Definition Language (Schnittstellenbeschreibungssprache oder Schnittstellendefinitionssprache) und ist laut Wikipedia „eine deklarative formale Sprache und beinhaltet eine Sprachsyntax zur Beschreibung von Schnittstellen einer Software-Komponente“. Durch die Verwendung der IDL sind alle Seiten der Dokumentation auf dieselbe Art und Weise strukturiert.

- AOO: http://www.openoffice.org/api/docs/common/ref/com/sun/star/module-ix.html
- LO: https://api.libreoffice.org/docs/idl/ref/index.html.

Bei AOO können Sie über einen globalen alphabetischen Index direkt zu jedem verwendeten API-Namen springen. Dort ist es leider aber nicht so einfach, einen Gesamtüberblick über das Element zu erlangen. Schauen Sie sich als Beispiel die Dokumentation über den Service TextCursor an: http://api.openoffice.org/docs/common/ref/com/sun/star/text/TextCursor.html.

Folgendes ist zu bemängeln:

- Die Dokumentation erscheint nicht in einem Guss, man muss viele Links verfolgen, um herauszufinden, was das Objekt kann. Es werden zwar alle unterstützten und optionalen Interfaces aufgezählt, aber man muss jedes Interface einzeln inspizieren, um dessen Potenzial kennenzulernen.
- Jedes Interface hat seine eigene Seite. Wenn ein Interface von einem anderen abstammt, muss man auf einer anderen Seite die Definitionen des elterlichen Interface nachschlagen.
- Ohne das Objekt zu inspizieren, weiß man nicht, ob die optionalen Elemente für ein spezifisches Objekt unterstützt werden.

LO führt zwar keine eigene Indexseite, bietet aber auf jeder Seite eine Suchbox, die interaktiv alle Namen auflistet, die mit den eingegebenen Buchstaben beginnen. Das ist deutlich praktischer als eine gesonderte Seite. Zum Vergleich die Dokumentation über den Service TextCursor: https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1text_1_1TextCursor.html

Hier gibt es einige Verbesserungen:

- In einem Vererbungsdiagramm (Inheritance diagram) werden alle Verbindungen zu anderen Services und Interfaces aufgeführt.
- Die Elemente (englisch Members) der ererbten Services und Interfaces, sowohl Attribute (Eigenschaften) als auch Methoden, sind über Aufklapplisten einsehbar.

Die meisten UNO-Services enthalten die Eigenschaften dbg_properties, dbg_methods und dbg_supportedInterfaces. Jede dieser Eigenschaften ist ein String, der die Liste der unterstützten Eigenschaften, Methoden und Interfaces enthält. Jeder String beginnt mit einer Wendung wie „Properties of Object "ThisComponent":“. Die einzelnen Elemente werden mit einem der Trenner der Tabelle 94 hintereinander angeschlossen. Manchmal folgen den Trennern zusätzliche Leerzeichen, manchmal auch weitere Zeilenumbruchzeichen – Chr$(10).

Tabelle 94. *UNO-„dbg_“-Eigenschaften.*

Eigenschaft	Trenner	Beschreibung
dbg_properties	“;”	Alle vom Objekt unterstützten Eigenschaften.
dbg_methods	“;”	Alle vom Objekt unterstützten Methoden.
dbg_supportedInterfaces	Chr$(10)	Alle vom Objekt unterstützten Interfaces.

Der Code im Listing 220 bietet eine einfache Methode zu sehen, was ein Objekt alles kann. Manchmal aber ist die Liste der Elemente zu lang, und Teile des Anzeigedialogs passen nicht auf den Bildschirm. Um dieses Problem zu vermeiden, splittet der Code im Listing 221 die Liste in kleinere, leicht zu beherrschende Brocken. Bild 75 zeigt nur einen der vielen Dialoge, die das Makro im Listing 221 ausgibt. Bedenken Sie, dass die Dialoge den Rahmen des Bildschirms sprengen können, also drücken Sie die Eingabetaste oder klicken Sie zum Beenden eines jeden Dialogs auf OK oder auf das Abbrechensymbol.

Listing 220. *Die „dbg_“-Eigenschaften liefern nützliche Informationen.*

```
Sub ExampleDisplayRawDbgInfoStr()
  Dim vObj
  vObj = ThisComponent
  MsgBox vObj.dbg_properties
  MsgBox vObj.dbg_methods
  MsgBox vObj.dbg_supportedInterfaces
End Sub
```

Listing 221. *Ausgabe der Informationen aus den Debug-Eigenschaften.*

```
Sub ExampleDisplayDbgInfoStr()
  Dim vObj
  vObj = ThisComponent
  DisplayDbgInfoStr(vObj.dbg_properties, ";", 40, "Properties")
  DisplayDbgInfoStr(vObj.dbg_methods, ";", 40, "Methods")
  DisplayDbgInfoStr(vObj.dbg_supportedInterfaces, Chr$(10), 40, "Interfaces")
End Sub

Sub DisplayDbgInfoStr(sInfo$, sSep$, nChunks, sHeading$)
  REM sInfo = Debug-Eigenschaft
  REM sSep = Trenner
  REM nChunks = Zeilen (Elemente) pro MsgBox
  REM sHeading = MsgBox-Titel

  Dim aInfo()                    'Array für die einzelnen Strings
  Dim i As Integer               'Indexvariable
  Dim j As Integer               'Integer-Variable für temporäre Werte
  Dim s As String                'Enthält den noch nicht erledigten Anteil
```

```
    s = sInfo$
    j = InStr(s, ":")               'Erster Doppelpunkt
    If j > 0 Then Mid(s, 1, j, "")  'Entfernt den Teil bis zum ersten Doppelpunkt
    Do
      aInfo() = Split(s, sSep$, nChunks)        'Splittet den String an den Trennern.
      s = aInfo(UBound(aInfo()))                'Der Rest des Strings, falls die Anzahl
      If InStr(s, sSep$) < 1 Then               'der Elemente größer als nChunks war.
        s = ""                                  'Wenn nicht, wird s geleert.
      Else                                      'Wenn es im Rest einen Trenner gibt,
        ReDim Preserve aInfo(nChunks - 2)       'wird das Array neu dimensioniert,
      End If                                    'damit das letzte Element entfernt wird.
      For i = LBound(aInfo()) To Ubound(aInfo())'In jedem Element werden Leerzeichen
        aInfo(i) = Trim(aInfo(i))               'am Anfang und am Ende entfernt.
        j = InStr(aInfo(i), Chr$(10))           'Manche haben ein Zeilenendezeichen,
        If j > 0  Then Mid(aInfo(i), j, 1, "")  'das entfernt werden sollte.
      Next
      MsgBox Join(aInfo(), Chr$(10)), 0, sHeading$
    Loop Until Len(s) = 0
End Sub
```

Wenn in einer der „dbg_“-Eigenschaften ein Typ angegeben ist, wird er mit „Sbx“ eingeleitet, s. Bild 75. Die mit Sbx beginnenden Namen sind die von Basic verwendeten internen Namen.

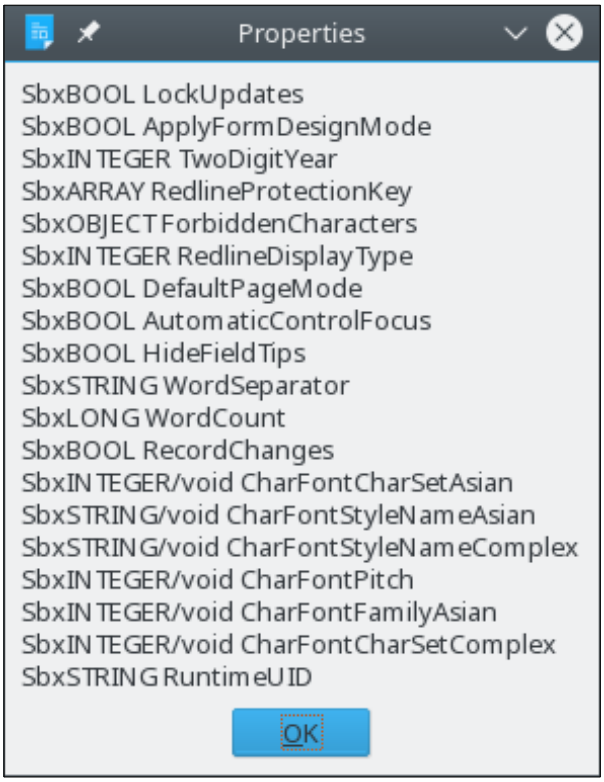


Bild 75. *Einige Eigenschaften eines Textdokuments, nur ein Dialog von vielen.*

Tipp	Im Abschnitt 18.8. Das Beispiel Objektinspektor wird ein Objektbrowser präsentiert, der alle diese Elemente zeigt.

Tipp	Ein populärer Objektbrowser ist Xray, dessen Gebrauch an der folgenden Adresse beschrieben wird: http://wiki.services.openoffice.org/wiki/Extensions_development_basic

10.5. Das Modul Reflection

Das Modul Reflection (com.sun.star.reflection) enthält Services und Interfaces, über die zur Laufzeit Informationen über Objekte und Interfaces gewonnen werden können. Eine Beschreibung aller Möglichkeiten würde den Rahmen dieses Buches sprengen. Darin einzudringen möge dem interessierten Leser vorbehalten bleiben. Zwei der Services sollen aber nicht unerwähnt bleiben: CoreReflection und TypeDescriptionManager.

10.5.1. Der Service CoreReflection (V. Lenhardt)

Über den Service com.sun.star.reflection.CoreReflection erhalten Sie die Möglichkeit, Informationen über UNO-Typen zu gewinnen. Dieser Service ist mittlerweile veraltet, stattdessen soll das Singleton theCoreReflection verwendet werden.

Der Service stellt über das Interface com.sun.star.reflection.XIdlReflection zwei Methoden bereit, einen bestimmten Typ zu untersuchen: mit forName(Typname) wird der Typ direkt benannt, mit getType(Objekt) wird der Typ des angegebenen Objekts verwendet.

In beiden Fällen wird das Interface com.sun.star.reflection.XIdlClass zurückgegeben, das eine Reihe von Methoden für Informationen über den UNO-Typ zur Verfügung stellt.

Als Beispiel soll der Umstand dienen, dass in LO ab der Version 4.1 für ein Datumsfeld ein anderer UNO-Typ verwendet wird als in AOO: in AOO ist es der Typ Long und in LO ein Struct (siehe Abschnitt Datumsfeld).

Das Listing 222 testet, ob das Datumsfeld ein Struct oder eine Long-Ganzzahl ist, so dass Sie einen Code schreiben können, der sowohl für LO als auch für AOO verwendbar ist. Der hier verwendete DefaultContext wird im Abschnitt 10.2.3. Kontext erläutert, Singletons im Abschnitt 10.2.4. Singletons.

Listing 222. *Ermittelt den Typ eines Datumsfelds über den Service CoreReflection.*

```
Sub GetDateFieldType
REM Ermittelt den Typ eines Datumsfelds.
  Dim oContext
  Dim oCoreReflection
  Dim oGetDateReturnType  'Der Rückgabetyp der "getDate"-Methode
  oContext = GetProcessServiceManager().DefaultContext
  oCoreReflection = oContext.getValueByName _
                    ("/singletons/com.sun.star.reflection.theCoreReflection")
  oGetDateReturnType = oCoreReflection.forName("com.sun.star.awt.XdateField")._
                       getMethod("getDate").getReturnType()
  If oGetDateReturnType.getTypeClass() = com.sun.star.uno.TypeClass.LONG Then
    MsgBox "Long"
  ElseIf oGetDateReturnType.getTypeClass() = com.sun.star.uno.TypeClass.STRUCT _
         And oGetDateReturnType.getName() = "com.sun.star.util.Date" Then
    MsgBox "Struct"
  Else
    MsgBox "Unbekannt"
  End If
End Sub
```

10.5.2. Die Verwendung des Typbeschreibungsmanagers

Der TypeDescriptionManager regelt die Typbeschreibungen und wirkt als zentraler Zugang für jede Typbeschreibung. Man erreicht dieses Objekt als Singleton aus dem Standardkontext. Ein direkt erzeugter TypeDescriptionManager ist kein nutzbares Objekt:

```
CreateUnoService("com.sun.star.reflection.TypeDescriptionManager")
```

Ich habe ein nutzbares Objekt mit der gemeinhin vorgeschlagenen Methode (in Kurzform) erzeugt, doch auch die direkte Verwendung des Typs funktionierte nicht:

```
CreateUnoService("com.sun.star.comp.stoc.TypeDescriptionManager")
```

Holen Sie sich den Typbeschreibungsmanager aus dem Standardkontext (s. Abschnitt 10.2.3. Kontext). Dabei können Sie auch sehen, wie man alle verfügbaren Singleton-Objekte auflistet (s. Abschnitt 10.2.4. Singletons).

```
Function GetTypeDescriptionManager()
  Dim sTDMName$  ' Name des  Typbeschreibungsmanagers.
  sTDMName = "/singletons/com.sun.star.reflection.theTypeDescriptionManager"
  GetTypeDescriptionManager() = GetDefaultContext().getValueByName(sTDMName)
End Function
```

Die folgende Methode listet alle „Dinge“ im Modul com.sun.star.awt auf.

Listing 223. *Typenauflistung in einem Modul.*

```
Sub EnumerateTypesTest
  Dim oTDM       ' Typbeschreibungsmanager.
  Dim oTDE       ' Typbeschreibungsauflistungen.
  Dim oTD        ' Beschreibung eines Typs.
  Dim typeArray ' Typen, für die Beschreibungen zurückgegeben werden.
  Dim s$         ' Ausgabestring.
  Dim i%         ' Zeilenzähler

  REM Alle unterstützten Typen.
  typeArray = Array(com.sun.star.uno.TypeClass.VOID, _
     com.sun.star.uno.TypeClass.CHAR, _
     com.sun.star.uno.TypeClass.BOOLEAN, _
     com.sun.star.uno.TypeClass.BYTE, _
     com.sun.star.uno.TypeClass.SHORT, _
     com.sun.star.uno.TypeClass.UNSIGNED_SHORT, _
     com.sun.star.uno.TypeClass.LONG, _
     com.sun.star.uno.TypeClass.UNSIGNED_LONG, _
     com.sun.star.uno.TypeClass.HYPER, _
     com.sun.star.uno.TypeClass.UNSIGNED_HYPER, _
     com.sun.star.uno.TypeClass.FLOAT, _
     com.sun.star.uno.TypeClass.DOUBLE, _
     com.sun.star.uno.TypeClass.STRING, _
     com.sun.star.uno.TypeClass.TYPE, _
     com.sun.star.uno.TypeClass.ANY, _
     com.sun.star.uno.TypeClass.ENUM, _
     com.sun.star.uno.TypeClass.TYPEDEF, _
     com.sun.star.uno.TypeClass.STRUCT, _
     com.sun.star.uno.TypeClass.UNION, _
     com.sun.star.uno.TypeClass.EXCEPTION, _
     com.sun.star.uno.TypeClass.SEQUENCE, _
     com.sun.star.uno.TypeClass.ARRAY, _
     com.sun.star.uno.TypeClass.INTERFACE, _
     com.sun.star.uno.TypeClass.SERVICE, _
     com.sun.star.uno.TypeClass.MODULE, _
     com.sun.star.uno.TypeClass.INTERFACE_METHOD, _
     com.sun.star.uno.TypeClass.INTERFACE_ATTRIBUTE, _
     com.sun.star.uno.TypeClass.UNKNOWN, _
     com.sun.star.uno.TypeClass.PROPERTY, _
     com.sun.star.uno.TypeClass.CONSTANT, _
     com.sun.star.uno.TypeClass.CONSTANTS, _
```

```
      com.sun.star.uno.TypeClass.SINGLETON)

  oTDM = GetTypeDescriptionManager()

  Dim sBaseName$ : sBaseName = "com.sun.star.awt"

  ' Wenn mehrere Ebenen durchlaufen werden sollen, ändern Sie
  ' com.sun.star.reflection.TypeDescriptionSearchDepth.ONE
  ' zu com.sun.star.reflection.TypeDescriptionSearchDepth.INFINITE.
  oTDE = oTDM.createTypeDescriptionEnumeration(sBaseName, _
            typeArray, _
            com.sun.star.reflection.TypeDescriptionSearchDepth.ONE)

  Do While oTDE.hasMoreElements()
    i = i + 1
    oTD = oTDE.nextTypeDescription()
    s$ = s & oTD.Name & Chr$(10)
    If i = 30 Then
      MsgBox s
      s = ""
      i = 0
    End If
  Loop
  MsgBox s
End Sub
```

Mit dem folgenden Makro (von einem Beispiel von Bernard Marcelly adaptiert) erhalten Sie die Information über einen Typ mit vollständigem Namen:

Listing 224. *Objektbeschreibung über einen vollständigen Namen.*

```
Function GetOOoConst(constString)
  Dim sTDMName$
  Dim oTDM

  sTDMName = "/singletons/com.sun.star.reflection.theTypeDescriptionManager"
  oTDM = GetDefaultContext().getValueByName(sTDMName)

  If oTDM.hasByHierarchicalName(constString) Then
    GetOOoConst = oTDM.getByHierarchicalName(constString)
  Else
    MsgBox "Unbekannter Name: " & constString, 16, "OOo-API-Konstante oder Enumeration"
  End If
End Function
```

Mit dieser Methode erhalten Sie aus einem Textstring die Werte für eine Konstante oder eine Enumeration.

```
Print GetOOoConst("com.sun.star.awt.FontSlant.ITALIC")
```

Damit kann man auch ein Objekt erhalten, das den Typ beschreibt. Dann kann man die Werte und Strings auflisten.

Listing 225. *Listet Enumerationen auf.*

```
Sub EnumerateEnumerations(sName$)
  Dim oTD       'Die Beschreibung eines Typs.
  Dim s$        'Ausgabestring.
  Dim sNames    'Array der Namen
```

```
  Dim lValues   'Array der Werte
  Dim i As Long
  Dim iCount As Integer

  oTD = GetOOoConst(sName)
  If IsNull(oTD) Or IsEmpty(oTD) Then
    Exit Sub
  End If

  If HasUnoInterfaces(oTD, "com.sun.star.reflection.XEnumTypeDescription") Then
    'MsgBox Join(oTD.getEnumNames(), Chr$(10))
    sNames = oTD.getEnumNames()
    lValues = oTD.getEnumValues()
    For i = LBound(sNames) To UBound(sNames)
      iCount = iCount + 1
      If (iCount > 40) Then
        MsgBox(s)
        s = ""
      End If
      s = s & lValues(i) & Chr$(9) & sNames(i) & Chr$(10)
    Next
  ElseIf HasUnoInterfaces(oTD, _
                          "com.sun.star.reflection.XConstantsTypeDescription") Then
    lValues = oTD.getConstants()
    For i = LBound(lValues) To UBound(lValues)
      iCount = iCount + 1
      If (iCount > 40) Then
        MsgBox(s)
        s = ""
      End If
      s = s & lValues(i).getConstantValue() & Chr$(9) & lValues(i).getName() & Chr$(10)
    Next
  Else
    MsgBox "Nicht unterstützter Typ " & sName
    Exit Sub
  End If
  MsgBox s
End Sub
```

So kann man Enumerationen auflisten.

```
Sub EnumerateFontSlantEnum()
  EnumerateEnumerations("com.sun.star.awt.FontSlant")
End Sub
```

So kann man Konstantengruppen auflisten.

```
Sub EnumerateFontWeightConstant()
  EnumerateEnumerations("com.sun.star.awt.FontWeight")
End Sub
```

10.6. Typdefinition Object oder Variant

Es ist fast immer sicher, Object-Variablen für UNO-Services zu verwenden. Es ist offenbar so sicher, dass sogar der Makrorecorder für UNO-Services Object-Variablen nutzt. Leider ist es nicht immer so sicher. Der OOo Developer's Guide stellt dezidiert fest, dass Variant-Variablen anstelle von Object-Variablen verwendet werden sollen. Deklarieren Sie Variablen für UNO-Services immer als Typ Variant, nicht als Typ Object. Der Basic-Typ Object ist auf reine Basic-Objekte zugeschnitten, also Objekte, die mit der Syntax „Dim As New" erzeugt werden können. Variant-Variablen sind für UNO-Services die beste Wahl, weil sie Probleme vermeiden, die aus der für Basic spezifischen Behand-

lung des Typs Object entstehen können. Ich habe Andreas Bregas gefragt (einen der Hauptentwickler der Basic-Infrastruktur), und er hat gesagt, dass in den meisten Fällen beide Typen funktionieren. Der OOo Developer's Guide zieht Variant vor, weil es einige wenige Fälle gibt, in denen die Verwendung des Typs Object zu einem Fehler aufgrund der Semantik des alten Basic-Object-Typs kommt. Das ist jedoch so selten, dass er sich an kein Beispiel erinnern konnte, wo es ein Problem darstellte. Umgekehrt bin ich mit der OOo-Version 1 auf ein Problem gestoßen, das sich von selbst erledigte, als ich den Variablentyp von Variant auf Object wechselte.

In dieser Übersetzung von OOME sind UNO-Objekte durchweg als Variant deklariert, außer wenn sie in Listings anderer Autoren als Object behandelt wurden. In der Praxis werden Sie allerdings beide Formen finden – und beides wird funktionieren.

10.7. Vergleich von UNO-Variablen

Mit der Funktion EqualUnoObjects stellen Sie fest, ob zwei Variablen dasselbe UNO-Objekt referenzieren. UNO-Structs werden als Wert kopiert, UNO-Services aber als Referenz. Das bedeutet, dass EqualUnoObjects immer False für zwei Variablen mit einem Struct zurückgibt, aber für UNO-Services True zurückgeben kann.

```
Dim vObj
vObj = ThisComponent
Print EqualUnoObjects(vObj, ThisComponent)          'True

Dim aProp As New com.sun.star.beans.PropertyValue
vObj = aProp
Print EqualUnoObjects(vObj, aProp)                  'False
```

Nach vielen Jahren Praxis mit OOo habe ich schließlich ein Anwendungsbeispiel für EqualUnoObjects gefunden. Eine Warnung: Dies ist ein fortgeschrittenes Thema und verwendet Konstrukte, die bisher noch nicht vorgestellt wurden. Das Problem stellt sich folgendermaßen dar: „Wenn in einem Tabellendokument eine Schaltfläche gedrückt wird, will ich wissen, zu welcher Zelle die Schaltfläche gehört." Mit einigen weiteren Informationen wird das Problem leichter verständlich:

- Jedes Dokument besitzt mindestens eine Folie (Draw page), in der die grafischen Dinge stecken wie zum Beispiel im Dokument eingebettete Bilder.
- Man kann Kontrollelemente, zum Beispiel Schaltflächen, in ein Dokument einbetten.
- Jedes Kontrollelement besitzt eine in der Folie eingebettete grafische Form.
- Jedes Kontrollelement besitzt auch ein „Data model", das Eigenschaften und andere solche Dinge enthält. Wenn durch das Aktivieren einer Fläche ein Makro aufgerufen wird, ist es trivial, eine Referenz zum „Data model" des Kontrollelements herzustellen.
- Vom Data model eines Kontrollelements kann ich nicht so einfach die grafische Form oder die Zelle entnehmen, zu der das Kontrollelement gehört.
- Von der grafischen Form in einem Tabellendokument kann ich leicht die Zelle finden – falls das Kontrollelement überhaupt an einer Zelle verankert ist.

Meine Lösung war, die grafische Form zu finden, die mit einem Kontrollelement assoziiert ist. Es ist leicht, den Namen der Kontrolle zu finden, aber ein solcher Name muss nicht einzigartig sein. Ich habe mich dazu entschieden, mir jede grafische Form in der Folie anzusehen und zu überprüfen, ob das Kontrollelement der gefundenen Form identisch mit dem vom Nutzer aktivierten Kontrollelement ist. (Zum Verständnis von Folien siehe Abschnitt 13.9. Folien: XDrawPagesSupplier.)

Listing 226. *Sucht die grafische Form eines bestimmten Kontrollelements.*

```
Function FindCellWithControl(oDrawPage, oControl)
  Dim oShape      'Die graphische Form.
```

```
  Dim oAnchor     'Die mit der graphischen Form im Tabellenblatt verankerte Zelle.
  oShape = FindShapeForControl(oDrawPage, oControl)
  If Not IsEmpty(oShape) Then
    If oShape.getAnchor().supportsService("com.sun.star.sheet.SheetCell") Then
      FindCellWithControl = oShape.getAnchor()
    End If
  End If
End Function

Function FindShapeForControl(oDrawPage, oControl)
  Dim i
  Dim oShape      'Eine graphische Form.

  REM Schleife über die grafischen Formen in der Folie.
  For i = 0 To oDrawPage.getCount() - 1
    oShape = oDrawPage.getByIndex(i)
    If oShape.supportsService("com.sun.star.drawing.ControlShape") Then
      If EqualUNOObjects(oControl, oShape.Control) Then
        FindShapeForControl = oShape
        Exit Function
      End If
    End If
  Next
End Function
```

Der Code gehört zu einer Ereignisbehandlung (Event-Handler), die von einer Schaltfläche ausgelöst wird, um die Zelle zu finden, die eben diese Schaltfläche enthält. (Zum Verständnis von Listeners und Handlers siehe Abschnitt 10.10. UNO-Listeners und Handlers.)

```
REM Der Parameter oEvent enthält Informationen über das auslösende Ereignis.
REM Seine Eigenschaft .Source enthält eine Referenz auf das Kontrollelement.
Sub ButtonHandler(oEvent)
  Dim oControlModel   'Das Modell (Data model) des Kontrollelements.
  Dim oParent         'Das Objekt, von dem das Kontrollelement abstammt.
  Dim oCell           'Die verknüpfte Zelle.

  'Print oEvent.Source.Model.getName()
  oControlModel = oEvent.Source.Model

  oParent = oControlModel.getParent()
  Do While Not oParent.supportsService("com.sun.star.sheet.SpreadsheetDocument")
    oParent = oParent.getParent()
  Loop

  oCell = FindCellWithControl _
          (oParent.getCurrentController().getActiveSheet().getDrawPage(), _
           oControlModel)
  If Not IsEmpty(oCell) Then
    Print "Das Kontrollelement ist in der Zelle " & oCell.AbsoluteName
  Else
    Print "Keine Zelle mit Kontrollelementen zu finden"
  End If
End Sub
```

10.8. Eingebaute globale UNO-Variablen

Basic verfügt über eingebaute globale Variablen zum schnellen Zugriff auf häufig genutzte OOo-Komponenten. Die bei weitem beliebteste Variable ist ThisComponent, die das aktuell aktive Dokument referenziert. Zur Demonstration der Variablen ThisComponent gibt das Makro im Listing 227

alle Formatvorlagen des aktuellen Dokuments aus. Bild 76 ist nur einer der vielen vom Makro erstellten Ausgabedialoge. Der Umstand, dass in einem Dokument eine Formatvorlage existiert, heißt nicht, dass sie auch im Dokument genutzt wird.

Listing 227. *Gibt alle in diesem Dokument bekannten Formatvorlagen aus.*

```
Sub DisplayAllStyles
  Dim vFamilies As Variant 'Alle Formatvorlagetypen
  Dim vFamNames As Variant 'Array mit den Namen der Formatvorlagetypen
  Dim vStyles As Variant   'Ein Formatvorlagetyp wie Listen- oder Seitenvorlagen
  Dim vStlNames As Variant 'Array mit den Namen der einzelnen Formatvorlagen
  Dim s As String          'Ausgabetext
  Dim n As Integer         'Zähler für den Durchlauf durch die Vorlagetypen
  Dim i As Integer         'Zähler für den Durchlauf durch die Vorlagen

  vFamilies = ThisComponent.StyleFamilies           'Holt die Vorlagen
  vFamNames = vFamilies.getElementNames()           'Welche Vorlagetypen?
  For n = LBound(vFamNames) To UBound(vFamNames)    'Durchläuft alle Vorlagetypen
    s = ""
    vStyles = vFamilies.getByName(vFamNames(n))     'Holt die Vorlagen eines Typs
    vStlNames = vStyles.getElementNames()           'Die Namen aller Vorlagen des Typs
    For i = LBound(vStlNames) To UBound (vStlNames) 'Durchläuft alle Vorlagen des Typs
      s = s & i & " : " & vStlNames(i) & Chr$(13)   'Baut den Ausgabestring
      If ((i + 1) Mod 35 = 0) Then                  'in Gruppen zu je 35 Zeilen
        MsgBox s, 0, vFamNames(n)                   'Die Ausgabe im Dialog
        s = ""                                      'String leeren, Neuaufbau
      End If
    Next i                                          'Die nächsten Vorlagen
    If Len(s) > 0 Then MsgBox s, 0, vFamNames(n)    'Der Rest der Vorlagen des Typs
  Next n                                            'Nächster Vorlagentyp
End Sub
```

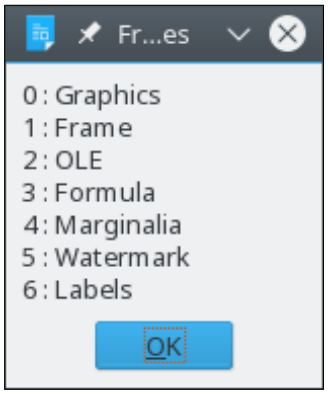


Bild 76. *In dem aktuellen Dokument bekannte Rahmenvorlagen.*

OOo basiert auf demselben Code wie StarOffice, das damals einen Desktop hatte. Alle einzelnen Fenster waren in diesem Desktop enthalten. Das Desktopmodell gibt es nicht mehr, aber aus Gründen der Rückwärtskompatibilität wirkt immer noch ein Desktop-Objekt als die globale Anwendung, die alle Dokumente einbindet. Obwohl ich häufig in Codes sehe, dass ein Desktop-Service mit der Funktion CreateUnoService erzeugt wird, so ist das aber in Basic nicht notwendig. Basic stellt die Variable StarDesktop für den Haupt-Desktop-Service in OOo zur Verfügung. Das Makro im Listing 228 zeigt, wie man mit dem StarDesktop durch alle aktuell offenen Dokumente wandern kann. Die Funktion GetDocType ist im Listing 219 definiert.

Listing 228. *Inspizierung aller offenen Komponenten.*

```
Sub IterateThroughAllDocs
  On Error Resume Next    'Nur Komponenten, die Dokumente sind
  Dim vComponents         'Alle Komponenten
  Dim vDocs               'Enumeration der Dokumente
  Dim vDoc                'Ein einzelnes Dokument
  Dim s As String
  GlobalScope.BasicLibraries.loadLibrary("Tools")'Enthält FileNameOutOfPath
  vComponents = StarDesktop.getComponents()       'Holt alle Komponenten
  vDocs = vComponents.createEnumeration()         'Enumeriert sie
  Do While vDocs.hasMoreElements()                'Solange noch welche da sind
    vDoc = vDocs.nextElement()                    'Holt die nächste Komponente
    s = s & GetDocType(vDoc) & "    "             'Der Dokumenttyp in den String
    s = s & FileNameOutOfPath(vDoc.getURL())      'Dazu der Dateiname
    s = s & Chr$(10)                              'Neue Zeile
  Loop
  MsgBox s, 0, "Aktuell geöffnete Komponenten"
End Sub
```

Tipp Die sichtbaren Hauptfenster in OOo werden Komponenten genannt. Jedes geöffnete Dokument ist eine Komponente, genauso die Basic-IDE und das Hilfefenster. In OOo bezeichnet das Wort „component“ fast immer ein geöffnetes Dokument.

Beim Durchlauf durch die geöffneten Dokumente (components) kann es sein, dass Sie unerwartete Dokumente vorfinden. Dabei handelt es sich um Komponentenfenster wie die Basic-IDE und das Hilfefenster. Das Makro im Listing 228 verwendet die Funktion FileNameOutOfPath. Das ist ein Makro und keine in Basic eingebaute Funktion. Sie ist im Modul Strings der Anwendungsbibliothek Tools gespeichert. Sie können Prozeduren einer Bibliothek nur dann aufrufen, wenn Sie die Bibliothek vorher geladen haben.

Die Variable GlobalScope referenziert die Anwendungsbibliotheken und wird benötigt, die Bibliothek Tools zu laden. Das Laden einer Bibliothek lädt alle Module dieser Bibliothek. Die OOo-Installation enthält Bibliotheken und Prozeduren, die nicht in Basic eingebaut sind. Rufen Sie die Methode LoadLibrary auf, bevor Sie die Routinen der Bibliotheken nutzen.

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

Zum Zugriff auf die Basic-Bibliotheken des aktuellen Dokuments verwenden Sie entweder die globale Variable BasicLibraries oder die Eigenschaft BasicLibraries des aktuellen Dokuments.

```
Print EqualUnoObjects(vObj.BasicLibraries, BasicLibraries)  'True
```

Mit der Variablen DialogLibraries haben Sie Zugriff auf die Dialog-Bibliotheken des aktuellen Dokuments. Im Gegensatz zu BasicLibraries hat ein einzelnes Dokument keine Eigenschaft mit dem Namen DialogLibraries, mit der man direkt die Dialog-Bibliotheken eines spezifischen Dokuments ansprechen kann. Dennoch können Sie die Basic- und Dialog-Bibliotheken eines Dokuments ganz leicht über eine indirekte Route erreichen, denn jedes Dokument besitzt die Eigenschaft LibraryContainer.

```
ThisComponent.LibraryContainer.getByName("OOME_4_0_deutsch").getModuleContainer()
ThisComponent.LibraryContainer.getByName("OOME_4_0_deutsch").getDialogContainer()
```

Die Methode getByName() der LibraryContainer-Eigenschaft gibt die benannte Bibliothek zurück. Die Methode getModuleContainer() gibt den Basic-Container, die Methode getDialogContainer() den Dialog-Container der spezifischen Bibliothek zurück. Der Code im Listing 229 verwendet jedoch die Variablen DialogLibraries und BasicLibraries, um die Anzahl der Dialoge und Module jeder Bibliothek des aktuellen Dokuments aufzulisten. Das Ergebnis sehen Sie im Bild 77.

Listing 229. *Blick auf die im aktuellen Dokument gespeicherten Bibliotheken und Dialoge.*

```
Sub ExamineDocumentLibraries
  Dim vLibs           'Die Bibliotheksnamen
  Dim vMod            'Die Modul-/Dialog-Objekte
  Dim nNumMods%       'Anzahl der Module oder Dialoge in einer Bibliothek
  Dim i%              'Temporäre Indexvariable
  Dim s$              'Temporäre Stringvariable
  s = "*** Dialog-Bibliotheken im Dokument" & Chr$(10) 'Initialisierung von s
  vLibs = DialogLibraries.getElementNames()            'Bibliotheksnamen
  For i = LBound(vLibs) To UBound(vLibs)               'Jeder einzelne Name
    vMod = DialogLibraries.getByName(vLibs(i))         'Holt die Dialog-Bibliothek
    nNumMods = UBound(vMod.getElementNames()) + 1      'Wie viele Dialoge?
    s = s & vLibs(i) & " hat " & nNumMods & " Dialoge" 'Aufbau des Ausgabestrings
    s = s & Chr$(10)
  Next i
  s = s & Chr$(10)
  s = s & "*** Basic-Bibliotheken im Dokument" & Chr$(10) 'Nun die Code-Bibliotheken
  vLibs = BasicLibraries.getElementNames()             'Bibliotheksnamen
  For i = LBound(vLibs) To UBound(vLibs)               'Jeder einzelne Name
    vMod = BasicLibraries.getByName(vLibs(i))          'Holt die Code-Bibliothek
    nNumMods = UBound(vMod.getElementNames()) + 1      'Wie viele Module?
    s = s & vLibs(i) & " hat " & nNumMods & " Module"  'Aufbau des Ausgabestrings
    s = s & Chr$(10)
  Next i

  MsgBox s, 0, "Bibliotheken"
End Sub
```

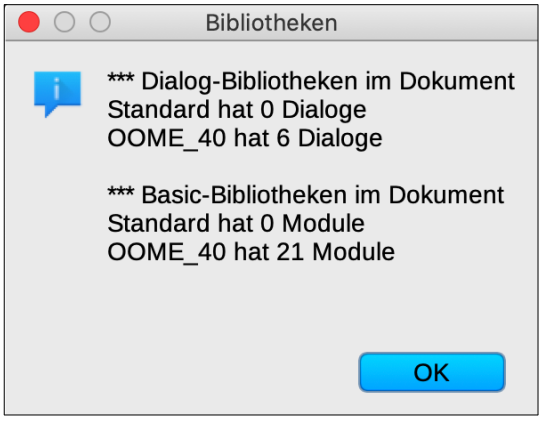


Bild 77. *Die Bibliotheken und Dialoge des aktuellen Dokuments.*

Um die Bibliotheken im Bibliothekscontainer der Anwendung zu sehen, modifizieren Sie den Code im Listing 229. Setzen Sie GlobalScope vor jedes Vorkommen von BasicLibraries und DialogLibraries.

10.9. Objekte und Eigenschaften suchen

Basic hat Funktionen zur namensbasierten Suche von Variablen und ihren Eigenschaften. Mir wurde ursprünglich diese Funktionalität als schlecht dokumentiert, möglicherweise veraltet und buggy beschrieben – und ich kann nicht widersprechen. Mit FindObject erhält man eine Referenz auf eine Variable des gesuchten Namens, und mit FindPropertyObject eine Referenz auf die benannte Eigenschaft eines Objekts. Das Makro im Listing 230 zeigt manche der Eigentümlichkeiten der Funktionen FindObject und FindPropertyObject.

Achtung Benutzen Sie keinesfalls die Funktionen FindObject und FindPropertyObject – sie sind schlecht dokumentiert, buggy und werden wahrscheinlich veralten. Ich erwähne sie hier nur der Vollständigkeit halber.

Listing 230. *Test der Methode FindObject.*

```
Sub TestFindObject
  Dim oTst
  Dim oDoc
  Dim s$

  oTst = FindObject("oDoc")

  s = "oDoc ist eine Variantvariable. " & Chr$(10) & _
      "Beim Start ist IsNull(oDoc) = " & IsNull(oDoc) & Chr$(10) & _
      "und IsEmpty(oDoc) = " & IsEmpty(oDoc) & Chr$(10) & Chr$(10) & _
      "oTst = FindObject(""oDoc"")" & Chr$(10) & _
      "IsNull(oTst) = " & IsNull(oTst) & _
      " und (oTst Is oDoc) = " & (oTst Is oDoc)

  MsgBox s

  REM Test mit dem aktuellen Dokument
  oDoc = ThisComponent
  oTst = FindObject("oDoc")

  REM Ja
  If oDoc Is ThisComponent Then Print "oDoc ist ThisComponent nach der Zuweisung."
  REM Nein
  If oTst Is oDoc Then Print "FindObject kann oDoc finden."
  REM Nein
  If oTst Is ThisComponent Then Print "oTst ist NICHT ThisComponent"

  REM Noch einmal, aber ohne Suche nach oDoc
  oDoc = ThisComponent
  oTst = FindObject("ThisComponent")
  REM Ja
  If oTst Is oDoc Then Print "oTst und oDoc sind dasselbe Objekt."
  REM Ja
  If oTst Is ThisComponent Then Print "oTst und ThisComponent sind dasselbe Objekt."
  REM Ja
  If oDoc Is ThisComponent Then Print "oDoc und ThisComponent sind dasselbe Objekt."

  REM ThisComponent hat die Eigenschaft DocumentInfo
  oTst = FindPropertyObject(ThisComponent, "DocumentInfo")
  REM Ist aber Null
  If IsNull(oTst) Then Print "Das Objekt DocumentInfo ist Null."

  REM Lädt die Bibliothek Gimmicks
  GlobalScope.BasicLibraries.loadLibrary("Gimmicks")

  REM Findet die Bibliothek, auch wenn sie keine Variable ist
  oDoc = FindObject("Gimmicks")
  Inspect oDoc

  REM Userfields ist ein Modul in der Bibliothek Gimmicks
  oTst = FindPropertyObject(oDoc, "Userfields")
```

```
    REM Mit Bibliotheken funktioniert es!
    Print "oTst Is Gimmicks.Userfields = " & (oTst Is Gimmicks.Userfields)     'True

    'Die Funktion StartChangesUserfields ist im Modul Userfields
    'Ruft die Routine auf
    oTst.Userfields.StartChangesUserfields
End Sub
```

Wenn Sie diese Funktion brauchen können, verwenden Sie sie. Ich habe versucht, mit ihnen einen Objektinspektor zu schreiben, aber die Funktion FindPropertyObject hat nicht so gearbeitet, wie ich es erwartet habe.

10.10. UNO-Listeners und Handlers

Wenn während eines Programmablaufs Ereignisse von außen kommen, zum Beispiel eine Tastatureingabe oder ein Druckereignis oder ein Mausklick, so kümmert sich normalerweise die Anwendung darum, wie damit umzugehen ist. Gelegentlich benötigen Sie aber Arbeitsschritte, die mit Makros gesteuert auf diese Ereignisse reagieren.

Man darf sich das vorstellen wie eine Rundfunkübertragung. Der Sender (englisch Broadcaster) erhält ein Ereignis, sagen wir ein Popkonzert. Er sendet dieses Ereignis weiter an die Hörer (englisch Listeners), die ihr Empfangsgerät auf diesen Sender eingestellt haben. Die Hörer können nun auswählen, was sie damit anfangen: zuhören, mitschneiden, tanzen, was auch immer.

In Makros wirkt ein Interface-Objekt als Broadcaster, wenn ihm ein Listener (auch ein Interface) für einen dem Broadcaster bekannten Ereignistyp zugewiesen wird. Ereignisse sind für UNO-Objekte nicht auf Unterbrechungen von außen beschränkt, sondern umfassen auch anwendungsinterne Geschehnisse, zum Beispiel das Laden oder Speichern eines Dokuments oder eine Änderung der Textauswahl. Welche Ereignisse erkannt werden, hängt vom konkreten Broadcaster ab. Ein solches Ereignis wird dann vom Broadcaster an den Listener weitergeleitet, konkret an spezielle Listener-Methoden. Welche es sind, ist abhängig vom eingesetzten Listener. Für Mausklicks sind es andere als für Tastatureingaben oder für Druckereignisse.

Man könnte jetzt denken, es sei einfacher, wenn der Broadcaster direkt die gewünschte Basic-Prozedur aufruft. Doch auch wenn in Basic-Prozeduren zuhauf UNO-Objektmethoden aufgerufen werden, so können umgekehrt UNO-Objekte keine Basic-Prozeduren aufrufen. Listeners verwenden einen Trick, so etwas zu ermöglichen. Beim Erzeugen eines Listeners wird ein beliebiger String festgelegt, der als Präfix zusammen mit dem eigentlichen Namen der Listener-Methode den Namen der Basic-Prozedur bildet, die Sie bereitstellen müssen. Wenn Sie beispielsweise als Präfix „Test_“ bestimmt haben und der Broadcaster die Listener-Methode „keyPressed“ aufruft, so leitet der Listener weiter zur Subroutine „Test_keyPressed“.

Für jede Listener-Methode sollten Sie immer eine Subroutine nach diesem Muster vorsehen, auch wenn Sie sie nicht benötigen. Lassen Sie sie in diesem Fall einfach leer, das heißt ohne Codeinhalt.

Achtung Wenn Sie eine für das Listener-Interface benötigte Methode nicht einsetzen, kann der Listener OOo zum Absturz bringen.

Listeners bieten die Möglichkeit, beim Eintreten eines Ereignisses Dinge zu erledigen, die von dem Ereignis abhängen. Sie können zum Beispiel zählen, wie oft eine bestimmte Taste gedrückt wurde. Sie können aber nicht verhindern, dass der Tastendruck von der Anwendung verwertet wird.

Sollten Sie jedoch ein Ereignis abfangen wollen, um etwas anderes anstelle der normalen Programmroutine auszuführen, so benötigen Sie einen Handler. Sie können zum Beispiel, wenn Sie wollen, immer wenn die Taste z gedrückt wird, ein y ausgeben. Ein Handler ist nichts anderes als ein Listener mit dem Unterschied, dass seine Methoden einen Rückgabewert besitzen, der dem Broadcaster mit-

teilt, ob das Ereignis durch die Handler-Methode erledigt ist. Daher müssen die Basic-Prozeduren Funktionen mit booleschem Rückgabewert sein. Geben sie True zurück, ist das Ereignis für den Broadcaster erledigt. Geben sie False zurück, leitet der Broadcaster das Ereignis an das ansonsten vorgesehene Objekt zur Ereignisbehandlung weiter.

10.10.1. Ihr erster Listener

Alle Listeners (und Handlers) stammen vom Interface com.sun.star.lang.XEventListener ab und müssen dessen einzige Methode einsetzen: disposing(Ereignisobjekt). Sie wird in dem Moment aufgerufen, wenn der Broadcaster unerreichbar wird. Wenn der Broadcaster also nicht mehr existiert, kann er nicht mehr verwendet werden, zum Beispiel wenn ein Druckauftrag erledigt ist und nicht mehr gebraucht wird oder wenn ein Dokument geschlossen wird. Das Listing 231 zeigt den einfachsten aller möglichen Listeners, XEventListener. Beachten Sie das Präfix „first_listen_“ vor dem eigentlichen obligatorischen Namen „disposing“. Ich habe diese Wendung gewählt, weil Sie sprechend ist.

Listing 231. *Ein einfacher Listener, der nichts tut.*

```
Sub first_listen_disposing(vArgument)
  MsgBox "Entfernt den ersten Listener"
End Sub
```

Sie brauchen für diesen einfachen Listener nur Listing 231. Um keine Missverständnisse aufkommen zu lassen: das Listing ist nicht der Listener. Sie müssen noch mit CreateUNOListener einen Listener erzeugen, der das Makro im Listing 231 aufrufen kann. CreateUNOListener benötigt zwei Argumente: das Präfix und den Namen des zu erzeugenden Service.

Die Routine des Listing 232 erzeugt den UNO-Listener und verbindet ihn über das Präfix mit der Routine des Listing 231. Wenn die Methode disposing des Objekts vListener aufgerufen wird, wird schließlich die Routine im Listing 231 aufgerufen.

Tipp Der Code im Listing 232 erzeugt einen Listener und ruft danach die Objektmethode disposing auf. Da der Listener über das Präfix im Listing 231 eingebunden ist, wird schließlich die Subroutine first_listen_disposing im Listing 231 aufgerufen.

Listing 232. *Erzeugt einen Listener und ruft dann die Methode disposing auf.*

```
Sub MyFirstListener
  Dim vListener                                  'Der Listener
  Dim vEventObj As New com.sun.star.lang.EventObject 'Das Ereignis
  Dim sPrefix$                                   'Das Präfix
  Dim sService$                                  'Der Name des Service
  sPrefix = "first_listen_"
  sService = "com.sun.star.lang.XEventListener"
  vListener = CreateUnoListener(sPrefix, sService)
  vListener.disposing(vEventObj)
End Sub
```

Im Listing 232 lebt der Listener nur innerhalb der Sub-Routine. Er ist außerdem mit keinem Broadcaster verknüpft, so dass er über kein Ereignis informiert werden kann. Darüber hinaus ist XEventListener nur der Prototyp eines Listeners, von dem alle spezialisierten Listeners abstammen. Listing 232 zeigt nur, wie ein Listener erzeugt und wie er wieder entfernt wird. Seine „disposing“-Methode werden Sie allerdings nicht in einem echten Makro verwenden. Sie wird automatisch von OOo aufgerufen, wenn ein Broadcaster verschwindet, zum Beispiel beim Schließen eines Dokuments.

10.10.2. Voraussetzungen für den Einsatz eines Listeners

Die beiden schwierigsten Teile bei der Installation eines Listeners sind die Auswahl einerseits des Broadcasters und andererseits des speziellen Listener-Interface. Die restlichen Schritte sind einfacher.

1. Wählen Sie den zu benutzenden Broadcaster aus.
2. Wählen Sie das einzusetzende Listener-Interface aus.
3. Erstellen Sie eine globale Variable für den Listener und vielleicht auch für den Broadcaster. (Prüfen Sie sich selbst, ob Sie verstanden haben, warum die Variable global sein muss, bevor Sie den anschließenden Tipp lesen.)
4. Legen Sie das Präfix fest, das Sie verwenden wollen. Es sollte selbsterklärend sein.
5. Schreiben Sie Subroutinen, bzw. Funktionen für **alle** Listener-Methoden.
6. Erzeugen Sie den UNO-Listener und weisen ihn der globalen Variablen zu.
7. Verknüpfen Sie den Listener mit dem Broadcaster.
8. Wenn Sie ihn nicht mehr brauchen, entfernen Sie den Listener vom Broadcaster.

Tipp Warum soll der Listener einer globalen Variablen zugewiesen werden? Die Erstellung des Listeners und die Zuweisung zum Broadcaster geschehen normalerweise in einer Subroutine, die danach beendet ist. Der Listener muss aber noch existieren, wenn die Prozedur nicht mehr läuft. In Basic geht das nur mit einer globalen Variablen. Später wird eine andere Prozedur den Listener vom Broadcaster entfernen. Diese Prozedur muss Zugriff auf den Listener haben.

Im Abschnitt 13.15.3. Beispiel für einen Druck-Listener in Calc wird ein Listener für Druckereignisse vorgestellt.

Es ist nicht immer einfach zu wissen, welcher Broadcaster zu nutzen ist. Man braucht schon ein gewisses Maß an Verständnis für das Design von OOo, aber das kommt mit der Erfahrung. Normalerweise ist der Start der Suche naheliegend – nämlich das aktuelle Dokument, das man mit der Variablen ThisComponent erreicht und das die Dokumentdaten enthält. Zum Beispiel akzeptiert das Dokument den XPrintJobListener, der den Druckverlauf überwacht. Die meisten Aufgaben, die Benutzerinteraktionen betreffen – Textauswahl, Mausbewegungen und Tastatureingaben –, laufen durch den Dokument-Controller. Auch Dialoge und Steuerelemente fungieren als Broadcaster.

Nach der Wahl des Broadcasters suchen Sie nach dessen Objekt-Methoden mit Namen wie „add...listener“ (für Handlers „add...handler“). Inspizieren Sie die Objekteigenschaft „dbg_methods“, um herauszufinden, welche Listeners unterstützt werden (jeder unterstützte Listener hat eine „add“ und eine „remove“-Methode).

Ein anderer Weg geht über den produzierten Ereignistyp (suchen Sie nach „XEventListener“ auf der Site http://api.libreoffice.org/docs/idl/ref/index.html, beziehungsweise über die Suchbox der Site https://api.libreoffice.org/docs/idl/ref/index.html). Dann sucht man Broadcaster, die den benötigten Ereignistyp unterstützen. Ich durchsuche gerne den OOo Developer’s Guide und das Internet. Zu weiteren Methoden zur Suche nach Informationen s. Kapitel 19. Informationsquellen.

Nehmen wir an, Sie wollen Änderungen der Textauswahl entdecken. Diese Aufgabe übernimmt ein XSelectionChangeListener. Dieses Interface definiert nur eine Methode, die eingebunden sein muss – selectionChanged(Ereignisobjekt). Da alle Listeners vom XEventListener abstammen, muss auch dessen Methode disposing(Ereignisobjekt) eingebunden sein. Der schnellste Weg, alle einzusetzenden Subroutinen und Funktionen herauszufinden, besteht darin, mit der Funktion CreateUNOListener einen Listener zu erzeugen und dann die Eigenschaft dbg_methods (s. Listing 233 und Bild 78) zu inspizieren.

Listing 233. *Welche Methoden werden von einem bestimmten Listener unterstützt?*

```
Sub InspectListenerMethods
  Dim sPrefix$
  Dim sService$
  Dim vService
  sPrefix = "sel_change_"
  sService = "com.sun.star.view.XSelectionChangeListener"
  vService = CreateUnoListener(sPrefix, sService)
  DisplayDbgInfoStr(vService.dbg_methods, ";", 35, "Methoden")
End Sub
```

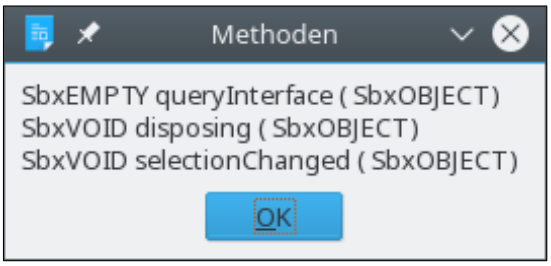


Bild 78. *Methoden im Interface XSelectionChangeListener.*

Die Methode queryInterface stammt vom Interface com.sun.star.uno.XInterface, von dem alle Interfaces abstammen, also auch XEventListener. Sie können diese Methode ignorieren, denn sie wird vom Broadcaster nicht verwendet, weil sie nichts mit Ereignissen zu tun hat. Sie müssen jedoch den Rest der Routinen definieren, denn der Broadcaster kann nicht wissen, welche Methode Sie für wichtig halten. Im Falle eines Ereignisses nutzt er die passende Methode, und wenn der Listener keine dazu gehörende Basic-Prozedur findet, gibt es einen Laufzeitfehler.

Denken Sie sich ein selbsterklärendes Präfix aus, so dass der Code mehr als nur einen Listener enthalten kann. Die Namen der Prozeduren müssen die Form Präfix plus originalen Namen der Listener-Methode haben.

Der Broadcaster liefert Informationen über das auslösende Ereignis in einem UNO-Struct, dessen Eigenschaften jeweils speziell auf den betreffenden Ereignistyp abgestimmt sind. Alle Ereignisobjekte stammen von com.sun.star.lang.EventObject ab. Dieses enthält nur einen Zeiger auf das Broadcaster-Objekt, und zwar in der Eigenschaft „Source“, die somit auch an alle speziellen Ereignisobjekte vererbt wird.

Wenn Sie in Ihren Listener-Prozeduren diese Informationen verwenden, müssen Sie einen Variant-Parameter für das Ereignis-Struct definieren. Wenn Sie nicht darauf zugreifen wollen, können Sie den Parameter weglassen. Er ist also praktisch optional, ohne das Schlüsselwort „Optional“. Dieser Parameter steht auch Handlers zur Verfügung, die mit Steuerelementen manuell über deren Eigenschaften-Dialog verknüpft werden (als Beispiel s. Listing 226).

10.10.3. Listener für Auswahländerungen

Ein Listener zur Auswahländerung wird beim CurrentController als Broadcaster angemeldet. Als Präfix für die Namen der aufzurufenden Subroutinen wählen wir willkürlich „sel_change_“, weil es den Zusammenhang deutlich macht. Wir benötigen zwei Subroutinen: für die Listener-Methoden „disposing“ und „selectionChanged“ (s. Bild 78 und Listing 234).

Listing 234. *Prozeduren für die Listener-Methoden disposing und selectionChanged.*

```
'Alle Listeners müssen dieses Ereignis unterstützen.
'Der Ereignis-Parameter muss nicht definiert werden.
Sub sel_change_disposing
  MsgBox "Entfernt den Auswahländerungs-Listener"
End Sub
```

```
Sub sel_change_selectionChanged(vEvent)
  REM vEvent ist ein Struct com.sun.star.lang.EventObject
  REM mit der einzigen Eigenschaft "Source".
  Dim vCurrentSelection           'Das Objekt der aktuellen Auswahl
  REM Zur Inspektion des Ereignisobjekts:
  'MsgBox vEvent.dbg_properties
  vCurrentSelection =  vEvent.source
  Print "Anzahl der selektierten Bereiche = " &_
        vCurrentSelection.getSelection().getCount()
End Sub
```

Die Funktion CreateUnoListener im Listing 235 erzeugt einen Listener, verknüpft ihn mit einem Broadcaster und ist damit beendet. Der Broadcaster bleibt als globale Variable lebendig und ruft den Listener auf, wenn etwas geschieht. Auch die Variable für den Listener muss global sein, damit er später erreichbar ist – zum Beispiel, wenn der Broadcaster eine seiner Methoden aufruft oder wenn es an der Zeit ist, ihn vom Broadcaster abzumelden.

Listing 235. *Startet die Beobachtung der Auswahländerungs-Ereignisse.*

```
Global vSelChangeListener   'Muss global sein
Global vSelChangeBroadCast

'Zum Start der Überwachung führen Sie dieses Makro aus.
Sub StartListeningToSelChangeEvents
  Dim sPrefix$
  Dim sService$
  sPrefix = "sel_change_"
  sService = "com.sun.star.view.XSelectionChangeListener"

  REM Das Anmeldungsobjekt zur Beobachtung der Auswahländerungen
  REM ist der CurrentController
  vSelChangeBroadCast = ThisComponent.getCurrentController

  'Erzeugt einen Listener zum Abfangen der Auswahländerungs-Ereignisse
  vSelChangeListener = CreateUnoListener(sPrefix, sService)

  'Registriert den Listener beim Dokument-Controller
  vSelChangeBroadCast.addSelectionChangeListener(vSelChangeListener)
End Sub
```

Das Verhalten des Codes im Listing 234 ist nicht optimiert und außerdem mehr als ärgerlich. Jedes Mal, wenn sich die Auswahl ändert, wird der Code aufgerufen, der einen Dialog auf dem Bildschirm öffnet. Das verhindert die Textauswahl. Also schnell wieder weg damit. Mit dem Code im Listing 236 wird der Listener wieder vom Broadcaster abgemeldet. Als globales Objekt existiert er dennoch weiter und kann später wieder mit einem Broadcaster verknüpft werden.

Listing 236. *Beendet die Beobachtung der Auswahländerungs-Ereignisse.*

```
Sub StopListeningToSelChangeEvents
  ' Entfernt den Listener
  vSelChangeBroadCast.removeSelectionChangeListener(vSelChangeListener)
End Sub
```

Auch wenn Sie wie im Listing 236 den Listener vom Broadcaster abmelden, so sollten Sie dennoch nicht auf die Subroutine „sel_change_disposing“ verzichten. Denn wenn Sie das Dokument vor dem Aufruf von StopListeningToSelChangeEvents schließen, wird gleichzeitig auch der Broadcaster entfernt und in diesem Zusammenhang der mit ihm verknüpfte Listener abgehängt. Dazu wird dessen

„disposing“-Methode aufgerufen. Wenn kein Sub sel_change_disposing gefunden wird, gibt es eine Fehlermeldung, deren Bedeutung Sie sich dann wahrscheinlich nicht erklären können.

So testen Sie den Listener:

1. Starten Sie die Beobachtung.
2. Probieren Sie sehr einfache Auswahländerungen aus: Klicken Sie einfach irgendwohin. Umschalt+Click für mehr. Strg+Click für eine zweite Auswahl.
3. Beenden Sie die Beobachtung.

Achtung Sichern Sie alle offenen Dokumente, wenn Sie Listeners testen, ein kleiner Fehler kann ausreichen, um OOo abstürzen zu lassen.

10.10.4. Handler für Tastatureingaben (V. Lenhardt)

Ein Handler ist ein Listener mit dem zusätzlichen Merkmal, dass seine Methoden einen booleschen Rückgabewert haben, der dem Broadcaster mitteilt, ob das Ereignis erledigt ist oder ob es an das ansonsten vorgesehene Objekt zur Ereignisbehandlung weitergeleitet werden muss. Es sind nur für 4 Ereignistypen gleichermaßen Handler-Interfaces wie Listener-Interfaces definiert:

- XKeyHandler: Tastatur
- XMouseClickHandler: Maustasten
- XEnhancedMouseClickHandler: Maustasten + Info über das Objekt unter der Maus
- XMouseMotionHandler: Mausbewegung

Als Beispiel soll die simple Caesar-Verschlüsselung eines Textes schon bei der Eingabe dienen. Bei dieser Verschlüsselung wird ein Buchstabe durch einen anderen ersetzt, der im Alphabet eine bestimmte Anzahl von Stellen entfernt ist. In meinem Beispiel ist der Versatz drei Stellen nach vorn und zwar nicht nur für Buchstaben im Alphabet, sondern für die darstellbaren Zeichen in der Unicode-Tabelle. Dazu benötigen wir das Interface XKeyHandler, das am CurrentController angemeldet wird. Als Präfix habe ich „Caesar_“ festgelegt. XKeyHandler benötigt Funktionen für die beiden Methoden „keyPressed“ und „keyReleased“ sowie die obligatorische Subroutine „disposing“ (s. Listing 237).

Der Handler wird aktiviert, wenn eine Taste gedrückt (pressed) wird, aber auch wenn die gedrückte Taste wieder losgelassen (released) wird. Das Makro interessiert sich nicht für das „losgelassen“-Ereignis, daher enthält die Funktion Caesar_keyReleased keinen speziellen Code, sondern überlässt dem Broadcaster durch die „False“-Rückmeldung die Weiterleitung des Ereignisses. Im „gedrückt“-Fall übernimmt die Funktion Caesar_keyPressed die Texteingabe, vorausgesetzt, die gedrückte Taste liefert ein darstellbares Zeichen. Dann wird durch die „True“-Rückgabe der Funktion der Broadcaster daran gehindert, das Ereignis weiterzuleiten. Falls aber eine Taste oder eine Tastenkombination gedrückt wurde, die keine Zeichenausgabe, sondern etwas anderes bewirkt, beispielsweise „F1“, „Return“ oder auch Strg-a, gibt die Funktion „False“ zurück, und der Broadcaster aktiviert die Standardreaktion.

Die Informationen über die gedrückte Taste liefert das Ereignis-Struct als Argument des Funktionsaufrufs. In diesem Fall ist es com.sun.star.awt.KeyEvent, das seinerseits die Eigenschaft Modifiers vom InputEvent erbt (s. Tabelle 95). Auch das von Maus-Listeners gelieferte Struct com.sun.star.awt.MouseEvent erbt diese Eigenschaft.

Tabelle 95. *Eigenschaften des Structs com.sun.star.awt.KeyEvent.*

Eigenschaft	Beschreibung
KeyChar	Das von der Taste generierte Unicodezeichen. Im Falle einer funktionalen Taste ist es Unicode-0.
KeyCode	Der für jede Taste spezifische Codewert (Integer): Konstantengruppe com.sun.star.awt.Key, zum Beispiel NUM8 = 264 oder W = 534 oder F1 = 768.

Eigenschaft	Beschreibung
KeyFunc	Der für bestimmte logische Funktionen spezifische Codewert (Integer): Konstantengruppe com.sun.star.awt.KeyFunction, zum Beispiel DONTKNOW = 0 oder UNDO = 11.
Modifiers	Geerbt von com.sun.star.awt.InputEvent. Enthält den Code (Integer) für gleichzeitig gedrückte Umschalttasten: Konstantengruppe com.sun.star.awt.KeyModifier: 0 = Keine Umschalttaste gedrückt SHIFT = 1 – jede der beiden Umschalttasten MOD1 = 2 – Strg-Taste (Cmd-Taste bei Mac OS X) MOD2 = 4 – Alt-Taste MOD3 = 8 – Ctrl-Taste (Mac OS X)

Wenn Sie den Tastaturhandler im Listing 237 starten (Sub TestCaesar), wird ein neues Textdokument geöffnet, in dem die Caesar-Verschlüsselung wirkt. Beim Schließen dieses neuen Dokuments wird dessen Controller entfernt und somit auch die Verknüpfung zum Handler. Die im Listing 237 gezeigte Subroutine StartCaesarHandler verknüpft allerdings den Handler mit dem aktuellen Dokument.

Listing 237. *Makro zur Caesar-Verschlüsselung bei der Eingabe.*

```
Global oCaesarHandler    'Der CurrentController als Broadcaster.
Global oCaesarCurrCont   'Der Tastaturhandler.

Sub TestCaesar
  Dim oDoc
  oDoc = StarDesktop.loadComponentFromURL _
                    ("private:factory/swriter", "_blank", 0, Array())
  oCaesarCurrCont = oDoc.CurrentController
  oCaesarHandler = CreateUnoListener("Caesar_", "com.sun.star.awt.XKeyHandler")
  oCaesarCurrCont.addKeyHandler(oCaesarHandler)
End Sub

REM Nicht nur für Listeners, sondern auch für Handlers obligatorisch.
Sub Caesar_disposing
End Sub

REM Das Ereignis "Taste losgelassen" wird nicht beachtet. Die Rückgabe
REM "False" leitet das Ereignis an die Standardroutine weiter.
Function Caesar_keyReleased As Boolean
  Caesar_keyReleased = False 'Nicht notwendig, denn der Funktionswert ist eh False.
End Function

REM Behandlung des Ereignisses "Taste gedrückt". Informationen über das
REM Ereignis stecken im "KeyEvent"-Struct vKeyEvent.
Function Caesar_keyPressed(vKeyEvent) As Boolean
  Dim sChar As String
  sChar = vKeyEvent.KeyChar   'Das von der Tastatur gelieferte Zeichen.
  Dim iMod As Integer
  iMod = vKeyEvent.Modifiers  'Code für gedrückte Umschalttasten.

  REM Die Umsetzung wird nur für darstellbare Zeichen vorgenommen.
  REM Das schließt alle Zeichen kleiner als ASCII 32 aus, wie auch
  REM ASCII 127, das die Funktion DEL (Löschen) auslöst.
  REM Ausgeschlossen sind auch alle Tasten, die bei gedrückter Alt-
  REM oder Strg-Taste gedrückt werden, denn bei diesen ist iMod > 1.
  If Asc(sChar) > 31 And Asc(sChar) <> 127 And iMod < 2 Then
    InsertCaesarChar(sChar)   'Fügt das geänderte Zeichen ein.
    Caesar_keyPressed = True  'Ereignis wurde vom Handler verschluckt.
    Exit Function
```

```
  End If

  REM Hier landen alle anderen Tastaturereignisse.
  Caesar_keyPressed = False  'Ereignis ist nicht behandelt.
                             'Weiterleitung zur Standardbehandlung.
End Function

REM Fügt im Textdokument ein Zeichen an der aktuellen Cursorposition ein,
REM dessen Unicode-Nummer um 3 höher ist als die des vom Tastaturereignis
REM gelieferten Zeichens sChar.
REM Zum tieferen Verständnis s. Kapitel 14. Textdokumente.
Sub InsertCaesarChar(sChar As String)
  Dim oViewCursor  'Die sichtbare Cursorposition im Dokument.
  Dim oText        'Der Text des Dokuments (enthält nicht nur den reinen
                   'String, sondern auch andere Elemente wie Tabellen u.a.
                   'sowie Formatierungen und Formatvorlagen).
  Dim oCursor      'Ein Teilbereich des Dokumenttextes.
  Dim iAsc% : iAsc = Asc(sChar) 'Ist weniger zu schreiben.

  REM Der CurrentController muss nicht extra geholt werden,
  REM denn er wird ja durch die globale Variable oCaesarCurrCont referenziert.
  oViewCursor = oCaesarCurrCont.getViewCursor() 'Die aktuelle Cursorposition.
  oText = oViewCursor.getText()                 'Der Dokumenttext.
  oCursor = oText.createTextCursorByRange(oViewCursor.getStart())
                               'Die Textstelle, an der eingefügt werden soll.

  REM Kontrollanzeige des originalen und des ersetzenden Zeichens.
  MsgBox sChar & "(" & iAsc & ")" & " -> " _
         & Chr(iAsc + 3) & "(" & (iAsc + 3) & ")", _
         64, "Zeichenersetzung"

  REM Fügt das veränderte Zeichen an der Cursorposition ein.
  oText.insertString(oCursor.getStart(), Chr(iAsc + 3), False)
End Sub

REM Erzeugt den Handler und meldet ihn am CurrentController an.
REM Damit beginnt die Beobachtung.
Sub StartCaesarHandler
  Dim sPrefix As String
  Dim sService As String

  sPrefix = "Caesar_"
  sService = "com.sun.star.awt.XKeyHandler"

  oCaesarCurrCont = ThisComponent.CurrentController      'Globale Variable.
  oCaesarHandler = CreateUnoListener(sPrefix, sService)  'Globale Variable.

  oCaesarCurrCont.addKeyHandler(oCaesarHandler)
End Sub

REM Meldet den Handler vom CurrentConroller ab und beendet so die Beobachtung.
Sub StopCaesarHandler
  oCaesarCurrCont.removeKeyHandler(oCaesarHandler)
  Print "Caesar abgemeldet"
End Sub
```

10.10.5. Listener für Dokumentereignisse (V. Lenhardt)

Ein Listener, der am Controller angemeldet ist, hat das Problem, dass er nicht mehr funktioniert, wenn Sie aus der Bearbeitungssicht in die Seitenansicht gewechselt haben. Wahrscheinlich erwarten

Sie auch gar nicht, dass der Listener in dieser Druckvorschau seinen Dienst tut. Aber wenn sie zurück in die Bearbeitungssicht wechseln, tut er es immer noch nicht.

Der Grund liegt darin, dass beim Wechsel der Sicht der jeweilige Controller zerstört und durch einen neuen ersetzt wird. Es ist also notwendig, den aktuellen Controller jeweils neu mit der globalen Variablen zu referenzieren und den Listener wiederum bei ihm anzumelden.

Wenn der Listener XDocumentEventListener am jeweiligen Dokument angemeldet ist, gibt das Dokument als Broadcaster ein Ereignis wie den Wechsel der Sicht an die Listener-Methode „documentEventOccured“ weiter. Weitere Methoden hat der Listener nicht. Das mitgelieferte Ereignisobjekt ist das Struct com.sun.star.document.DocumentEvent mit den Eigenschaften:

- EventName – Name des Ereignisses (String)
- ViewController – der aktuelle Controller (Objekt)
- Supplement – weitere Informationen über das Ereignis.

Uns interessiert hier das Ereignis mit dem Namen „OnViewCreated“, das gemeldet wird, wenn eine neue Sicht erstellt wurde. Alle verfügbaren Namen finden Sie in der API als Service com.sun.star.document.Events, s.auch Tabelle 113. Der übermittelte Controller hat auch einen Namen, und zwar heißt er „PrintPreview“ für die Seitenansicht und „Default“ für die Bearbeitungssicht. Diesen Namen finden Sie als Eigenschaft „ViewControllerName“ des Controllerobjekts.

Mit diesem Rüstzeug können Sie nun auf den Wechsel der Sicht reagieren, s. Listing 238. In diesem Beispiel beziehe ich mich auf den im Listing 237 vorgestellten Tastaturhandler.

Listing 238. *Makro zur Beobachtung von Dokumentereignissen.*

```
Global oCaesarDoc       'Das Dokument, dessen Ereignisse beobachtet werden.
Global oDocEventListen  'Der Ereignis-Beobachter.

Sub Doc_event_disposing
End Sub

REM Das Ereignis steckt in vEvent, Struct com.sun.star.document.DocumentEvent:
REM EventName = Name als String, ViewController = Controller als Objekt
Sub Doc_event_documentEventOccured(vEvent)
  If vEvent.EventName = "OnViewCreated" Then
    REM Der Name des Controllers in der Bearbeitungssicht ist "Default",
    REM in der Seitenansicht heißt er "PrintPreview".
    REM Wir brauchen den Caesar-Listener nur in der Bearbeitungssicht.
    If vEvent.ViewController.ViewControllerName = "Default" Then
      oCaesarCurrCont = vEvent.ViewController       'Neureferenzierung des Controllers
      oCaesarCurrCont.addKeyHandler(oCaesarHandler) 'Neuanmeldung des Handlers
    End If
  End If
End Sub

REM Startet den Dokumentereignis-Beobachter.
Sub StartDocEventListener
  oCaesarDoc = ThisComponent
  oDocEventListen = CreateUnoListener( _
                    "Doc_event_", "com.sun.star.document.XDocumentEventListener")
  oCaesarDoc.addDocumentEventListener(oDocEventListen)
End Sub

REM Beendet den Dokumentereignis-Beobachter.
Sub StopDocEventListener
  oCaesarDoc.removeDocumentEventListener(oDocEventListen)
End Sub
```

Achtung Wenn Sie ein Textdokument, für dessen Controller Listeners angemeldet sind, in die Seitenansicht umschalten, wird die Controllerinstanz zerstört und durch eine neue Controllerinstanz ersetzt. Ebenso beim Umschalten zurück in die Bearbeitungssicht. Die Listeners haben dann ihren Broadcaster verloren und funktionieren nicht mehr. Verwenden Sie einen XDocumentEventListener, s. Listing 238.

10.11. Erzeugung eines UNO-Dialogs

Mit CreateUnoDialog erzeugen Sie einen schon definierten Dialog. Dieser Abschnitt zeigt Ihnen die Funktion CreateUnoDialog, er zeigt nicht, wie Dialoge aufgebaut werden. CreateUnoDialog benötigt ein einziges Argument, den kompletten Pfad zu einer Dialogdefinition.

```
CreateUnoDialog(GlobalScope.BasicLibraries.LibName.DialogName)
```

Die Variable GlobalScope bietet den Zugriff auf Bibliotheken der Anwendungsebene. Die Bibliothek Tools enthält den Dialog mit dem Namen DlgOverwriteAll. Sie müssen erst die Bibliothek laden, bevor Sie den enthaltenen Dialog nutzen können. Sie können die Bibliothek manuell laden oder direkt aus dem Makro. Über **Extras > Makros > Dialoge Verwalten** öffnen Sie den Dialog zur Verwaltung der Basic-Makros, mit dem man Bibliotheken manuell laden kann. In einem Makro laden Sie eine Bibliothek mit der Anweisung loadLibrary.

```
GlobalScope.BasicLibraries.loadLibrary("Tools")
CreateUnoDialog(GlobalScope.BasicLibraries.Tools.DlgOverwriteAll)
```

Wurde der Dialog innerhalb eines Dokuments definiert, verwenden Sie die Variable BasicLibraries.

```
CreateUnoDialog(BasicLibraries.LibName.DialogName)
```

Tipp Ein Ereignis (Event) besagt, dass etwas geschehen ist. Wenn bestimmte Dinge geschehen, wird ein Event-Objekt erzeugt, das eine Beschreibung dessen gibt, was geschehen ist. Das Event-Objekt wird an einen Listener geschickt oder zu irgendeiner anderen Routine, die beim Auftreten des Ereignisses aufgerufen werden soll. Sie können eine „Event"-Routine schreiben, die bei einem Ereignis, zum Beispiel beim Klick auf eine Schaltfläche, die Kontrolle übernimmt.

Wenn Sie einen Dialog konzipieren, schreiben Sie normalerweise Event-Handlers als Subroutinen in Basic, zum Beispiel tut eine Schaltfläche in einem Dialog gar nichts, wenn nicht Ereignisse an Subroutinen oder Funktionen gebunden werden, die Sie geschrieben haben. Sie können einem Dialog eine Schließen-Schaltfläche hinzufügen und dann dem zu dieser Schaltfläche gehörenden Ereignis „Aktion ausführen" ein Makro zuweisen, das den Dialog schließt. Wenn ein Makro, das von einem Event-Handler aufgerufen wird, auf einen laufenden Dialog zugreift, dann muss der Dialog in einer Variablen stecken, auf die das Makro Zugriff hat. Im allgemeinen definiere ich eine Variable als Private für diese Art von Daten, s. Listing 239. (Zum tieferen Verständnis von Dialogen siehe Kapitel 18. Dialoge und Steuerelemente.)

Listing 239. *Zeigt mit Hilfe eines Dialogs Informationen über ein Objekt an.*

```
Private MySampleDialog
Sub DisplayObjectInformation(Optional vOptionalObj)
  Dim vControl  'Zugriff auf das Text-Kontrollfeld des Dialogs
  Dim s$        'Temporäre Stringvariable
  Dim vObj      'Objekt, über das Informationen ausgegeben werden

  REM Ohne Angabe eines Objekts wird das aktuelle Dokument genommen.
  If IsMissing(vOptionalObj) Then
    vObj = ThisComponent
  Else
    vObj = vOptionalObj
  End If

  REM Erzeugt den Dialog und setzt den Titel
```

```
  MySampleDialog = CreateUnoDialog(DialogLibraries.OOME_40.MyFirstDialog)
  MySampleDialog.setTitle("Typ der Variablen: " & TypeName(vObj))

  REM Zugriff auf das Textfeld des Dialogs
  REM Ich habe diesen Text manuell hinzugefügt
  vControl = MySampleDialog.getControl("TextField1")

  If InStr(TypeName(vObj), "Object") < 1 Then
    REM Wenn dies KEIN Objekt ist, wird nur eine einfache Information ausgegeben
    vControl.setText(Dlg_GetObjTypeInfo(vObj))

  ElseIf Not HasUnoInterfaces(vObj, "com.sun.star.uno.XInterface") Then
    REM Es ist ein Objekt, aber kein UNO-Objekt
    REM Ich kann nicht HasUnoInterfaces aufrufen, wenn es kein UNO-Objekt ist
    vControl.setText(Dlg_GetObjTypeInfo(vObj))

  Else
    REM Dies ist ein UNO-Objekt, also der Zugriff auf die "dbg_"-Eigenschaften.
    REM Manchmal ist die Methode getImplementationName() nicht eingebunden.
    On Error Resume Next
    MySampleDialog.setTitle(" Typ der Variablen: " & vObj.getImplementationName())
    s = "*************** Methods **************" & Chr$(10) &_
        Dlg_DisplayDbgInfoStr(vObj.dbg_methods, ";") & Chr$(10) &_
        "*************** Properties **************" & Chr$(10) &_
        Dlg_DisplayDbgInfoStr(vObj.dbg_properties, ";") & Chr$(10) &_
        "*************** Services **************" & Chr$(10) &_
        Dlg_DisplayDbgInfoStr(vObj.dbg_supportedInterfaces, Chr$(10))
    vControl.setText(s)
  End If

  REM Der Dialog soll sich selbst starten
  MySampleDialog.execute()
End Sub
```

Tipp Deklarieren Sie den Dialog in einer Private-Variablen. Eine Private-Variable verhindert, dass ohne Not andere Module betroffen sind.

Das Makro im Listing 239 zeigt einen Dialog, der Informationen über ein Objekt ausgibt, das als Argument übergeben wird. Mit der Funktion HasUnoInterfaces teste ich, ob es ein UNO-Service ist. Zuerst einmal prüfe ich, ob der TypeName des Objekts den Text „Object“ hat. Das sagt mir, ob das Objekt auch wirklich ein Objekt ist.

```
If InStr(TypeName(vObj), "Object") < 1 Then
```

Wenn das Objekt kein Objekt ist, das ein UNO-Interface unterstützt, wird es inspiziert und Informationen darüber werden ausgegeben. Listing 240 zeigt, wie der Text zustande kommt. Bild 79 zeigt das Ergebnis.

Listing 240. *Typinformationen als String.*

```
Sub Run_My_Sample_Dialog_simple()
  Dim s(3, -3 To 3, 5) As String
  DisplayObjectInformation(s())
End Sub

Function Dlg_GetObjTypeInfo(vObj) As String
  Dim s As String
  s = "TypeName = " & TypeName(vObj) & Chr$(10) &_
```

```
    "VarType = " & VarType(vObj) & Chr$(10)
  If IsNull(vObj) Then
    s = s & "IsNull = True"
  ElseIf IsEmpty(vObj) Then
    s = s & "IsEmpty = True"
  Else
    If IsObject(vObj) Then s = s & "IsObject = True" & Chr$(10)
    If IsUnoStruct(vObj) Then s = s & "IsUnoStruct = True" & Chr$(10)
    If IsDate(vObj) Then s = s & "IsDate = True" & Chr$(10)
    If IsNumeric(vObj) Then s = s & "IsNumeric = True" & Chr$(10)
    If IsArray(vObj) Then
      On Local Error Goto DebugBoundsError:
      Dim i%            'Dimensionenzähler
      Dim sTemp$        'Temporärer String
      s = s & "IsArray = True" & Chr$(10) & "Bereich = ("
      Do While (i >= 0)
        'Listet auch mehrdimensionale Arrays auf.
        i = i + 1
        sTemp = LBound(vObj, i) & " To " & UBound(vObj, i)
        If i > 1 Then s = s & ", "
        s = s & sTemp
      Loop
      DebugBoundsError:
      'Tritt auf, wenn auf eine nicht vorhandene Dimension zugegriffen wird.
      On Local Error Goto 0
      s = s & ")" & Chr$(10)
    End If
  End If
  Dlg_GetObjTypeInfo = s
End Function
```

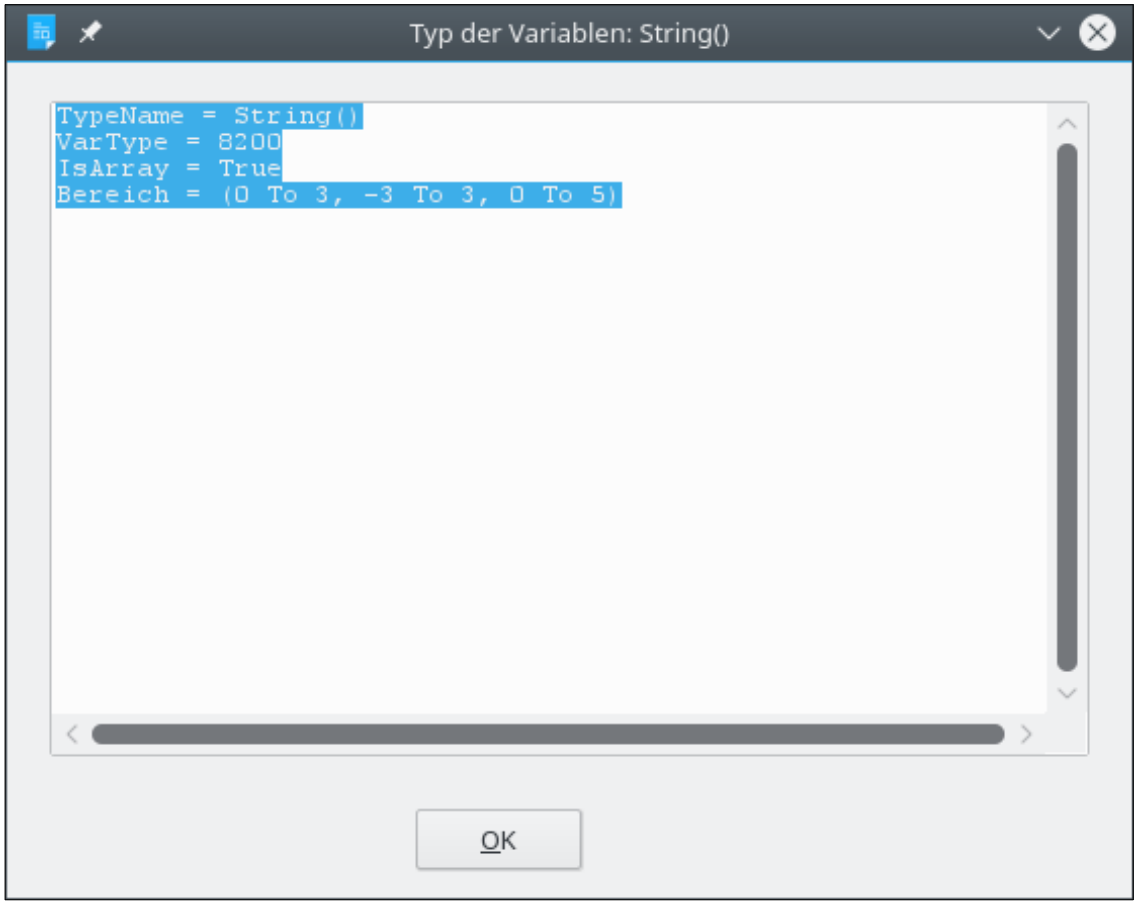


Bild 79. *Die Variable ist ein String-Array.*

Wenn das erste Argument ein UNO-Interface unterstützt, werden mit Hilfe der „dbg_“-Eigenschaften alle Methoden, Eigenschaften und Services angezeigt, die von diesem Objekt unterstützt werden, s. Listing 241 und Bild 80. Das ist ganz ähnlich dem Code im Listing 221, außer dass ein String zurückgegeben wird anstelle einer Reihe einfacher Dialoge. Dies ist ein exzellenter Weg, auf die Schnelle zu sehen, welche Methoden, Eigenschaften und Interfaces ein Objekt unterstützt. Beachten Sie auch, dass in der Titelzeile des Dialogs der Typ der Variablen angezeigt wird.

Listing 241. *Konvertiert den Debug-String in einen String mit Zeilenumbruchzeichen.*

```
Sub Run_My_Sample_Dialog()
  DisplayObjectInformation(ThisComponent)
End Sub

Function Dlg_DisplayDbgInfoStr(sInfo$, sSep$) As String
  Dim aInfo()                   'Array für die einzelnen Strings
  Dim i As Integer              'Indexvariable
  Dim j As Integer              'Integer-Variable für temporäre Werte
  Dim s As String               'Der noch nicht fertige Anteil
  s = sInfo$
  j = InStr(s, ":")                'Erster Doppelpunkt
  If j > 0 Then Mid(s, 1, j, "")  'Entfernt den Teil bis zum ersten Doppelpunkt.
  aInfo() = Split(s, sSep$)                    'Splittet den String am Trenner.
  For i = LBound(aInfo()) To Ubound(aInfo())   'Jeder Teil wird überprüft zum Entfernen
    aInfo(i) = Trim(aInfo(i))                  'der Leerzeichen am Anfang und am Ende.
    j = InStr(aInfo(i), Chr$(10))              'Manche haben einen zusätzlichen
    If j > 0  Then Mid(aInfo(i), j, 1, "")     'Zeilenumbruch, der entfernt werden muss.
  Next
```

```
    Dlg_DisplayDbgInfoStr = Join(aInfo(), Chr$(10))
End Function
```

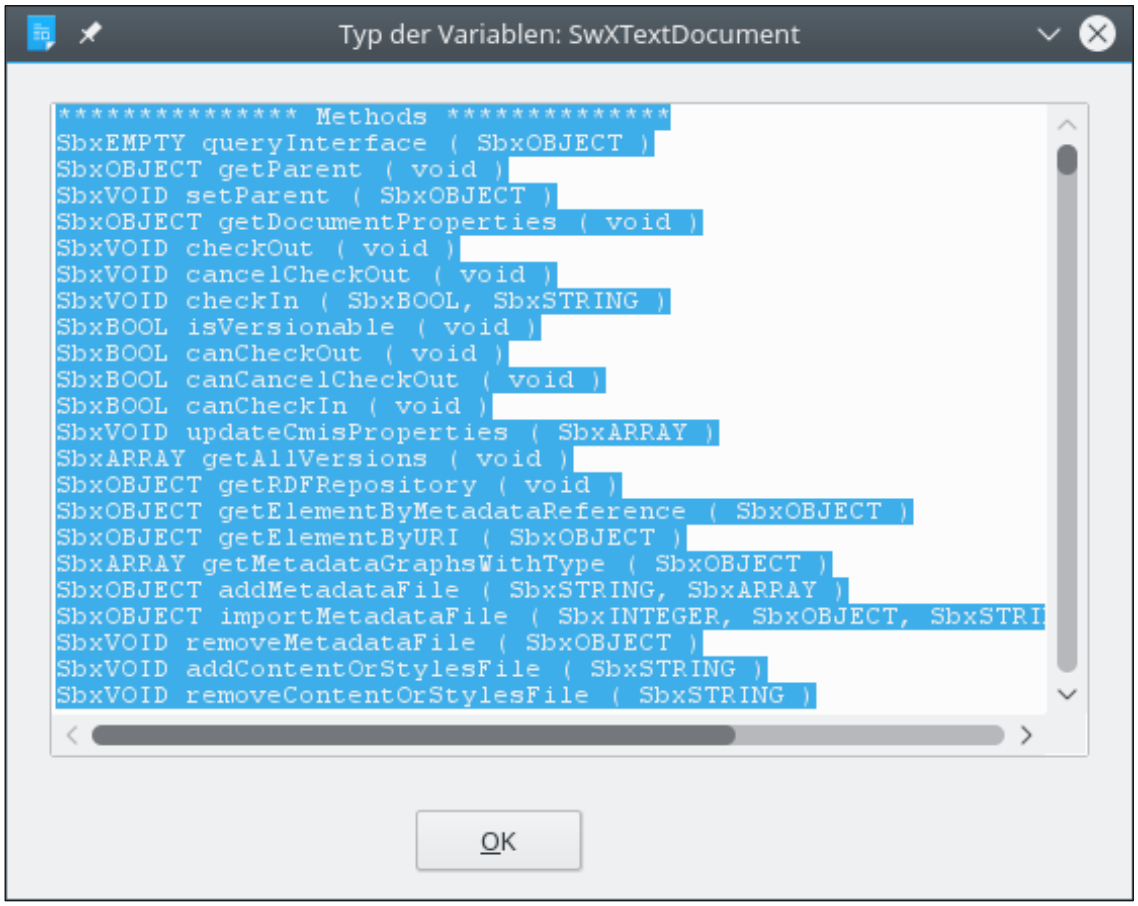


Bild 80. *Die Variable ist ein Textdokument.*

10.12. Services für Dateien und Verzeichnisse

Manche der Basic-Methoden zur Dateibearbeitung sind buggy und unzuverlässig. Sie möchten vielleicht lieber den Einsatz einiger eingebauter OOo-Services erwägen.

10.12.1. Pfadangaben

Die meisten Makros in diesem Kapitel nutzen die Funktion CurDir, um das Verzeichnis zum Speichern der Dateien zu bestimmen. Für AOO bietet der Service PathSettings (com.sun.star.util.PathSettings) Lese- und Schreibzugriff (und die Möglichkeit, einen Listener anzumelden) auf die Pfad-Eigenschaften der Anwendung. Für LO wurde der Service jedoch mit der Version 4.3 als veraltet erklärt. Sie sollten nun das Singleton „thePathSettings“ verwenden. Für LO ist der veraltete Service PathSettings nur die Instanz des Singletons „thePathSettings“.

Den Service erzeugen Sie mit AOO folgendermaßen:

```
oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")
```

Mit LO benötigen Sie für die Kopie eines Singletons seinen Namen, mit dem Sie es vom Standardkontext (DefaultContext) aufrufen:

```
oContext = GetProcessServiceManager().DefaultContext
oPathSettings = _
            oContext.getValueByName("/singletons/com.sun.star.util.thePathSettings")
```

Beachten Sie, dass dem Namen der Text „/singletons/“ vorausgeht.

Obwohl die Dokumentation in diesem Punkt nicht ganz deutlich ist, deuten meine Beispiele darauf

hin, dass der Pfad als URL zurückgegeben wird. Andererseits nutzt der Service PathSettings den Service PathSubstitution, in dem ganz konkret festgelegt ist, dass URLs geliefert werden. Die Funktion OOMEWorkDir im Listing 242 zeigt, wie das Arbeitsverzeichnis ermittelt wird.

Listing 242. *Ermittlung des Arbeitsverzeichnisses.*

```
Sub PrintOOMEWorkDir
  Print OOMEWorkDir
End Sub

Function OOMEWorkDir() As String
  Dim s$
  Dim  oPathSettings      'OOo-Pfadeigenschaften
  Dim oContext            'Standardkontext

  oContext = GetProcessServiceManager().DefaultContext
  oPathSettings = _
              oContext.getValueByName("/singletons/com.sun.star.util.thePathSettings")
  If IsEmpty(oPathSettings) Then
    'AOO nutzt den älteren Singleton-Typ:
    oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")
  End If

  s = oPathSettings.Work  'Arbeitsverzeichnis
  If s = "" Then
    s = GetPathSeparator()
  'Test, ob die Rückgabe im URL-Format erfolgte.
  ElseIf Right(s, 1) <> "/" And Right(s, 1) <> "\\" Then
    If Left(s, 5) = "file:" Then
      s = s & "/"
    Else
      s = s & GetPathSeparator()
    End If
  End If
  OOMEWorkDir() = s & "OOMEWork" & GetPathSeparator()
End Function
```

Ein Makro, das temporäre Dateien oder Verzeichnisse für Beispiele erzeugt, wird Listing 243 zum Erstellen und Löschen des Arbeitsverzeichnisses aufrufen.

Listing 243. *Erstellt und löscht das Arbeitsverzeichnis OOMEWork*

```
Function CreateOOMEWorkDir() As Boolean
  CreateOOMEWorkDir() = False
  Dim s$
  s = OOMEWorkDir()
  If Not FileExists(s) Then
    MkDir s
  End If
  CreateOOMEWorkDir() = FileExists(s)
End Function

Function RmOOMEWorkDir() As Boolean
  RmOOMEWorkDir() = False
  Dim s$
  s = OOMEWorkDir()
  If FileExists(s) Then
    RmDir s
  End If
```

```
    RmOOMEWorkDir() = Not FileExists(s)
End Function
```

Die Dokumentation listet unterstützte Eigenschaften auf. Aber nach genauer Überprüfung des Objekts habe ich noch mehr als nur die dokumentierten Eigenschaften gefunden.

Tabelle 96. *Dokumentierte PathSettings-Eigenschaften.*

Eigenschaft	Anzahl	Welches Verzeichnis
Addin	Einfach	Enthält Tabellen-Addins, die das alte Addin-API nutzen.
AutoCorrect	Mehrfach	Enthält die Einstellungen für den AutoKorrektur-Dialog.
AutoText	Mehrfach	Enthält die AutoText-Module.
Backup	Einfach	Wo die automatischen Dokumentsicherungen gespeichert sind.
Basic	Mehrfach	Enthält die Basic-Dateien für die Autopiloten.
Bitmap	Einfach	Enthält die externen Symbole für die Symbolleisten.
Classification	Einfach	Nicht dokumentiert.
Config	Einfach	Enthält Konfigurationsdateien. Diese Eigenschaft ist nicht im Optionendialog Pfade enthalten und kann nicht geändert werden.
Dictionary	Einfach	Enthält die OOo-Wörterbücher.
Favorite	Einfach	Enthält die gespeicherten Verzeichnis-Bookmarks.
Filter	Einfach	Wo Filter gespeichert sind.
Fingerprint	Einfach	Nicht dokumentiert.
Gallery	Mehrfach	Enthält die Gallery-Datenbank und Multimediadateien.
Graphic	Einfach	Erscheint mit dem Dialog zum Öffnen einer Grafik oder zum Speichern einer neuen Grafik.
Help	Einfach	Enthält die OOo-Hilfetexte.
Iconset	Einfach	Nicht dokumentiert.
Linguistic	Einfach	Enthält die OOo-Rechtschreibungsdateien.
Module	Einfach	Enthält die OOo-Module.
Numbertext	Einfach	Nicht dokumentiert.
Palette	Einfach	Enthält die Farbpalettendateien mit den benutzerdefinierten Farben und Mustern (*.SOB und *.SOF).
Plugin	Mehrfach	Enthält die Plugins.
Storage	Einfach	Wo Mail- und News-Dateien sowie andere Informationen (zum Beispiel über FTP-Server) gespeichert sind. Diese Eigenschaft ist nicht im Optionendialog Pfade enthalten und kann nicht geändert werden.
Temp	Einfach	Enthält die OOo-Temp-Dateien.
Template	Mehrfach	Enthält die OOo-Dokumentvorlagen.
UIConfig	Mehrfach	Globale Verzeichnisse für die Konfigurationsdateien der Benutzerschnittstelle. Die Konfiguration der Benutzerschnittstelle ist verschmolzen mit den Benutzereinstellungen im UserConfig-Verzeichnis.
UserConfig	Einfach	Enthält die Benutzereinstellungen, einschließlich der Konfigurationsdateien der Benutzerschnittstelle für Menüs, Symbolleisten, Tastenkürzel und Statusleisten.
UserDictionary	Einfach	Enthält die Benutzerwörterbücher. Veraltet.
Work	Einfach	Das Arbeitsverzeichnis. Diese Eigenschaft ist im Optionendialog Pfade enthalten und kann vom Benutzer geändert werden.

Das Makro im Listing 244 listet Ihnen die Pfadeinstellungen auf Ihrem Rechner auf. Auf meinem Rechner finde ich zahlreiche weitere Pfade wie Work_internal, Work_user oder Work_writable. Das

Makro demonstriert eine Reihe von fortgeschrittenen Techniken, die in diesem Kapitel nicht weiter erläutert werden.

- Wie ein neues Dokument erzeugt wird.
- Wie Text in ein Dokument eingefügt wird.
- Wie eine Absatzvorlage gesetzt wird.
- Wie Absatzwechsel in ein Textobjekt eingefügt werden.

Das folgende Makro wurde ursprünglich von Danny Brewer geschrieben, der viel geleistet hat, um das Wissen über OOo-Makros zu erweitern, bevor er sich anderen Dingen zuwandte. Ich habe das Makro so weit modifiziert, dass es alle Variablen deklariert und die zurückgegebenen Eigenschaftstypen als Array-Werte behandelt.

Listing 244. *Gibt die PathSettings in einem neuen Textdokument aus.*

```
Sub DisplayPathSettings
  Dim oPathSettings    ' Der Service PathSettings.
  Dim oPropertySetInfo ' Zugriff auf die Service-Eigenschaften.
  Dim aProperties      ' Enthält alle Service-Eigenschaften.
  Dim oDoc             ' Referenz auf ein neu erstelltes Dokument.
  Dim oText            ' Das Text-Objekt des Dokuments.
  Dim oCursor          ' Cursor im Text-Objekt.
  Dim oProperty        ' Eine Service-Eigenschaft.
  Dim cPropertyName$   ' Der Name der Eigenschaft.
  Dim cPropertyValue   ' Der Wert der Eigenschaft kann ein Array sein oder eine Reihe
                       ' von Strings.
  Dim aPaths           ' Die Pfade als Array.
  Dim cPath$           ' Ein einzelner Pfad aus dem Array.
  Dim j As Integer     ' Indexvariable.
  Dim i As Integer     ' Indexvariable.

  Dim oContext
  oContext = GetProcessServiceManager().DefaultContext
  oPathSettings = _
            oContext.getValueByName("/singletons/com.sun.star.util.thePathSettings")
  If IsEmpty(oPathSettings) Then
    'AOO nutzt den älteren Singleton-Typ:
    oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")
  End If

  ' Beispiel, wie man die gewünschte einzelne Eigenschaft ermittelt.
  'oPathSettings.Work

  ' Holt die Information über die Eigenschaften der Pfadeinstellungen.
  oPropertySetInfo = oPathSettings.getPropertySetInfo()

  ' Bildet ein Array der Eigenschaften.
  aProperties = oPropertySetInfo.getProperties()

  ' Erzeugt ein Dokument zur Datenausgabe.
  oDoc = StarDesktop.loadComponentFromURL("private:factory/swriter", _
                                                    "_blank", 0, Array())
  oText = oDoc.getText()
  oCursor = oText.createTextCursor()

  oText.insertString(oCursor, "Pfadeinstellungen", False)
```

```
    oCursor.ParaStyleName = "Überschrift 1"
    oText.insertControlCharacter(oCursor, _
              com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)

    ' Iteriert durch das Eigenschaften-Array
    '  und schreibt die Informationen über jede Eigenschaft in die Ausgabedatei.
    For i = LBound(aProperties) To UBound(aProperties)
      oProperty = aProperties(i)
      cPropertyName = oProperty.Name
      cPropertyValue = oPathSettings.getPropertyValue(cPropertyName)

      oText.insertString(oCursor, cPropertyName, False)
      oCursor.ParaStyleName = "Überschrift 3"
      oText.insertControlCharacter(oCursor, _
                com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
      If IsArray(cPropertyValue) Then
        ' Manchmal werden mehrere URLs als Array zurückgegeben.
        aPaths = cPropertyValue
      ElseIf Len(cPropertyValue) > 0 Then
        ' Manchmal werden mehrere URLs durch Semikolon getrennt.
        ' Sie werden in ein String-Array gesplittet.
        aPaths = Split(cPropertyValue, ";")
      Else
        aPaths = Array()
      End If
      For j = LBound(aPaths) To UBound(aPaths)
        cPath = aPaths(j)
        oText.insertString(oCursor, cPath, False)
        oText.insertControlCharacter(oCursor, _
           com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
      Next

      oText.insertControlCharacter(oCursor, _
          com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
    Next i
End Sub
```

Es ist ganz einfach, einen Pfad zu setzen, entweder durch direkte Zuweisung oder über das Objekt PropertySetInfo. OOo verlässt sich darauf, dass diese Werte korrekt sind. Wenn Sie also fehlerhafte Pfade angeben, wird sich das negativ auf die Arbeit von OOo auswirken.

```
oPathSettings.Work = ConvertToUrl("C:\MyWorkDir")
oPathSettings.setPropertyValue("Work", "C:\MyWorkDir")
```

10.12.2. Ersetzung von Pfadvariablen

OOo verwaltet intern eine Reihe von Variablen für Pfadfestlegungen, deren Namen als String die Form "$(name)" haben. Der Service com.sun.star.util.PathSubstitution bietet Methoden, diese Variablennamen in Strings durch ihre Werte, also durch die echten Pfade zu ersetzen. Das funktioniert in beiden Richtungen. Reale Pfade können in Strings durch die entsprechenden Variablennamen rückersetzt werden.

Tabelle 97. *Variablen für OOo-Pfade.*

Name	Beschreibung
$(inst)	Installationspfad der OOo-Basisebene.
$(prog)	Programmpfad der OOo-Basisebene.
$(brandbaseurl)	Installationspfad der OOo-Marken-Ebene.

Name	Beschreibung
$(user)	Benutzer-Installationsverzeichnis.
$(work)	Benutzer-Arbeitsverzeichnis. „Eigene Dateien“ für Windows, das home-Verzeichnis des Benutzers für Linux.
$(home)	Das home-Verzeichnis des Benutzers. „Dokumente und Einstellungen“ für Windows, das home-Verzeichnis des Benutzers für Linux.
$(temp)	Das aktuelle Temp-Verzeichnis.
$(path)	Der Wert der Umgebungsvariable PATH.
$(username)	Der Login-Name des gerade aktiven Nutzers, nicht der Windows-Domain-Name. Seit LibreOffice 5.2
$(lang)	Der Ländercode innerhalb OOo: 01=englisch und 49=deutsch.
$(langid)	Der Sprachencode innerhalb OOo: 1033=englisch (USA), 1031=deutsch (Deutschland).
$(vlang)	Die Sprache, wie sie OOo als String verwendet, wie "de" in einem deutschen OOo.

Mit der Methode getSubstituteVariableValue wird jeweils ein Name durch seinen Wert ersetzt. Ist der Name nicht bekannt, resultiert ein Laufzeitfehler, s. Listing 245 und Bild 81.

Listing 245. *Ersetzung einer Pfadvariablen mit PathSubstitution.*

```
Sub UsePathSubstitution
  Dim oPathSub      ' Der Service PathSubstitution.
  Dim names         ' Liste der zu ersetzenden Namen.
  Dim subName$      ' Ein einzelner Name.
  Dim i As Integer  ' Indexvariable.
  Dim s$            ' Ausgabestring.

  'getSubstituteVariableValue liefert den Wert einer einzelnen Variablen:
  'Laufzeitfehler, wenn der Name der Variablen nicht bekannt ist.
  On Local Error Goto Oops

  names = Array("inst", "prog", "brandbaseurl", "user", "username", "work", "home", _
                "temp", "path", "langid", "vlang")

  oPathSub = CreateUnoService("com.sun.star.util.PathSubstitution")

  For i = LBound(names) To UBound(names)
    subName = "$(" & names(i) & ")"
    s = s & names(i) & " = "
    s = s & oPathSub.getSubstituteVariableValue(subName) & Chr(10)
  Next
  MsgBox s, 0, "Unterstützte Namen"
  Exit Sub

  Oops:
  s = s & "Variable nicht bekannt" & Chr(10)
  Resume Next
End Sub
```

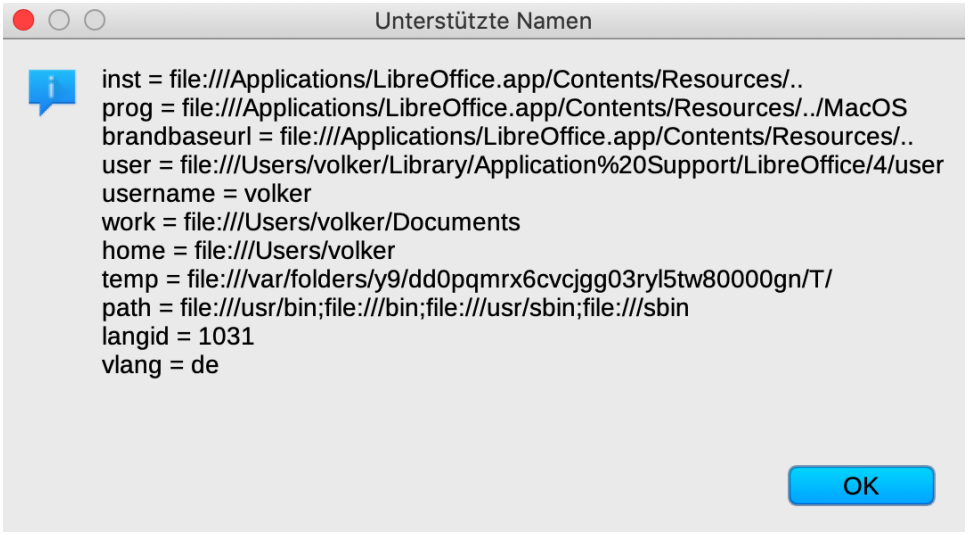


Bild 81. *Der Service PathSubstitution.*

Die Methoden substituteVariables und reSubstituteVariables ersetzen mehrere Werte in beliebigen Strings gleichzeitig. Die Strings werden mit substituteVariables nach bekannten Variablennamen gescannt, die dann im String durch die realen Pfade ersetzt werden. Mit reSubstituteVariables werden die Strings nach bekannten Pfaden gescannt und durch ihre Variablennamen ersetzt. Die Pfadangaben müssen absolut sein und in URL-Form vorliegen, s. Listing 246 und Bild 82.

Leider funktioniert die Methode substituteVariables nicht fehlerfrei. In LO (5.3) werden keine Namen im String erkannt, in AOO (4.1.4) wenigstens der erste Name, wenn er am Stringanfang steht. Die Namen werden jedoch erkannt, wenn nichts anderes im String steht. Dann aber gibt es keinen Unterschied zur Methode getSubstituteVariableValue.

Listing 246. *Ersetzung und Rückersetzung von Pfadvariablen in Strings.*

```
Sub UsePathReSubstitution()
  Dim oPathSub      'Der Service PathSubstitution.
  Dim s$            'Ausgabestring.
  Dim sTemp$        'Von der Methode zurückgegebener String

  oPathSub = CreateUnoService("com.sun.star.util.PathSubstitution")

  'Zwei Variablen sollen durch die realen Pfade ersetzt werden.
  'Das Argument False bedeutet, dass kein Fehler erzeugt wird,
  'wenn eine unbekannte Variable verwendet wird.
  s = "$(temp)/OOME/ oder $(work)"
  sTemp = oPathSub.substituteVariables(s, False)
  s = "String mit Variablennamen:" & Chr(10) & s & Chr(10) & _
      "Ersetzung:" & Chr(10) & sTemp & Chr(10) & Chr(10)

  'In dieser Richtung wird der gesamte String als einzelner Pfad betrachtet.
  'Das heißt, dass Leerzeichen in URL-Notation kodiert werden.
  sTemp = oPathSub.substituteVariables("$(temp)", False) & "/OOME/ oder " & _
          oPathSub.substituteVariables("$(work)", False)
  s = s & "String mit Pfadangaben:" & Chr(10) & sTemp & Chr(10) &_
      "Rückersetzung:" & Chr(10) & _
      oPathSub.reSubstituteVariables(sTemp) & Chr$(10)
  MsgBox s
End Sub
```

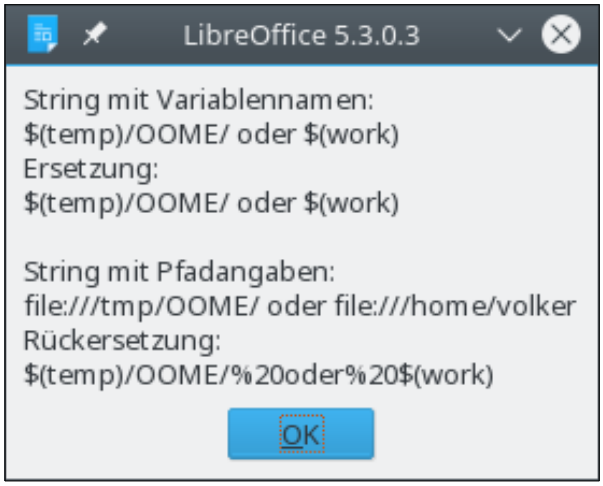


Bild 82. *Pfad-Ersetzung und -Rückersetzung.*

Achtung Die Methode substituteVariables funktioniert nicht. In LO (5.3) werden keine Namen im String erkannt, in AOO (4.1.4) nur der erste Name, und auch nur, wenn er am Stringanfang steht. Ein Name wird jedoch erkannt, wenn nichts anderes im String steht. Dann aber gibt es keinen Unterschied zur Methode getSubstituteVariableValue. Die Methode reSubstituteVariables funktioniert allerdings fehlerfrei.

10.12.3. Der einfache Dateizugriff SimpleFileAccess

Wenn OOo intern auf Dateien zugreift, so geschieht das mit anderen Routinen als mit denen von Basic. Einer der Services ist com.sun.star.ucb.SimpleFileAccess mit folgenden Methoden:

Tabelle 98. *Methoden im Service com.sun.star.ucb.SimpleFileAccess.*

Methode	Beschreibung
copy(vonURL, zuURL)	Kopiert eine Datei.
move(vonURL, zuURL)	Verschiebt eine Datei.
kill(URL)	Löscht eine Datei oder ein Verzeichnis, auch wenn das Verzeichnis nicht leer ist.
isFolder(URL)	Gibt True zurück, wenn es der URL eines Verzeichnisses ist.
isReadOnly(URL)	Gibt True zurück, wenn die Datei schreibgeschützt ist.
setReadOnly(URL, boole)	Setzt den Dateistatus auf schreibgeschützt, wenn das boolesche Argument True ist, ansonsten wird der Schreibschutz aufgehoben.
createFolder(URL)	Legt ein neues Verzeichnis an.
getSize(URL)	Gibt die Dateigröße als Long zurück.
getContentType(URL)	Gibt den Inhaltstyp einer Datei als String zurück. Auf meinem Rechner hat eine odt-Datei den Typ application/vnd.sun.staroffice.fsys-file.
getDateTimeModified(URL)	Gibt das Datum der letzten Änderung der Datei zurück, als Struct com.sun.star.util.DateTime, mit den Eigenschaften Year, Month, Day, Hours, Minutes, Seconds und Sekundenbruchteilen, die in AOO HundredthSeconds (Hundertstel Sekunden) heißen, in LO NanoSeconds.
getFolderContents(URL, boole)	Gibt den Inhalt eines Verzeichnisses als String-Array zurück. Jeder String ist ein absoluter Pfad als URL. Wenn boole True ist, werden Dateien und Verzeichnisse gelistet. Wenn boole False ist, werden nur Dateien zurückgegeben.
exists(URL)	Gibt True zurück, wenn die Datei oder das Verzeichnis existiert.
openFileRead(URL)	Öffnet eine Datei zum Lesen, gibt einen Eingabestream zurück.
openFileWrite(URL)	Öffnet eine Datei zum Schreiben, gibt einen Ausgabestream zurück.

Methode	Beschreibung
openFileReadWrite(URL)	Öffnet eine Datei zum Lesen und Schreiben, gibt einen Stream zurück.
setInteractionHandler(Handler)	Setzt einen interaktiven Handler für weitere Operationen. Dies ist ein fortgeschritteneres Thema und wird hier nicht behandelt.
writeFile(zuURL, inputStream)	Überschreibt den Dateiinhalt mit den angegebenen Daten.

10.12.4. Streams, Pipes und Sockets

Streams

Ein Stream ist ein Datenstrom zum Lesen oder Schreiben von Daten aus einer Eingabequelle in eine Ausgabequelle, die über das Dateiensystem hinausgehen kann. Beispielsweise transferiere ich Dateien über Streams zwischen dem normalen Dateiensystem und einem Datenbankfeld. Dieser Abschnitt behandelt längst nicht das ganze Potential von Streams und berührt nicht einmal solche Dinge wie Markable Streams und Object Streams. Mit Streams kann man allerhand Sachen machen, wie die Einrichtung eines Beobachters (Listener), der automatisch aufgerufen wird, wenn ein bestimmtes Ereignis eintritt. Wissbegierige Leser mögen sich das Dokument über Streams anschauen:

- http://udk.openoffice.org/common/man/concept/streams.html
- http://api.openoffice.org/docs/common/ref/com/sun/star/io/module-ix.html
- https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1io.html

Achtung Die Methode readLine() entfernt nicht das Zeilenende-Zeichen, wenn das Dateiende erreicht ist.

Tabelle 99. *Stream-Methoden.*

Methode	Stream	Beschreibung
available()	InputStream	Gibt die Anzahl der verfügbaren Bytes als Long zurück.
closeInput()	InputStream	Schließt den Eingabestream.
closeOutput()	OutputStream	Schließt den Ausgabestream.
flush()	OutputStream	Leert den Puffer.
getLength()	XSeekable	Gibt die Länge des Streams zurück.
getPosition()	XSeekable	Gibt den Stream-Offset als 64-Bit-Integer zurück.
isEOF()	TextInputStream	Gibt True zurück, wenn das Dateiende erreicht ist.
readBoolean()	DataInputStream	Liest einen 8-Bit-Wert und gibt ein Byte zurück. 0 bedeutet False, alle anderen Werte bedeuten True.
readByte()	DataInputStream	Liest einen 8-Bit-Wert und gibt ein Byte zurück.
readBytes(ByteArray, Long)	InputStream	Liest die angegebene Anzahl Bytes und gibt die Zahl der gelesenen Bytes zurück. Ist die angeforderte Anzahl nicht gleich der gelesenen, ist das Dateiende erreicht.
readChar()	DataInputStream	Liest einen 16-Bit-Unicode-Wert und gibt ihn zurück.
readDouble()	DataInputStream	Liest einen 64-Bit-IEEE-Double und gibt ihn zurück.
readFloat()	DataInputStream	Liest einen 32-Bit-IEEE-Float und gibt ihn zurück.
readHyper()	DataInputStream	Liest einen 64-Bit-Big-Endian-Integer und gibt ihn zurück.
readLine()	TextInputStream	Liest Text bis zum Zeilenende (CR, LF, oder CR/LF) oder bis EOF und gibt ihn als String zurück (ohne CR, LF).
readLong()	DataInputStream	Liest einen 32-Bit-Big-Endian-Integer und gibt ihn zurück.
readShort()	DataInputStream	Liest einen 16-Bit-Big-Endian-Integer und gibt ihn zurück.

Methode	Stream	Beschreibung
readSomeBytes(ByteArray, Long)	InputStream	Liest die angegebene Anzahl Bytes und gibt die Zahl der gelesenen Bytes zurück. Ist die gelesene Anzahl gleich 0, ist das Dateiende erreicht.
readString(CharArray, Boolean)	TextInputStream	Liest Text, bis eins der angegebenen Trennzeichen oder EOF erreicht ist und gibt ihn als String zurück. Das boolesche Argument bestimmt, ob das Trennzeichen zurückgegeben (False) oder entfernt wird (True).
readUTF()	DataInputStream	Liest einen Unicode-Zeichen-String und gibt ihn zurück.
seek (INT64)	XSeekable	Verschiebt den Stream-Zeiger auf die angegebene Position.
setEncoding(String)	TextInputStream	Setzt den Zeichensatz (s. http://www.iana.org/assignments/character-sets).
skipBytes(Long)	InputStream	Überspringt die angegebene Anzahl Bytes.
truncate()	XTruncate	Setzt die Größe der Datenquelle auf null.
writeBoolean(Boolean)	DataOutputStream	Schreibt einen booleschen Wert als 8-Bit-Wert. 0 bedeutet False, alle anderen Werte bedeuten True.
writeByte(Byte)	DataOutputStream	Schreibt einen 8-Bit-Wert und gibt ein Byte zurück.
writeBytes(ByteArray())	OutputStream	Schreibt alle Bytes in den Stream.
writeChar(Char)	DataOutputStream	Schreibt ein 16-Bit-Unicode-Zeichen.
writeDouble(Double)	DataOutputStream	Schreibt einen 64-Bit-IEEE-Double.
writeFloat(Float)	DataOutputStream	Schreibt einen 32-Bit-IEEE-Float.
writeHyper(INT64)	DataOutputStream	Schreibt einen 64-Bit-Big-Endian-Integer.
writeLong(Long)	DataOutputStream	Schreibt einen 32-Bit-Big-Endian-Integer.
writeShort(Short)	DataOutputStream	Schreibt einen 16-Bit-Big-Endian-Integer.

Es gibt eine ganze Reihe verschiedener Typen von Stream-Services und -Schnittstellen (s. Tabelle 99). Ein einfacher Stream, wie er von SimpleFileAccess zurückgegeben wird, unterstützt nur das Lesen und Schreiben von Rohdaten. Man muss die Daten in ein Byte-Array konvertieren. Gewöhnlich erstellt man aber gezielt einen passenden Stream wie DataOutputStream oder TextInputStream und nutzt ihn als Hülle für den einfachen Stream des Service SimpleFileAccess. Das Listing 246 funktioniert unter Apples macOS nur, wenn OOo aus einem Terminal heraus gestartet wurde.

Listing 247. *Mit SimpleFileAccess Textdateien lesen und schreiben.*

```
Sub ExampleSimpleFileAccess
  Dim oSFA        ' Der Service SimpleFileAccess.
  Dim sFileName$  ' Name der zu öffnenden Datei.
  Dim oStream     ' Der von SimpleFileAccess zurückgegebene Stream.
  Dim oTextStream ' Der Service TextStream.
  Dim sStrings    ' Strings für den Lese-/Schreibtest.
  Dim sInput$     ' Der gelesene String.
  Dim s$          ' Ausgabestring.
  Dim i%          ' Indexvariable.
  If IsRootDir Then Exit Sub

  sStrings = Array("Eins", "UTF:Ää", "1@3")

  ' Die Testdatei.
  sFileName = CurDir() & "/WegMitMir.out"

  ' Erzeugt den Service SimpleFileAccess.
```

```
  oSFA = CreateUnoService("com.sun.star.ucb.SimpleFileAccess")

  ' Erzeugt den Schreibstream.
  oTextStream = CreateUnoService("com.sun.star.io.TextOutputStream")

  'Falls die Datei schon existiert, wird sie gelöscht.
  If oSFA.exists(sFileName) Then
    oSFA.kill(sFileName)
  End If

  ' Die Datei wird zum Schreiben geöffnet.
  oStream = oSFA.openFileWrite(sFileName)

  ' Verknüpft den einfachen Stream mit dem Textstream.
  ' Der Textstream wird den einfachen Stream nutzen.
  oTextStream.setOutputStream(oStream)

  ' Schreibt die Strings.
  For i = LBound(sStrings) To UBound(sStrings)
    oTextStream.writeString(sStrings(i) & Chr$(10))
  Next

  ' Schließt den Stream.
  oTextStream.closeOutput()

  ' Erzeugt den Lesestream.
  oTextStream = CreateUnoService("com.sun.star.io.TextInputStream")
  oStream = oSFA.openFileRead(sFileName)
  oTextStream.setInputStream(oStream)
  For i = LBound(sStrings) To UBound(sStrings)
    sInput = oTextStream.readLine()
    s = s & CStr(i)

    ' Wenn EOF erreicht ist, wird der Zeilentrenner nicht entfernt.
    ' Ich halte das für einen Bug.
    If oTextStream.isEOF() Then
      If Right(sInput, 1) = Chr$(10) Then
        sInput = Left(sInput, Len(sInput) - 1)
      End If
    End If

    ' Prüft, ob der gelesene String identisch mit dem geschriebenen ist.
    If sInput <> sStrings(i) Then
      s = s & " : MIST "
    Else
      s = s & " : OK "
    End If
    s = s & "(" & sStrings(i) & ")"
    s = s & "(" & sInput & ")" & Chr$(10)
  Next
  oTextStream.closeInput()
  MsgBox s
  oSFA.kill(sFileName)
End Sub
```

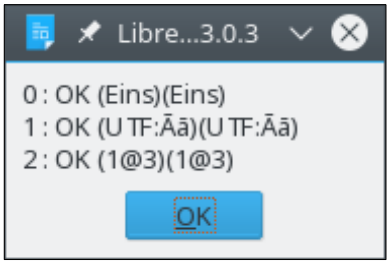


Bild 83. *Textdatei mit SimpleFileAccess.*

Pipes

Eine Pipe ist ein Ausgabe- und Eingabestream. In den Ausgabestream geschriebene Daten werden gepuffert, bis sie vom Eingabestream gelesen sind. Der Service Pipe verwandelt einen Ausgabestream in einen Eingabestream, auf Kosten eines zusätzlichen Puffers. Es ist einfach, eine Pipe zu erzeugen und zu schließen. Obwohl CreatePipe im Listing 248 Datenstreams erzeugt, würde eine ganz einfache Änderung stattdessen einen Textstream erzeugen.

Listing 248. *Eine Pipe erzeugen und schließen.*

```
Function CreatePipe()
  Dim oPipe    ' Der Service Pipe.
  Dim oDataInp ' Der Service DataInputStream.
  Dim oDataOut ' Der Service DataOutputStream.

  oPipe    = createUNOService ("com.sun.star.io.Pipe")
  oDataInp = createUNOService ("com.sun.star.io.DataInputStream")
  oDataOut = createUNOService ("com.sun.star.io.DataOutputStream")
  oDataInp.setInputStream(oPipe)
  oDataOut.setOutputStream(oPipe)
  CreatePipe = oPipe
End Function

Sub ClosePipe(oPipe)
  oPipe.Successor.closeInput
  oPipe.Predecessor.closeOutput
  oPipe.closeInput
  oPipe.closeOutput
End Sub
```

TestPipes (Listing 249) nutzt eine Pipe, um ein Byte-Array in einen Double und einen Double in ein Byte-Array zu konvertieren.

Listing 249. *Konvertiert ein Byte-Array zu Double und Double zu einem Byte-Array.*

```
Sub TestPipes
  Dim oPipe        ' Der Service Pipe.
  Dim d As Double
  Dim i As Integer
  Dim s$

  oPipe = CreatePipe()

  ' Zuerst wird eine Bytefolge geschrieben, die die Zahl 3,1415 darstellt.
  oPipe.Predecessor.writeBytes(Array(64, 9, 33, -7, -16, 27, -122, 110))
  d = 2.6    '4004CCCCCCCCCCD
  oPipe.Predecessor.writeDouble(d)
```

```
    ' Nun wird die Pipe gelesen.
    d = oPipe.Successor.readDouble()
    s = "Das Byte-Array gelesen als: " & CStr(d) & Chr$(10) & Chr$(10)

    ' Nun wird der Double-Wert gelesen, der als Bytefolge geschrieben wurde.
    s = s & "2,6 = "
    Do While oPipe.Successor.available() > 0
      i = oPipe.Successor.readByte()
      REM Falls das Byte negativ ist
      i = i And 255
      If i < 16 Then s = s & "0"
      s = s & Hex(i) & " "
    Loop
    ClosePipe(oPipe)
    MsgBox s
End Sub
```

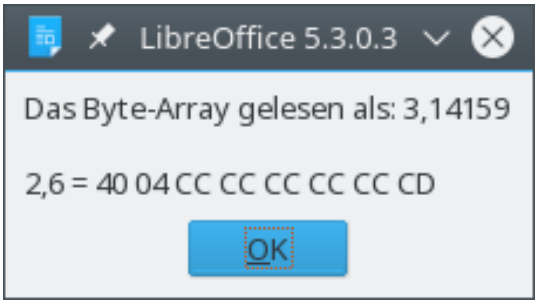


Bild 84. *Demonstration einer Pipe*

Sockets (A. Heier)

Bei Schnittstellen für Netzwerkverbindungen wird auch von Sockets gesprochen. Ein Socket ist vereinfacht ausgedrückt nichts anderes als ein vom Betriebssystem verwalteter Speicherbereich auf der Netzwerkkarte, der mit Daten beschrieben und gelesen werden kann. Auf diesen Speicherbereich greifen Netzwerkkarte und Anwendung gemeinsam zu. Dabei dient die IP-Adresse zusammen mit dem Port zur Identifikation des richtigen Speicherbereiches.

Um nun mit Socketverbindungen arbeiten zu können, stellt die API die Services Connector und Acceptor bereit, mit den entsprechenden Interfaces XConnector und XAcceptor. Die beiden haben jeweils eine Methode, die zum Verbindungsaufbau genutzt wird. XAcceptor besitzt noch eine weitere Methode, die zum Beenden des Service dient.

Tabelle 100. *Über die Services com.sun.star.connection.Connector und com.sun.star.connection.Acceptor verfügbare Methoden.*

Methode	Interface	Beschreibung
connect(String)	XConnector	Baut aktiv eine neue Verbindung zur Interprozesskommunikation mit dem übergebenen Parameter auf. Gibt nach erfolgreichem Verbindungsaufbau einen bidirektionalen Bytestream zurück (com.sun.star.connection.XConnection).
accept(String)	XAcceptor	Das Gegenstück zu connect(). Es wird passiv auf eine eingehende Verbindungsanfrage gewartet. Wird auf die gleiche Weise verwendet.
stopaccepting()	XAcceptor	Beendet das Warten auf eine eingehende Verbindung.
read(ByteArray(), Long)	XConnection	Die Methode read() befüllt das übergebene Integer-Array mit der im zweiten Parameter bestimmten Anzahl an Bytes. Die Methode blockiert die Anwendung, bis die Leseoperation beendet ist.
write(ByteArray())	XConnection	Schreibt die als Integer-Array übergebenen Daten in den Datenstrom. Die Methode blockiert die Anwendung, bis die Schreiboperation beendet ist.

Methode	Interface	Beschreibung
flush()	XConnection	Leert alle internen Puffer.
close()	XConnection	Beendet unverzüglich alle Lese- und Schreiboperationen. Schließt das Socket.
getDescription()	XConnection	Gibt Informationen über das Socket zurück.

Bitte beachten Sie, dass die beiden Methoden connect() und accept() „non-blocking“ sind. Erst die beiden Methoden read() und write() blockieren den Programmablauf.

Bevor nun mit der Einführung in die Socketprogrammierung begonnen wird, möchte ich Sie auf die folgenden Einschränkungen hinweisen:

- Es ist leider nicht möglich, über Verkettung einzelner IO-Ströme eine Socketverbindung herzustellen, um über Netzwerk zu kommunizieren. An ein Socket kann eine Pipe gebunden werden, jedoch müssen dann beide Prozesse auf demselben Rechner laufen. Das wird an dieser Stelle jedoch nicht weiter behandelt.
- Die Verkettung mit anderen Streams, wie einem DataInputStream oder DataOutputStream ist nicht möglich. Ein Hinweis für versierte Leser: Den Services Connector und Acceptor fehlt dazu das Interface com.sun.star.io.XConnectable.
- Weiter ist Voraussetzung, dass die Anzahl der Bytes für die Leseoperation vorher bekannt ist oder berechnet wird und exakt genau so viele Bytes empfangen werden. Sie könnten hier nun auf die Idee kommen und einen Workaround programmieren, in dem Sie immer nur 1 Byte lesen, bis das Ende des Datenstroms erreicht ist. Bedauerlicherweise wird das Ende des Datenstroms nicht erkannt. In Tests mit diesem Workaround hat read() stets eine 1 zurückgegeben. Das nochmalige Anwenden von read() nach dem Ende des Datenstroms führte dann dazu, dass auf ein nicht mehr vorhandenes Byte zum Auslesen gewartet wurde. Es konnte das Ende nicht zuverlässig detektiert werden, und die Anwendung fror ein. Es fehlen auch Methoden, die einen Timeout einer Verbindung überwachen.
- Möglicherweise ist in einer zukünftigen Basic-Version die Funktionalität des Interface XConnection2 über Makroprogrammierung erreichbar. Dieses Interface würde die Methoden readSomeBytes(Array(), Long) und available() zur Verfügung stellen, die die Netzwerkprogrammierung deutlich erleichtern würden.

Achtung Die Schreib- und Lesebefehle sind „blocking calls“ und blockieren die Anwendung, bis die entsprechende Anzahl an Bytes geschrieben, bzw. gelesen wurde.

Soweit zu den Randbedingungen. Damit ist der Rahmen für den Einsatz von Netzwerkverbindungen gegeben.

Damit Daten überhaupt über das Netzwerk versendet werden können, müssen Sie sie zwingend in ein bestimmtes Format bringen. Dazu werden alle Daten in ein Array aus „Bytes“ (also Integer in Basic) geschrieben und dann in den Datenstrom geschrieben. Dabei müssen die Wertgrenzen von -128 bis +127 eingehalten werden. Wie das funktioniert, wird nachfolgend erläutert.

In OOo hat ein Integer 2 Bytes und ein UNO-Byte eben 1. Ein Byte ist kein Standarddatentyp in Basic, aber in der API ist er enthalten. Zusätzlich kommt hinzu, dass ein Byte in der API den Wertebereich von -128 bis +127 hat, also signed ist. Damit können Werte wie z. B. 255 nicht direkt als Byte übergeben werden (bei der direkten Übergabe von 255 würde OOo einen Fehler melden, da die Wertgrenze bei Byte verletzt wurde). 255 dezimal entspricht &HFF, aber &HFF entspricht eben nicht -1 dezimal. Daher werden von Werten größer 127 nun 256 abgezogen, so kann man den Wertebereich anstatt von 0 bis 127 auf 0 bis 255 ausdehnen. Gerade beim Versenden und Empfangen von

Zeichenketten wird der ASCII-Code im Wertebereich von 0 … 255 benötigt. Um das zu verstehen, sehen Sie sich die bitweise Darstellung an:

```
Dim i As Integer
i = &HFF
```

Die interne Darstellung bei Integer sieht folgendermaßen aus:

```
Nibble:      4    3    2    1

hexadezimal: 0    0    F    F
binär:       0000 0000 1111 1111
```

Im ganz linken Bit (MSB = Most Significant Bit) wird das Vorzeichen bestimmt (0 = positive Zahl und 1 = negative Zahl). Daher auch die Grenzen:

```
Nibble:      4    3    2    1

hexadezimal: 7    F    F    F    = +32767
binär:       0111 1111 1111 1111 = +32767

hexadezimal: F    F    F    F    = -32768
binär:       1111 1111 1111 1111 = -32768
```

Wie kommt man aber jetzt nun bei einem Wertebereich von -128 bis +127 dazu, 255 als vorzeichenloses Byte darzustellen? Auch hier hilft Ihnen wieder die Betrachtung der internen Darstellung:

```
Nibble:      4    3    2    1

hexadezimal: 0    0    F    F    = +255
binär:       0000 0000 1111 1111 = +255

hexadezimal: 0    1    0    0    = +256
binär:       0000 0001 0000 0000 = +256
i = 255 - 256

hexadezimal: F    F    F    F    = -1
binär:       1111 1111 1111 1111 = -1
```

Wie ist das nun zu interpretieren? Bei der Konvertierung von signed nach unsigned Byte wird in Basic das Vorzeichenbit mit betrachtet.

Bei der Übergabe eines „Bytes“ (eigentlich Integer) an eine Datensenke (z. B. Socket), die ein signed Byte erwartet, wird in OOo intern das Vorzeichen korrekt interpretiert und OOo meldet keinen Überlauf des Wertebereichs als Fehler bei Werten größer 127.

Dazu werden intern die Nibble 3 und 4 weggelassen (= Typecasting) und nur Nibble 1 und 2 in den Strom geschrieben. Am anderen Ende der Verbindung kommen so nun korrekt aufbereitete Daten im Wertebereich von 0 … 255 an.

Eine mögliche Konvertierungsfunktion kann so aussehen:

Listing 250. *Integer nach Byte konvertieren.*

```
Function ConvertIntegerToByte(ByVal nByte As Integer) As Integer
  'Wenn der Parameter zwischen den Wertgrenzen liegt, einfach durchreichen
  If (nByte >= -128) And (nByte <= 127) Then
    ConvertIntegerToByte() = nByte
  'wenn der Parameter zwischen +128 und +255 liegt, dann 256 abziehen
  ElseIf (nByte > 127) And (nByte <= 255) Then
    ConvertIntegerToByte() = nByte - 256
  'wenn Parameter ausserhalb der Wertgrenzen, dann Fehlerwert zurückgeben
  Else
    ConvertIntegerToByte() = 256
```

```
    End If
End Function
```

Das nachfolgende Listing 251 zeigt den prinzipiellen Ablauf zum Aufbau und Beenden einer Socketverbindung. Für eine passive Serververbindung mit Acceptor geschieht das analog. Für den Verbindungsaufbau muss die Verbindung beschrieben werden. Das geht mit einem String, der die folgenden Informationen in einer durch Komma getrennten Liste von Attributen enthält: [Verbindungstyp], [Host], [Port] – z. B. „socket,host=localhost,port=1234“.

Das Makro kontaktiert den Zeitserver-Dienst der Friedrich-Alexander-Universität Erlangen-Nürnberg und holt sich von dort die aktuelle Zeit über den Port 37 im Format „Time“. Das sind die seit dem 01.01.1900 00:00 Uhr vergangenen Sekunden als eine vorzeichenlose 32-Bit-Zahl, also 4 Bytes.

Die Zeitserver-Dienste stellen prinzipiell drei Varianten über Ports zur Verfügung.

- Port 123 - NTP-Protokoll nach RFC 2030 – 16 Bytes in Summe
- Port 37 - Time-Protokoll nach RFC 868 – 4 Byte: Sekunden seit dem 01.01.2900 00:00 Uhr
- Port 13 - Daytime-Protokoll nach RFC 867 – Uhrzeit als ASCII-Text

Die beiden Formate „Time“ und „Daytime“ sind veraltet. Sie haben Schwächen, was die Genauigkeit betrifft und sollten deshalb nicht unbedingt produktiv eingesetzt werden.

Eine Liste vor allem deutscher Zeitserver finden Sie unter folgender Adresse:
http://www.helmut.hullen.de/filebox/DCF77/ntpsrvr.html.

Listing 251. *Über eine Socketverbindung einen Zeitserver kontaktieren.*

```
Sub TimeServer()
  Dim oConnector                'Connection-Objekt
  Dim oConnection               'Die aufzubauende Verbindung
  Dim sConDesc(2) As String     'Verbindungsbeschreibungen
  Dim nReceived(3) As Integer   'Container für Antwort vom Server
  Dim nBytesReceived As Long    'Anzahl empfangener Bytes
  Dim i As Integer
  Dim n As Integer              'Index der Verbindungsbeschreibungen
  Dim dValue As Double          'Long ist zu klein
  Dim sResult As String         'Die Antwort als String

  'Verbindungsbeschreibung
  'enthält die Beschreibung der Verbindung als kommagetrennte Werteliste
  'z.B. socket,host=localhost,port=2345 für eine TCP/IP-Verbindung.
  sConDesc(0) = "socket,host=ntp0.fau.de,port=37"
  sConDesc(1) = "socket,host=ntp1.fau.de,port=37"
  sConDesc(2) = "socket,host=ntp2.fau.de,port=37"

  'Objekt für den Verbindungsaufbau erzeugen
  oConnector = CreateUnoService("com.sun.star.connection.Connector")

  On Local Error Goto NextServer
  'Erstellt eine neue Verbindung zur Interprozesskommunikation.
  'Exception: NoConnectException, ConnectionSetupException
  oConnection = oConnector.connect(sConDesc(n))
  'Antwort empfangen
  nBytesReceived = oConnection.read(nReceived(), 4)

  'Verbindung schließen
  oConnection.close()
```

```
    sResult = "Anzahl empfangener Bytes: " & nBytesReceived & CHR$(10) & _
         "Byte 0: " & nReceived(0) & CHR$(10) & _
         "Byte 1: " & nReceived(1) & CHR$(10) & _
         "Byte 2: " & nReceived(2) & CHR$(10) & _
         "Byte 3: " & nReceived(3)

    'Umrechnen
    For i = 0 To 3
      If nReceived(i) <= 0 Then
        nReceived(i) = nReceived(i) + 256
      End If
      dValue = dValue * 256 + nReceived(i)
    Next
    sResult = sResult & CHR$(10) & _
        "Sekunden seit dem 01.01.1900 um 00:00 Uhr: " &  dValue
        '1 Tag = 86400 = 24h x 60min x 60sec
    sResult = sResult & CHR$(10) & "Ergebnis als Datum und Uhrzeit: " & _
        CDate(CLng(DateValue("01/01/1900")) + dValue/86400)

    MsgBox sResult
    Exit Sub

    NextServer:
      n = n + 1
      'Nächsten Server ausprobieren.
      If n < 3 Then
        Resume
      Else
        MsgBox "Kein Server erreichbar."
      End If
End Sub
```

Bild 85. *Uhrzeit vom Zeitserver.*

Abschließend seien Sie noch auf die Möglichkeit hingewiesen, einen Listener für die Socket-Verbindung zu registrieren. Siehe hierzu auch Abschnitt 10.10. UNO-Listeners und Handlers.

```
'Ereignisbehandlung für die bestehende Verbindung implementieren
oStreamListener = CreateUnoListener("Client_", "com.sun.star.io.XStreamListener")
'Listener an Connection binden
oConnection.addStreamListener(oStreamListener)
```

Später muss dieser Listener natürlich auch wieder entfernt werden:

```
'Remove Streamlistener.
oConnection.removeStreamListener(oStreamListener)
```

Die zu überschreibenden Methoden im Detail:

Tabelle 101. *Die zu überschreibenden Methoden der Socket-Listener.*

Methode	Interface	Beschreibung
started()	XStreamListener	Der Datentransfer wurde gestartet.
closed()	XStreamListener	Der Datentransfer wurde normal oder von außerhalb beendet.
terminated()	XStreamListener	XActiveDataControl.terminated() wurde aufgerufen.
error(oException)	XStreamListener	Fehlerbehandlung bei Verbindungsfehlern. Ein- oder Ausgabefehler ist aufgetreten. Interner Fehler in einer Quelle trat als Ereignis auf. Übergabe eines Objektes mit Informationen zum Ereignis.
disposing(oEventSource)	XEventListener	Hinweis: oEventSource als Struct com.sun.star.lang.EventObject

10.12.5. Asynchroner Callback (A. Heier)

Ein asynchroner Callback ist eine Form der Interprozesskommunikation innerhalb eines Programmes oder zwischen unterschiedlichen Anwendungen. Eine Message Queue ist eine Nachrichtenwarteschlange im Speicherbereich der Anwendung, in der die vom Programm auszuführenden Funktionsaufrufe zur schrittweisen Ausführung hinterlegt sind. Hier werden in einem vom Betriebssystem zur Verfügung gestellten Speicherbereich die Adressen der im Speicher geladenen Funktionen verwaltet.

In diesem Kontext beschränkt sich die Sichtweise auf die Makroprogrammierung innerhalb von LibreOffice und dessen Message Queue. Der hierfür benutzte Service ist ein asynchroner Callback, der es erlaubt, im laufenden Programm Rücksprünge in diese Nachrichtenwarteschlange vorzunehmen. Damit ist es möglich, aktuell laufende Programmteile bei Makros zu umgehen und quasi parallel dazu weitere Prozeduren aufzurufen. Das Adjektiv asynchron weist bereits auf die Fähigkeit zum außerordentlichen Funktionsaufruf hin, womit auch eine nebenläufige Ausführung, also ein Thread im eigentlichen Sinne gemeint ist.

Tabelle 102. *Die einzige Methode des Service com.sun.star.awt.AsyncCallback.*

Methode	Beschreibung
addCallback(XCallback, aData)	Bindet einen Callbackaufruf in die Message Queue ein.

Damit der Callbackaufruf auch von der Laufzeitumgebung ausgeführt wird, ist es notwendig, ihn an einen Listener zu binden. Das Interface XCallback stellt die Methode notify zur Verfügung, mit der der asynchrone Funktionsaufruf möglich wird.

Tabelle 103. *Einzige Funktion im Interface com.sun.star.awt.XCallback.*

Konstante	Beschreibung
notify(aData)	Kümmert sich um den Prozeduraufruf in der Message Queue. Es ist möglich, beliebige Daten mit dem übergebenen Datentyp an den asynchronen Aufruf während der Laufzeit zu übergeben.

Das nachfolgende Listing 252 zeigt die Technik zur Nutzung eines asynchronen Callbacks. Bei der Anmeldung des Listeners ist die Übergabe eines Werts an die notify-Prozedur notwendig. Wenn Sie den Wert nicht weiter als Argument nutzen, können Sie bei der Definition der Prozedur auf den Parameter verzichten (nur in Basic).

Der Callback kann beliebige, soweit mit Basic umsetzbare, Aufgaben ausführen. Für wiederkehrende Aufgaben wird in der notify-Prozedur jeweils ein weiterer Callback gestartet. Ein Beispiel dazu finden Sie im Abschnitt 18.7.2. Countdown als modaler Dialog mit asynchronem Callback.

Tipp Beachten Sie die Lebensdauer der Variablen bei der nebenläufigen Bearbeitung. Für den asynchronen Callback angelegte Variablen werden vom Garbage Collector (automatische Speicherbereinigung) zerstört, wenn das nebenläufige Programm vollständig abgearbeitet wurde. Daher ist es wichtig, wie und wann die Variablen instanziiert werden.

Listing 252. *Anwendung eines asynchronen Callbacks.*

```
Dim oASync
Dim oCallback

Sub TestCallback
  CreateAsyncCallback()
  MsgBox "außerhalb des asynchronen Callbacks"
End Sub

Sub CreateAsyncCallback()
  If IsEmpty(oASync) Then
    oASync = CreateUnoService ("com.sun.star.awt.AsyncCallback")
    oCallback = CreateUnoListener ("Callback1_", "com.sun.star.awt.XCallback")
  End If
  oASync.addCallback(oCallback, 1)
End Sub

Sub Callback1_notify()
     REM hier der abzuarbeitende Programmteil
     Beep()
End Sub
```

10.13. Fazit

Die von Basic bereitgestellten Subroutinen und Funktionen decken einen breiten Rahmen von Operationen ab, die zum Zugriff auf das Innere von OOo benötigt werden. Mit den in diesem Kapitel vorgestellten Methoden inspizieren und verwenden Sie die UNO-Services. Dieses Kapitel untersucht die Grundlagen zur Erzeugung von UNO-Listeners. UNO-Listeners bieten Basisfunktionalitäten, neue UNO-Listeners erweitern diese Funktionalität, um dem Programmierer weitgehende Kontrolle über Anwendungen und Verhalten des Systems zu gewähren. Jeder UNO-Listener beobachtet einen oder mehrere Broadcaster und bietet die Fähigkeit, auf Systemereignisse zu reagieren, die über die Anwendungen hinaus übertragen werden.

11. Der Dispatcher

Dieses Kapitel baut auf dem schon erläuterten OOo-Modell auf und stellt dann den Dispatcher vor. Der Dispatcher bietet einen einfachen Mechanismus, trotz beschränkten Wissens über die interne Arbeitsweise interne Funktionalitäten aufzurufen, ist aber nicht gerade als erste Wahl für Ablaufeingriffe in OOo anzusehen.

11.1. Die Umgebung

OOo teilt die Funktionalität einer Komponente (Dokument) in die drei Teile Model (Modell), Controller und Frame (Rahmen) auf.

FRAME

Kombiniert die einzelnen Teile. Enthält den Controller eines Models und kennt alle Einzelheiten des Ausgabefensters. Der Frame besitzt jedoch keinerlei Funktionalitäten für das Ausgabefenster, er weiß nur, dass eins existiert.

Ein Funktionsaufruf (Dispatch) wird an den Frame geschickt, denn er kontrolliert alles und kann den Dispatch gezielt weiterleiten.

Model

Besteht aus den Dokumentdaten und den Methoden zur Manipulation der Daten.

Über das Modell kann man die Daten direkt bearbeiten.

Controller

Achtet auf aktuelle Darstellung und Daten, manipuliert das Dokument über den von der Benutzerschnittstelle einfließenden Input.

Der Controller kennt solche Dinge wie die aktuelle Selektion und kann Selektionen vornehmen.

Tipp Komponente heißt fast immer geöffnetes Dokument. Aber auch andere Fenster sind Komponenten, zum Beispiel die Basic-IDE und das Hilfefenster.

11.1.1. Zwei unterschiedliche Methoden, OOo zu steuern

Die flexibelste Methode zur Manipulation eines OOo-Dokuments ist der Zugriff auf die internen UNO-Objekte. Der Weg über das Modell bietet bedeutende Steuerungsmöglichkeiten, aber man muss die unterschiedlichen Services und Interfaces gut verstehen.

Einen anderer Weg, der sehr wenig Verständnis der OOo-Interna voraussetzt, bietet der UNO-Dispatcher. Der Dispatcher übernimmt eine Anweisung wie „.uno:Undo“ und kümmert sich um den Rest der Details. Der Frame stellt die für die Arbeit benötigten Dispatcher bereit. Vereinfacht gesagt, stellen Sie sich einen Dispatch wie die Auswahl einer Aktion aus dem Menü vor.

Obwohl die direkte Kontrolle über UNO-Services weitest reichende Funktionalität und Flexibilität bietet, sind doch manche Operationen leichter mit dem Dispatcher erledigt, manchmal ist der Dispatcher auch die einzige Möglichkeit, eine Aufgabe durchzuführen. Zum Beispiel ist für die Nutzung des Zwischenspeichers der Dispatcher die beste Lösung. Sogar der Makrorecorder erledigt fast alle Aufgaben mit einem Dispatcher.

Drei Dinge werden für die Arbeit mit einem Dispatcher benötigt: (1) der Dispatch-Befehl, (2) Argumente zur Steuerung des Dispatch und (3) ein Objekt, das den Dispatch-Befehl ausführen kann (der Dispatch-Provider, normalerweise ein Frame). Jedes Dokument hat einen Controller, der sozusagen

als Schnittstelle zwischen der Außenwelt und dem Dokument fungiert. Mit dem Controller wählen Sie zum Beispiel Text aus, suchen die aktuelle Cursorposition oder ermitteln das aktive Tabellenblatt in der Tabellenkalkulation. Der aktuelle Controller kann auch den Frame des Dokuments zurückgeben, der den Dispatch-Befehl unterstützt.

Listing 253. *Der Service DispatchHelper vereinfacht die Dispatch-Ausführung.*

```
oDispHelper = CreateUnoService("com.sun.star.frame.DispatchHelper")
```

Der Dispatch-Helfer setzt die Funktion executeDispatch ein, in der das meiste der Funktionalität steckt, die für die Dispatch-Ausführung benötigt wird. Die Tabelle 104 listet die Argumente auf, die Sie für die Methode executeDispatch brauchen.

Bei der Ausführung eines Dispatch, wie wir ihn hier behandeln, können Sie den Befehl auch an einen benannten Frame abschicken. Im Normalfall verwenden Sie aber als Zielframe den Frame des aktuellen Dokuments. Dazu verwenden Sie Argument „_self“ oder einen leeren String. In diesem Fall können Sie das Argument auch ganz weglassen. Zu tieferen Informationen über Frames s. Abschnitt 12.1 Der Service Frame ff.

Tabelle 104. *Argumente für executeDispatch.*

Argument	Beschreibung
XDispatchProvider	Dispatch-Provider, der den Dispatch-Befehl ausführt.
URL-String	Der Dispatch-Befehl, als String.
Ziel-Frame als String	Name des Frames, der den Dispatch erhält, entweder „_self“ oder ein leerer String: der aktuelle Frame (jeder andere Wert ist ungültig).
Long	Optionale Suchschalter zur Suche des Ziel-Frames. Entweder null oder keine Angabe (s. Listing 254), weil das Argument nicht unterstützt wird.
PropertyValue()	Optionale Argumente, die von der konkreten OOo-Fassung abhängen.

Listing 254. *Ausführung des Dispatch-Befehls „undo“.*

```
Sub NewUndo
  Dim oDispHelper
  Dim oProvider
  oProvider = ThisComponent.CurrentController.Frame
  oDispHelper = CreateUnoService("com.sun.star.frame.DispatchHelper")
  oDispHelper.executeDispatch(oProvider,".uno:Undo", "", , Array())
End Sub
```

Das folgende Makro fügt den Inhalt der Zwischenablage an die Cursorposition des Dokuments im Frame „MeinFrame“ ein. Es setzt natürlich voraus, dass ein so benannter Frame existiert, entweder der aktive oder ein anderer Frame.

Listing 255. *Ausführung des Dispatch-Befehls „paste“.*

```
Sub NewUndo
  Dim oDispHelper
  Dim oProvider
  Dim nSearch As Long
  nSearch = com.sun.star.frame.FrameSearchFlag.GLOBAL + _
            com.sun.star.frame.FrameSearchFlag.CREATE
  oProvider = ThisComponent.CurrentController.Frame
  oDispHelper = CreateUnoService("com.sun.star.frame.DispatchHelper")
  oDispHelper.executeDispatch(oProvider,".uno:Paste", "MeinFrame", nSearch, Array())
End Sub
```

Die Dispatch-Befehle haben sowohl einen Namen als auch eine Nummer, „Slot“ genannt. Obwohl ein Dispatch mit dem einen oder dem anderen abgesetzt werden kann, so sagten mir doch die Entwickler des ursprünglichen OpenOffice.org, dass sich die Slot-Nummer in der Zukunft ändern kann,

so dass der Gebrauch des Namens sicherer ist. Wenn es nicht ohne Slot geht, so finden Sie in der Bibliothek Tools die Subroutine DispatchSlot, die einen Dispatch einzig mit der Slot-Nummer absetzt.

Listing 256. *Dispatch an einen numerischen Slot.*

```
'Lädt die Bibliothek, die den Befehl DispatchSlot enthält.
GlobalScope.BasicLibraries.loadLibrary("Tools")
DispatchSlot(5301)  'Startet den About-Dialog, genauso wie ".uno:About"
```

Tipp Sie können keine Routine einer Bibliothek aufrufen, bevor die Bibliothek geladen ist. Sie können eine Bibliothek manuell aus dem Makrodialog laden oder wie im Listing 256 mit der Methode LoadLibrary. Die mit OOo installierte Bibliothek Tools enthält die Subroutine DispatchSlot.

Manche Dispatch-Befehle erwarten Argumente. Das Beispiel im Listing 257 setzt einen Dispatch mit Argumenten ab. Der Befehl GoToCell muss wissen, zu welcher Zelle gesprungen werden soll. Dieses Makro setzt den aktuellen Cursor in einem Tabellenblatt in die Zelle B3.

Listing 257. *Dispatch-Befehle können Argumente erhalten.*

```
Dim args2(0) As New com.sun.star.beans.PropertyValue
args2(0).Name = "ToPoint"
args2(0).Value = "$B$3"      ' Position nach B3
Dim oDispHelper
Dim oProvider
oProvider = ThisComponent.CurrentController.Frame
oDispHelper = CreateUnoService("com.sun.star.frame.DispatchHelper")
oDispHelper.executeDispatch(oProvider, ".uno:GoToCell", "", 0, args2())
```

11.1.2. Dispatch-Befehle suchen

Es ist schwierig, an eine vollständige Liste der Dispatch-Befehle zu kommen. Ich habe früher einmal eine komplette Liste aller unterstützten Dispatch-Befehle zusammengetragen, aber die Liste ändert sich laufend, und mir ist auch nicht klar, ob sie überhaupt von großem Nutzen ist. Wie auch immer, ich werde Ihnen zeigen, wie Sie die Liste selbst erstellen können. Falls Bedarf sein sollte, könnte ich überzeugt werden, eine neue Liste zusammenzustellen. Mein Dank an Ariel Constenla-Haile für diese Information.

Informationen über das WIKI holen

http://wiki.services.openoffice.org/wiki/Framework/Article/OpenOffice.org_3.x_Commands enthält eine Liste von Befehlen und Slots. Sicher fehlen einige Befehle, wie zum Beispiel „.uno:ObjectTitleDescription“. Was noch schwerer wiegt, ist der Umstand, dass eine ältere Softwareversion abgebildet ist. Einige Namen differieren zwischen AOO und LO. Zum Beispiel heißt das, was in AOO view200 heißt, in LO Zoom200Percent.

Im Quelltext suchen

Ich habe mir den Quelltext heruntergeladen und nach „.uno:“ gesucht, um die UNO-Befehle zu finden. Das ist zwar ein recht mühsam, aber es funktioniert.

Das Interface durchsuchen

Das folgende Makro nutzt das LO-Singleton theUICommandDescription (in AOO den Service UICommandDescription) zur Aufzählung der unterstützten Module. Ein neues Tabellendokument wird erstellt, und dann für jedes Modul eine Tabelle. Die von einem Modul unterstützten Befehle

werden in der betreffenden Tabelle eingetragen. Beachten Sie, dass das Makro Zeit braucht, weil die Datenmenge doch sehr groß ist.

Listing 258. *Holt die Befehle vom ModuleManager.*

```
Sub Print_All_Commands
  'Erstellt ein neues Tabellendokument zur Aufnahme der Dispatch-Befehle
  Dim oDoc
  oDoc = StarDesktop.loadComponentFromURL(_
                    "private:factory/scalc", "_default", 0, Array())

  Dim oSheets           : oSheets = oDoc.getSheets()

  'Der AOO-Service UICommandDescription bietet den Zugang zu Befehlen der
  'Benutzerschnittstelle als Teil der OOo-Module wie Writer oder Calc.
  'In LO ist der Service seit der Version 4.3 veraltet.
  'Verwenden Sie stattdessen das Singleton theUICommandDescription.
  Dim oUICommandDescription
  oContext = GetProcessServiceManager().DefaultContext
  oUICommandDescription = _
      oContext.getValueByName("/singletons/com.sun.star.frame.theUICommandDescription")
  If IsEmpty(oUICommandDescription) Then
    'AOO nutzt den älteren Singleton-Typ:
    oUICommandDescription = CreateUnoService("com.sun.star.frame.UICommandDescription")
  End If

  'Identifiziert die Office-Module und bietet lesenden Zugriff auf die Konfiguration
  'der Office-Module.
  Dim oModuleIdentifier
  oModuleIdentifier = CreateUnoService("com.sun.star.frame.ModuleManager")

  Dim oModuleUICommandDescription, aModules$(), aCommand
  Dim aCommands$()
  Dim n&, i&
  'Holt die Liste der Modulnamen wie "com.sun.star.presentation.PresentationDocument".
  'Erstellt eine Tabelle für jedes Modul.
  aModules = oModuleIdentifier.getElementNames()
  For n = 0 To UBound(aModules)
    oModuleUICommandDescription = oUICommandDescription.getByName(aModules(n))

    'Holt die von diesem Modul unterstützten Befehle.
    ReDim aCommands$()
    aCommands = oModuleUICommandDescription.getElementNames()

    If n <= UBound(oSheets.getElementNames()) Then
      oSheets.getByIndex(n).setName(aModules(n))
    Else
      oSheets.insertNewByName(aModules(n), n)
    End If

    oSheets.getCellByPosition(0, 0, n).getText().setString("Command")
    oSheets.getCellByPosition(1, 0, n).getText().setString("Label")
    oSheets.getCellByPosition(2, 0, n).getText().setString("Name")
    oSheets.getCellByPosition(3, 0, n).getText().setString("Popup")
    oSheets.getCellByPosition(4, 0, n).getText().setString("Property")

    For i = 0 To UBound(aCommands)
      aCommand = oModuleUICommandDescription.getByName(aCommands(i))
```

```
      Dim sLabel$, sName$, bPopup As Boolean, nProperty&, k&
      For k = 0 To UBound(aCommand)
        If aCommand(k).Name = "Label" Then
          sLabel = aCommand(k).Value
        ElseIf aCommand(k).Name = "Name" Then
          sName = aCommand(k).Value
        ElseIf aCommand(k).Name = "Popup" Then
          bPopup = aCommand(k).Value
        ElseIf aCommand(k).Name = "Property" Then
          nProperty = aCommand(k).Value
        End If
      Next

      oSheets.getCellByPosition(0, i + 1, n).getText().setString(aCommands(i))
      oSheets.getCellByPosition(1, i + 1, n).getText().setString(sLabel)
      oSheets.getCellByPosition(2, i + 1, n).getText().setString(sName)
      If bPopup Then
        oSheets.getCellByPosition(3, i + 1, n).getText().setString("True")
      Else
        oSheets.getCellByPosition(3, i + 1, n).getText().setString("False")
      End If
      oSheets.getCellByPosition(4, i + 1, n).getText().setString(CStr(nProperty))
    Next

    Dim oColumns     'Tabellenspalten
    oColumns = oSheets.getByIndex(n).getColumns()
    Dim j%
    For j = 0 To 4
      oColumns.getByIndex(j).setPropertyValue("OptimalWidth", True)
    Next
  Next
End Sub
```

XDispatchInformationProvider baut eine Liste der Dispatch-Befehle auf, die vom aktuellen Controller zurückgegeben wird, und gibt sie in einem Tabellendokument aus. Vor einigen Jahren, als ich dieses Makro schrieb, wurden weniger als 32.000 Zeilen ausgegeben. Mit LO 6 sind es über 300.000 mit einer Reihe von Duplikaten. Benutzen Sie das Listing 259 daher nur aus Neugier und trinken Sie in Ruhe Ihren Kaffee. Ich habe einen Abbruch bei 300.000 Zeilen eingebaut.

Listing 259. *Holt die Befehle vom aktuellen Controller.*

```
Sub XDispatchInformationProvider
  'Erstellt ein neues Tabellendokument zur Aufnahme der Dispatch-Befehle
  Dim oDoc
  oDoc = StarDesktop.loadComponentFromURL(_
                    "private:factory/scalc", "_default", 0, Array())

  'Tabellenblatt 1
  Dim oSheet : oSheet = oDoc.getSheets().getByIndex(0)
  oSheet.getCellByPosition(0, 0).getText().setString("Group")
  oSheet.getCellByPosition(1, 0).getText().setString("Command")

  Dim oController :   oController = ThisComponent.getCurrentController()
  If IsNull(oController) Or _
    Not HasUnoInterfaces(oController, _
                         "com.sun.star.frame.XDispatchInformationProvider") Then
    'TODO: ein Warnhinweis
```

```
      Exit Sub
    End If

    'IDs der Befehlsgruppen
    Dim iSupportedCmdGroups%()
    iSupportedCmdGroups = oController.getSupportedCommandGroups()

    Dim i&, j&, k&
    For i = 0 To UBound(iSupportedCmdGroups)
      'Array der Befehle in der entsprechenden Gruppe
      Dim aDispatchInfo()
      aDispatchInfo = _
               oController.getConfigurableDispatchInformation(iSupportedCmdGroups(i))

      For j = 0 To UBound(aDispatchInfo)
        'Struct mit den Eigenschaften Command und GroupId
        Dim aDispatchInformation
        aDispatchInformation = aDispatchInfo(j)
        k = k + 1
        oSheet.getCellByPosition(0, k).getText().setString(iSupportedCmdGroups(i))
        oSheet.getCellByPosition(1, k).getText().setString(aDispatchInformation.Command)
        'Ohne Abbruch scheint es immer weiter zu gehen
        If k > 300000 Then
          Exit Sub
        End If
      Next
    Next
  End Sub
```

In einem E-Mailwechsel mit mir hat ein Entwickler Zweifel an der Vollständigkeit und Genauigkeit der Liste geäußert, weil die von UICommandDescription bereitgestellten bibliografischen Befehle solche wie „.uno:ArrowShapes“ enthielten, was in diesem Kontext unsinnig ist.

Den Quellcode lesen

SFX2-basierte Komponenten nutzen eine gemeinsame Grundklasse für die Einrichtung ihres Controller-Objekts, SfxBaseController. Für Module auf der Basis von SFX2 ist es möglich, die Dispatch-Befehle mit ihren Argumenten aus den zu den einzelnen Modulen gehörenden SDI-Dateien zu parsen (Verzeichnis <Modul>/sdi/). Oder direkt aus den Header-Dateien in der Kompilierungsumgebung. Halten Sie sich an Dateien mit Namen wie sfxslots.hxx, svxslots.hxx, scslots.hxx, sdslots.hxx, swslots.hxx, basslots.hxx und smslots.hxx.

Für neue Module wird es wohl einfacher sein, durch den Quellcode zu browsen, zum Beispiel in chart2:

- chart2/source/controller/main/ChartController.cxx
- chart2/source/controller/main/DrawCommandDispatch.cxx
- chart2/source/controller/main/ShapeController.cxx

Tipp Die Groß-/Kleinschreibung ist bei den Namen der Dispatch-Befehle zu beachten.

11.2. Ein Makro mit dem Dispatcher schreiben

Wenn Sie keine Methode im UNO-API für eine spezielle Aufgabe finden können, fällt die Wahl als nächstes auf den Dispatcher. Makros, die vom Makrorecorder erstellt werden, nutzen Dispatches. Über das Menü **Extras > Makros > Makro Aufzeichnen** starten Sie den Makrorecorder.

Tipp Nicht alle Komponenten unterstützen den Makrorecorder, zum Beispiel Draw.

Im Makrorecorder gibt es viele Einschränkungen. Zum Beispiel verfolgt er nicht, was passiert, wenn ein Dialog geöffnet wird. Ich habe diese Einschränkung entdeckt, als ich den Recorder nutzte, um ein Makro zur Einfügung von Textdateien zu erstellen. Der Importfilter „Text Kodiert (*.txt)“ öffnet einen Dialog mit Fragen über die einzufügende Datei. Die in diesem Dialog verwendeten Werte werden vom Makrorecorder nicht mit übernommen.

Wenn Sie ein Makro mit Hilfe des API schreiben wollen, fangen Sie damit an, dass Sie sich durch die unterstützten Befehle lesen, wie sie von einem der oben aufgeführten Makros gezeigt werden.

Der Befehl SendOutLineToStarImpress erstellt eine Impress-Präsentation mit einer Gliederung, die aus den Überschriften des aktuellen Dokuments erstellt wird. Argumente sind nicht nötig.

Listing 260. *Erstellt eine Präsentation mit der Überschriftengliederung dieses Dokuments.*

```
Sub CreateOutlineInImpress
  Dim oDispHelper
  Dim oProvider
  oProvider = ThisComponent.CurrentController.Frame
  oDispHelper = CreateUnoService("com.sun.star.frame.DispatchHelper")
  oDispHelper.executeDispatch(oProvider, ".uno:SendOutlineToStarImpress", _
    "", , Array())
End Sub
```

11.3. Dispatch-Fehlfunktion – ein erweitertes Zwischenspeicherbeispiel

Das Anliegen war einfach: Ein komplettes Writer-Dokument soll in die Zwischenablage kopiert werden. Der Makrorekorder produzierte sehr schnell eine Lösung. Leider misslingt das Makro in LO, wenn es von einer dem Dokument hinzugefügten Schaltfläche gestartet wird. Es funktioniert aber aus der IDE heraus. In AOO gibt es kein Fehlverhalten, auch nicht aus einer Schaltfläche heraus. Das Problem wird umgangen, wenn der Fokus vor dem Ausführen der Dispatch-Befehle auf das aktuelle Dokumentfenster gesetzt wird.

Listing 261. *Alles auswählen und kopieren mit Hilfe von Dispatch-Befehlen.*

```
Sub CopyToClipboard_Dispatch
  Dim document
  Dim dispatcher
  document   = ThisComponent.CurrentController.Frame

  ' Die nächste Zeile wurde NICHT vom Makrorekorder eingefügt. Ohne diese Zeile
  ' scheitert das Kopieren beim Starten aus einer Schaltfläche heraus.
  document.ContainerWindow.setFocus

  dispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")
  dispatcher.executeDispatch(document, ".uno:SelectAll", "", 0, Array())
  dispatcher.executeDispatch(document, ".uno:Copy", "", 0, Array())
End Sub
```

Die API ist robuster als der Dispatcher und liefert korrekte Ergebnisse, wo ein Dispatch versagen könnte. So mag die API auch dann notwendig sein, wenn das Makro aus OOo im Headless-Modus gestartet wird, das heißt ohne Bildschirm oder andere grafische Ausgabe. Dieser Modus wird allerdings in diesem Buch nicht weiter behandelt.

Listing 262. *Alles auswählen und kopieren mit Hilfe der API.*

```
Sub CopyToClipboard_API()
  Dim o          'Übertragbarer Inhalt
  Dim oClip      'Zwischenspeicher-Service
  Dim oContents 'Inhalt des Zwischenspeichers
  Dim sClipName As String

'Selektiert den gesamten Dokumentinhalt. Die Selektion ist kopierbar.
  ThisComponent.CurrentController.select(ThisComponent.Text)
  o = ThisComponent.CurrentController.getTransferable()

  sClipName = "com.sun.star.datatransfer.clipboard.SystemClipboard"
  oClip = CreateUnoService(sClipName)
'Überträgt die Auswahl in die Zwischenablage
  oContents = oClip.setContents(o, Null)
End Sub
```

11.4. Fazit

Die Dispatch-Befehle sind mächtig und erfordern nur geringe Kenntnisse der internen Arbeitsweise von OOo. Obwohl manche Funktionalität, zum Beispiel der Undo-Befehl, nur mit dem Dispatcher möglich ist, wird man für Makros, die lange Zeit nutzbar sein sollen, mit der direkten Verwendung der internen Objekte besser fahren.

12. StarDesktop

Der Desktop dient als Hauptanwendung, die OOo steuert. Dieses Kapitel präsentiert allgemeine Techniken – Zugriff auf indexierte Objekte, Listen geöffneter Dokumente, Öffnen neuer Dokumente – zugleich mit der Behandlung und Demonstration der Basisfunktionalität des Objekts Desktop. Neben dem Desktop-Objekt behandelt dieses Kapitel auch die Variable ThisComponent.

Das Desktop-Objekt ist seit LO 4.3 das Singleton com.sun.star.frame.theDesktop, vorher und für AOO auch jetzt noch ein com.sun.star.frame.Desktop-Service, mit vier primären Funktionen.

1. StarDesktop verhält sich als Frame. Der Desktop ist der übergeordnete Frame und steuert die Frames aller Dokumente.
2. StarDesktop verhält sich als Desktop. Der Desktop ist die Hauptanwendung mit der Fähigkeit, Dokumente zu schließen – wenn zum Beispiel die Anwendung heruntergefahren wird.
3. StarDesktop verhält sich als Dokumentöffner. Die Rolle als Hauptanwendung erlaubt dem Desktop auch, vorhandene Dokumente zu öffnen und neue Dokumente zu erzeugen.
4. StarDesktop verhält sich als Ereignis-Broadcaster. Der Desktop benachrichtigt vorhandene Dokumente (oder jeden anderen Listener), wenn etwas geschehen ist – wenn zum Beispiel die Anwendung geschlossen werden wird.

Der Desktop wird als Hauptanwendungsobjekt beim Start von OOo erzeugt. Den Zugriff auf das OOo-Desktop-Objekt erhalten Sie über die globale Variable StarDesktop.

12.1. Der Service Frame

Der Desktop ist ein com.sun.star.frame.Frame-Service (zur Erinnerung: ein Objekt kann, und tut es auch normalerweise, mehr als einen Service einbinden). Für ein Dokument ist der Hauptzweck des Frames, als Bindeglied zwischen dem Dokument und dem sichtbaren Fenster zu dienen. Der Frame des Desktops aber dient im Wesentlichen als Wurzelframe, der alle anderen Frames enthält. Ein Frame kann eine Komponente und keine oder mehrere Unterframes enthalten – betrachten Sie der Einfachheit halber eine Komponente als Dokument. Im Falle des Desktops sind alle anderen Frames Unterframes dieses Wurzelframes und die Komponente (das Dokument) ist eine Gruppe von Daten. Jedes Dokument enthält einen Frame, über den es mit dem sichtbaren Fenster interagiert. Der Desktop aber ist ein Frame, damit er alle Frames der Dokumente enthalten, steuern und darauf zugreifen kann.

Der Service com.sun.star.frame.Frame – kurz gesagt: der Frame – bietet eine ganze Reihe interessanter Eigenheiten, die als Teil des Desktop-Objekts nicht unbedingt nützlich sind. Zum Beispiel sind Title und StatusIndicator als Teil des Frame-Service definiert, sind aber für das Desktop-Objekt nutzlos, denn es hat kein Anzeigefenster. Diese Eigenschaften sind nur in einem Frame mit einem Anzeigefenster sinnvoll.

Tipp	Obwohl es möglich ist, über den Desktop als Frame-Service eine Liste der enthaltenen Frames zu erhalten, ist es im allgemeinen besser, über das Interface XDesktop die Dokumente statt der Frames zu listen. Auf Frames greife ich gewöhnlich zu, um Fenstertitel zu erhalten.

Mit der Desktop-Methode getActiveFrame() referenzieren Sie den aktiven Frame, s. Listing 263. Der aktive Frame ist derjenige, der den aktuellen Fokus hat. Wenn das Fenster mit dem aktuellen Fokus kein OOo-Fenster ist, gibt getActiveFrame den letzten OOo-Frame zurück, der den Fokus hatte.

Listing 263. *Gibt den Titel des aktuellen Frames aus.*

```
Print StarDesktop.getActiveFrame().Title
```

Mit der Methode getFrames() listen oder durchsuchen Sie alle im Desktop enthaltenen Frames. Es wird ein Objekt zurückgegeben, das das Interface com.sun.star.frame.XFrames einbindet. Ein Frame kann andere Frames enthalten. Das Interface XFrames bietet den Zugang zu den enthaltenen Frames.

Tipp	Wenn Sie die Webadresse für die API-Informationen über das Interface XFrames suchen, so verwenden Sie den vollen Interface-Namen. Es ist für Sie wichtig zu lernen, Webseiten auf den API-Seiten mit dem kompletten Service- oder Interface-Namen zu suchen.

12.1.1. Das Interface XIndexAccess

Das Interface XFrames erbt auch vom Interface com.sun.star.container.XIndexAccess. Wie der Name schon andeutet, erlaubt dieses Interface den Zugriff auf die enthaltenen Frames über einen numerischen Index. Eine große Zahl anderer Interfaces stammen auch vom Interface XIndexAccess ab und bieten dadurch die Möglichkeit, auf die enthaltenen Element mit einem einfachen numerischen Index zuzugreifen, s. Listing 264 und Bild 86. Auch das Objekt Sheets in einem Tabellendokument lässt den Zugriff auf jede Tabelle über den Index (auch über den Namen) zu.

Tipp	Prägen Sie sich die Verwendung des Interface XIndexAccess ein, denn OOo nutzt diesen Dienst an vielen anderen Stellen.

Listing 264. *Ausgabe der Frame-Titel geöffneter Komponenten.*

```
Sub DisplayFrameTitles
  Dim vFrames As Variant            'Alle Frames
  Dim vFrame As Variant             'Ein einzelner Frame
  Dim i As Integer                  'Index zur Liste der Frames
  Dim s As String                   'Der Ausgabestring

  vFrames = StarDesktop.getFrames() 'Holt alle Frames

  REM getCount() gibt die Zahl der enthaltenen Frames zurück.
  REM Wenn es fünf Frames sind, dann hat i die Werte 1, 2, 3, 4 und 5
  REM Die Methode getByIndex(i) ist jedoch null-basiert. Das bedeutet,
  REM dass sie die Werte 0, 1, 2, 3 und 4 benötigt.
  For i = 1 To vFrames.getCount()
    vFrame = vFrames.getByIndex(i - 1)
    s = s & CStr(i - 1) & " : " & vFrame.Title & Chr$(10)
  Next
  MsgBox s, 0, "Frame-Titel"
End Sub
```

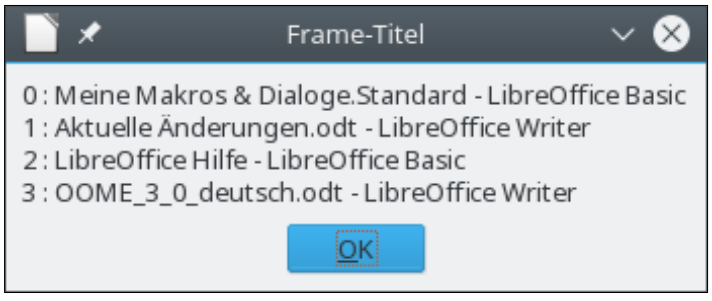


Bild 86. *Titel der Frames der obersten Ebene.*

12.1.2. Frames mit den FrameSearchFlag-Konstanten suchen

Die com.sun.star.frame.FrameSearchFlag-Konstanten nutzen Sie zur Suche nach Frames, s. Tabelle 105, und zwar als Argument der Methode queryFrames() – definiert im Interface XFrames – zur Erstellung einer Aufzählungsliste von Frames. Die FrameSearchFlag-Konstanten braucht man auch,

um einen Frame zu suchen, wenn ein Dokument geladen wird und festgelegt wird, welche Frames einen Dispatch erhalten. Sie haben die FrameSearchFlag-Konstanten schon im Kapitel 11 kennen gelernt.

Tabelle 105. *Die Konstantengruppe com.sun.star.frame.FrameSearchFlag.*

#	Name	Beschreibung
0	AUTO	Veraltet. Verwenden Sie 6 = SELF + CHILDREN.
1	PARENT	Schließt den Basisframe ein (Elternteil).
2	SELF	Schließt diesen Frame ein.
4	CHILDREN	Schließt die von diesem Frame abgeleiteten Frames ein (Kinder).
8	CREATE	Erzeugt einen Frame, wenn der gewünschte nicht gefunden wird.
16	SIBLINGS	Schließt die vom Basisframe abgeleiteten Frames ein (Geschwister).
32	TASKS	Schließt alle Frames aller Tasks in der aktuellen Frames-Hierarchie ein.
23	ALL	Schließt alle Frames außer TASKS-Frames ein: 23 = 1+2+4+16 = PARENT + SELF + CHILDREN + SIBLINGS.
55	GLOBAL	Schließt jeden Frame ein: 55 = 1+2+4+16+32 = PARENT + SELF + CHILDREN + SIBLINGS + TASKS.

Die Werte der Tabelle 105 sind Bitschalter, die kombiniert werden können. Zum Beispiel gibt es die Werte für ALL und GLOBAL nur aus Zweckmäßigkeit, denn es handelt sich jeweils um eine Kombination anderer Schalter, wie man aus der Tabelle entnehmen kann. Sie können Ihre eigenen Werte kreieren, wenn keine geeignete Kombination zur Verfügung steht. Suchen Sie zum Beispiel nach ALL und lassen einen Frame erstellen, wenn nichts gefunden wurde: mit dem Wert 31 = 23 + 8 = ALL + CREATE.

Tipp FrameSearchFlags sind Bitschalter, die mit Hilfe des Operators Or kombiniert werden können. Wenn Sie nicht wissen, was ein Bitschalter ist, lächeln und nicken Sie einfach nur.

Der Code im Listing 265 nutzt die FrameSearchFlag-Konstanten, um Frames zu finden, die Kinder des Desktop-Objekts sind. Die Ausgabe vom Listing 265 gleicht der Ausgabe vom Listing 264 darin, dass eine Liste aller vom Desktop-Objekt abgeleiteten Frames erstellt wird. Beachten Sie, dass von der Methode queryFrames() ein Array zurückgegeben wird. Das weiß ich, weil ich auf der API-Website nachgeschaut habe. Sie können zwar das zurückgegebene Objekt inspizieren, um zu sehen, was für ein Typ es ist, es ist aber nicht möglich, durch reine Inspizierung die Werte der Argumente zur Methode queryFrames() zu bestimmen.

Listing 265. *Anfrage QueryFrames zur Auflistung der Frame-Titel.*

```
Sub QueryFrames
  Dim vFrames As Variant            'Alle Frames
  Dim vFrame As Variant             'Ein einzelner Frame
  Dim i As Integer                  'Index zur Liste der Frames
  Dim s As String                   'Der Ausgabestring
  REM Ruft die Methode queryFrames() des Interface XFrames auf.
  REM Ein Argument ist erforderlich, ein FrameSearchFlag.
  REM Hiermit werden die Kind-Frames des Desktop gesucht.
  vFrames = StarDesktop.getFrames().queryFrames(_
                com.sun.star.frame.FrameSearchFlag.CHILDREN)

  For i = LBound(vFrames) To UBound(vFrames) ' Der Rückgabewert ist ein Array.
    s = s & vFrames(i).Title & Chr$(10)      ' Titel und neue Zeile für die Ausgabe.
  Next
```

```
    MsgBox s, 0, "Frame-Titel"                ' Ausgabe der Titel.
End Sub
```

12.2. Das Interface XEventBroadcaster

Wenn in OOo gewisse wichtige Ereignisse (Events) geschehen, wird das spezielle Ereignis an alle Objekte übertragen, die als Listener für dieses Ereignis angemeldet sind – zum Beispiel, um darüber informiert zu sein, wenn ein Dokument geschlossen werden soll. Das Interface com.sun.star.document.XEventBroadcaster lässt den Desktop als Broadcaster agieren.

Zum Hinzufügen und Entfernen von Event-Listeners definiert das Interface XEventBroadcaster die Methoden addEventListener() und removeEventListener(). Diese beiden Methoden werden gewöhnlich nicht direkt aufgerufen, weil üblicherweise listenerspezifische Methoden verwendet werden. Zum Beispiel enthält das Controller-Objekt für Tastaturereignisse die Methoden addKeyHandler() und removeKeyHandler().

12.3. Das Interface XDesktop

Das Interface com.sun.star.frame.XDesktop definiert die Grundfunktionalität des Service Desktop. Der Desktop enthält die Komponenten der obersten Ebene, die in einem Frame betrachtet werden können. Er enthält und steuert also den Lebenszyklus der OOo-Dokumente, des Hilfefensters, der Basic-IDE und der anderen Komponententypen.

Der Desktop ist ein Frame-Service, so dass er als Wurzelframe agieren kann, der alle anderen Frames enthält. Der Desktop hat daher Zugriff und Kontrolle auf alle anderen Frames. Diese Kontrolle schließt die Fähigkeit ein, Dokumente zu öffnen, alle Frames zu schließen und OOo zu beenden.

12.3.1. Schließen des Desktops und der enthaltenen Komponenten

Um den Desktop und alle enthaltenen Frames zu schließen, rufen Sie die Methode terminate() auf. Es gibt keine Garantie dafür, dass der Desktop geschlossen wird, es ist eher ein Vorschlag oder eine Bitte, OOo möge doch beendet werden. Vor dem Schließen fragt der Desktop jedes Objekt, das vor dem Schließen benachrichtigt werden wollte, ob es es damit einverstanden ist. Falls nur ein Listener das verneint, wird OOo nicht beendet. Jedes geöffnete Dokument ist als Listener für das Schließen angemeldet und fragt Sie, ob Sie ein Dokument speichern möchten, wenn es noch nicht geschehen ist. So funktioniert das – falls Sie sich einmal darüber gewundert haben.

Alle OOo-Dokumenttypen unterstützen das Interface com.sun.star.util.XCloseable. Dieses Interface definiert die Methode „close(bForce As Boolean)“. Wenn bForce (= gewaltsam) False ist, darf das Objekt sich weigern, geschlossen zu werden. Im Falle von True ist keine Weigerung möglich.

Laut Mathias Bauer, einem der in den Anfangstagen führenden Entwickler, unterstützt das Desktop-Objekt aus Gründen der Rückwärtskompatibilität das Interface XCloseable nicht. Die Methode terminate() wurde verwendet, bevor entschieden wurde, dass sie zum Schließen von Dokumenten und Fenstern nicht gut geeignet war. Wenn diese Methode nicht schon eingesetzt gewesen wäre, würde auch der Desktop das Interface XCloseable unterstützen.

Der Code im Listing 266 zeigt, wie man ein Dokument auf eine sichere, von der verwendeten OOo-Version unabhängige Art schließt. Wenn Sie genau wissen, dass Ihr Code auf OOo 1.1 oder später läuft, können Sie einfach die Methode close() nehmen.

Listing 266. *Die sichere Art, ein Dokument in jeder OOo-Version zu schließen.*

```
If HasUnoInterfaces(oDoc, "com.sun.star.util.XCloseable") Then
  oDoc.close(True)
Else
  oDoc.dispose()
End If
```

Der Code im Listing 266 setzt voraus, dass die Variable oDoc ein OOo-Dokument referenziert. Manche Komponententypen unterstützen das Interface XCloseable nicht. Ein Beispiel ist die Basic-IDE. Der Desktop hat Methoden, die aktuell geöffneten Dokumente aufzulisten und auf die aktuelle Komponente zuzugreifen. Wir kommen bald darauf zurück.

Die Methode dispose() wirft ein Dokument gnadenlos heraus, auch wenn es verändert wurde. Das macht die Methode close() nicht. Mit setModified(False) wird ein modifiziertes Dokument als nicht modifiziert gekennzeichnet, wodurch die Methode close() daran gehindert wird, sich darüber zu beklagen, dass Sie ein modifiziertes Dokument schließen wollen.

12.3.2. Komponenten enumerieren mit XEnumerationAccess

Normalerweise bezieht sich eine Komponente auf ein OOo-Dokument, es kann aber auch etwas anderes sein, so die Basic-IDE oder die eingebaute Hilfeseite. Von der Methode getComponents() – definiert im Interface XDesktop – wird eine ungezählte Auflistung, eine so genannte Enumeration, der vom Desktop kontrollierten Komponenten zurückgegeben. Dieses Objekt unterstützt das Interface com.sun.star.container.XEnumerationAccess.

Tipp	OOo hat viele Interfaces, die eine Liste von Objekten zurückgeben – manche Methoden liefern ein Array von Objekten. Im Desktop wird XIndexAccess zur Auflistung von Frames verwendet und XEnumerationAccess zur Auflistung von Komponenten.

Eine Komponente, die zugleich ein OOo-Dokument ist, unterstützt das Interface XModel. Ein Modell repräsentiert die zugrunde liegenden Dokumentendaten. Wenn also eine Komponente das Interface XModel nicht unterstützt, ist sie kein OOo-Dokument. Mit anderen Worten, die Unterstützung des Interface XModel impliziert, dass eine Komponente Daten enthält. Wenn es keine Daten gibt, ist es kein Dokument. Prüfen Sie jede Komponente mit der Funktion HasUnoInterfaces, ob sie ein OOo-Dokument ist. Um ein bestimmtes Dokument zu finden, durchsuchen Sie alle Komponenten und überprüfen den URL oder andere unterscheidende Kriterien.

Es ist möglich, dass ein Dokument keinen URL hat. In diesem Falle ist der URL-String leer. Die Subroutine FileNameOutOfPath versagt bei einem Leerstring. EnumerateComponentNames sichert diesen Fall ab.

Listing 267. *Zeigt, wie Komponenten enumeriert werden.*

```
Sub EnumerateComponentNames
  Dim vComps          'Objekt des enumerierten Zugriffs
  Dim vEnumerate      'Enumerationsobjekt
  Dim vComp           'Einzelne Komponente
  Dim s As String     'Ausgabestring
  Dim sURL As String  'Dokument-URL

  REM Die Bibliothek Tools wird geladen, weil deren Modul Strings
  REM die Funktion FileNameOutOfPath() enthält.
  GlobalScope.BasicLibraries.loadLibrary("Tools")

  vComps = StarDesktop.getComponents()   'com.sun.star.container.XEnumerationAccess
  If Not vComps.hasElements() Then     'Das muss ich nicht tun, aber
    Print "Es gibt keine Komponenten"  'es zeigt, dass ich es tun kann
    Exit Sub
  End If

  vEnumerate = vComps.createEnumeration() 'com.sun.star.container.XEnumeration
  Do While vEnumerate.hasMoreElements()   'Gibt es noch weitere Elemente?
    vComp = vEnumerate.nextElement()      'Das nächste Element
```

```
    REM Der URL wird nur von Dokument-Komponenten bezogen.
    REM Dadurch wird beispielsweise die IDE überschlagen.
    If HasUnoInterfaces(vComp, "com.sun.star.frame.XModel") Then
      sURL = vComp.getURL()
      If sURL = "" Then
        s = s & "<Komponente ohne URL>" & Chr$(10)
      Else
        s = s & FileNameOutOfPath(sURL) & Chr$(10)
      End If
    End If
  Loop
  MsgBox s, 0, "Dokumentnamen"
End Sub
```

Listing 267 gibt Bild 87 aus, mit der Liste der Dateinamen aller aktuell geöffneten Dokumente.

Bild 87. *Dateinamen der aktuell geöffneten Dokumente.*

12.3.3. Die aktuelle Komponente

Mit getCurrentComponent() erhält man die Komponente, die den aktuellen Fokus hat, was allerdings nicht notwendigerweise ein OOo-Dokument sein muss. Um das aktuellste OOo-Dokument zu erhalten, nutzen Sie die globale Variable ThisComponent.

Listing 268. *ThisComponent referenziert das aktuelle OOo-Dokument.*

```
Print ThisComponent.getURL()                    'Funktioniert aus der Basic-IDE
Print StarDesktop.getCurrentComponent().getURL() 'Funktioniert nicht aus der Basic-IDE
```

Tipp StarDesktop.getCurrentComponent() gibt die aktuell fokussierte Komponente zurück. Wenn die Basic-IDE geöffnet ist, wird diese Komponente zurückgegeben. Mit der globalen Variablen ThisComponent wird das aktuellste Dokument zurückgegeben.

12.3.4. Die aktuelle Komponente (noch einmal)

Obwohl StarDesktop.getCurrentComponent() in vielen Situationen verwendet werden kann, ist es doch keine verlässliche Methode, das Dokument zu erreichen, das als letztes den Fokus hatte. Sie liefert entweder die Komponente, die in diesem Moment den Fokus hat, oder die Komponente, die den Fokus hatte, bevor der auf eine andere Anwendung überging. Dieses Verhalten verursacht Probleme beim Debuggen von Basic-Programmen, denn dann ist die Basic-IDE die aktuelle Komponente. ThisComponent ist daher StarDesktop.CurrentComponent vorzuziehen. Wenn der Fokus auf ein neues Dokumentfenster gesetzt wird, ändert sich nichts am Wert von ThisComponent. Dieser Wert wird beim Beginn des Makros gesetzt und dann nicht mehr verändert.

Listing 269. *Zwei Methoden, den Fokus auf das Dokument oDoc2 zu setzen.*

```
oDoc2.CurrentController.Frame.ContainerWindow.toFront()
oDoc2.CurrentController.Frame.activate()
```

Tipp Wenn ein Makro infolge eines Ereignisses aufgerufen wird, referenziert ThisComponent das Dokument, das das Ereignis produziert hat, falls das Makro nicht in einem anderen Dokument sein sollte. Daher ist es besser, globale Makros, die von anderen Dokumenten aufgerufen würden, in einer globalen Bibliothek und nicht innerhalb eines Dokuments unterzubringen.

Makros können durch bestimmte auftretende Ereignisse (möglicherweise als registrierte Event-Handlers) aufgerufen werden. Wenn ein Makro gestartet wird, weil ein Ereignis eintrat, referenziert ThisComponent das Dokument, von dem das Ereignis ausging, auch wenn es nicht den aktuellen Fokus hat. Die Variable ThisComponent ist nur in Basic verfügbar.

Tipp Es ist möglich, ein Makro auszuführen, wenn kein Dokument geöffnet ist. Es versteht sich von selbst, dass das Makro in einer Bibliothek der Anwendungsebene gespeichert sein muss und nicht in einem Dokument.

12.3.5. Der aktuelle Frame

Obwohl der Service Frame dem Desktop erlaubt, die enthaltenen Frames zu listen, ist es aber das Interface XDesktop, das die Methode getCurrentFrame() definiert. Diese Methode gibt den Frame der aktuell fokussierten Komponente zurück.

Tipp Die Komponente, die im aktuellen Frame steckt, kann ein Dokument sein, muss es aber nicht.

Der Code im Listing 270 zeigt eines der vielen Dinge, die Sie mit dem aktuellen Frame anstellen können. Das Makro holt sich den aktuellen Frame, der Frame gibt das aktuelle Fenster zurück, und dann reduziert das Makro die Größe des aktuellen Fensters.

Listing 270. *Verkleinert das aktuelle Fenster um 25%*

```
REM Verkleinert das aktuelle Fenster auf 75% der aktuellen Größe.
Sub ShrinkWindowBy75
  Dim vFrame     'Aktueller Frame
  Dim vWindow    'Container-Fenster

  REM Nun kommt ein Struct für ein rechteckiges Objekt.
  REM Ich hätte für die Rückgabe auch Typ Variant oder Object
  REM wählen können, aber ich kenne ja den Rückgabetyp, also
  REM habe ich dafür auch den korrekten Variablentyp gewählt.
  Dim vRect As New com.sun.star.awt.Rectangle

  vFrame = StarDesktop.getCurrentFrame()
  vWindow = vFrame.getContainerWindow()

  REM Als Struct wird das Objekt mit seinem Wert kopiert, nicht als Referenz.
  REM Das heißt, dass ich vRect ändern kann, es wird nicht
  REM die Position und die Größe des Fensters verändern.
  REM In meinen Tests sind vRect.X und vRect.Y gleich null, was falsch ist.
  REM vRect.Width und vRect.Height sind aber korrekt.
  vRect = vWindow.getPosSize()

  REM Beim Ändern der Position und der Größe bestimmt das letzte Argument,
  REM welches der Argumente zu nutzen ist.
  'com.sun.star.awt.PosSize.X        Setzt nur die X-Position
  'com.sun.star.awt.PosSize.Y        Setzt nur die Y-Position
  'com.sun.star.awt.PosSize.WIDTH    Setzt nur die Breite
  'com.sun.star.awt.PosSize.HEIGHT   Setzt nur die Höhe
```

```
    'com.sun.star.awt.PosSize.POS      Setzt nur die Position
    'com.sun.star.awt.PosSize.SIZE     Setzt nur die Größe
    'com.sun.star.awt.PosSize.POSSIZE  Setzt sowohl die Position als auch die Größe
    vWindow.setPosSize(vRect.X, vRect.Y, 3 * vRect.Width / 4, 3 * vRect.Height / 4, _
          com.sun.star.awt.PosSize.SIZE)
End Sub
```

12.4. Ein Dokument öffnen

Der Desktop bindet das Interface com.sun.star.frame.XComponentLoader ein. Es ist ein einfaches Interface zum Öffnen von Komponenten über ihren URL. Im Interface XComponentLoader ist die Methode loadComponentFromUrl() definiert, mit der Sie ein bestehendes Dokument öffnen oder ein neues Dokument erstellen.

LoadComponentFromUrl() gibt ein Objekt com.sun.star.lang.XComponent zurück. Die Argumente:

1. aURL als String,
2. aTargetFrameName als String,
3. nSearchFlags als Long,
4. aArgs als Array com.sun.star.beans.PropertyValue

Das erste Argument ist der URL des zu ladenden Dokuments. Um ein bestehendes Dokument zu öffnen, müssen Sie dessen URL angeben, nicht das für das Betriebssytem spezifische Format. Falls nötig, konvertieren Sie das Format mit der Funktion ConvertToURL.

```
Print ConvertToURL("c:\temp\file.txt") 'file:///c:/temp/file.txt
```

Der URL kann eine lokale Datei, aber auch eine Datei im Internet sein, s. Listing 271. Ich empfehle Ihnen eine Hochgeschwindigkeits-Internetverbindung, wenn Sie dieses Makro ausführen, denn es lädt ein Dokument mit 500 Seiten. Starten Sie das Makro auch nur, wenn Sie OOo 3.4 oder später haben, denn sonst wird OOo abstürzen, wenn Sie AndrewMacro schließen.

Listing 271. *Dokument wird über HTTP geladen.*

```
Sub LoadMacroDocFromHttp
  Dim noArgs()           'Leeres Array für die Argumente
  Dim vComp              'Die geladene Komponente
  Dim sURL As String     'URL des zu ladenden Dokuments
  sURL = "http://www.pitonyak.org/AndrewMacro.odt"
  vComp = StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, noArgs())
End Sub
```

OOo nutzt besondere URLs, wenn ein neues anstelle eines schon existierenden Dokuments erstellt werden soll.

Tabelle 106. *URLs zum Erstellen neuer Dokumente.*

URL	Dokumenttyp
"private:factory/scalc"	Tabellendokument
"private:factory/swriter"	Textdokument
"private:factory/swriter/web"	HTML-Dokument
"private:factory/swriter/GlobalDocument"	Masterdokument
"private:factory/sdraw"	Zeichnungsdokument
"private:factory/smath"	Formeldokument
"private:factory/simpress"	Präsentationsdokument
"private:factory/schart"	Diagramm
".component:Bibliography/View1"	Bibliografie – Bearbeitung der bibliografischen Einträge

URL	Dokumenttyp
".component:DB/QueryDesign" ".component:DB/TableDesign" ".component:DB/RelationDesign" ".component:DB/DataSourceBrowser" ".component:DB/FormGridView"	Datenbankkomponenten

Das Makro im Listing 272 öffnet fünf neue Dokumente – jedes in einem neuen Fenster.

Listing 272. *Erzeugt neue Dokumente.*

```
Sub LoadEmptyDocuments
  Dim noArgs()           'Leeres Array für die Argumente
  Dim vComp              'Die geladene Komponente
  Dim sURLs()            'URLs der zu ladenden neuen Dokumenttypen
  Dim sURL As String     'URL des zu ladenden Dokuments
  Dim i As Integer
  sURLs = Array("scalc", "swriter", "sdraw", "smath", "simpress")
  For i = LBound(sURLs) To UBound(sURLs)
    sURL = "private:factory/" & sURLs(i)
    vComp = StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, noArgs())
  Next
End Sub
```

Tipp Sie können ein Dokument auch aus einem Frame heraus öffnen.

Wird eine Komponente (Dokument) geladen, wird sie in einen Frame gesetzt: das zweite Argument für loadComponentFromUrl() spezifiziert den Namen des Frames. Wenn es einen Frame mit diesem Namen gibt, nutzt das neu geladene Dokument den existierenden Frame. Der Code im Listing 272 präsentiert den speziellen Framenamen „_blank“ – etwas weiter unten werden die speziellen Framenamen vorgestellt. Framenamen, die nicht „speziell“ sind, benennen existierende Frames. Diese Namen dürfen nicht mit einem Unterstrich beginnen. Wenn Sie einen Framenamen angeben, müssen Sie OOo über die Suchoptionen sagen, wo der Frame zu suchen ist. Die Werte der Tabelle 105 enumerieren gültige Werte für die Frame-Suchoptionen. Folgende Punkte sind zu beachten:

- Wenn ein Dokument in einem vorhandenen Frame geladen wird, bleibt der Name des Frames erhalten. Wenn aber ein neuer Frame erstellt wird, so erhält er den Namen „“ (leerer String).
- Wenn Sie einen nicht existierenden Framenamen angeben, wird das Dokument in einem neuen Frame geöffnet, auch wenn Sie in den Suchoptionen den FrameSearchFlag.CREATE ausschließen.

Achtung In der API (AOO und LO) finden Sie in der Beschreibung der Methode loadComponentFromUrl, dass wenn ein Frame des angegebenen Namens nicht existiert, ein neuer Frame erzeugt wird. Allerdings erhält mit momentan AOO 4.1.6 und LO 6.4.5.2 der neue Frame immer als Namen einen leeren String, und zwar auf Linux, macOS und Windows.

Außerdem wird im Fall einer negativen Suche auch dann ein neuer Frame erstellt, wenn die Suchoption com.sun.star.frame.FrameSearchFlag.CREATE gezielt ausgeschaltet ist.

Seien es Bugs oder Features, man kann damit leben.

Das Interface XFrame stellt unter anderem die Methoden setName(string) und getName() zur Namensverwaltung zur Verfügung.

Das folgende Makro stammt in der Grundstruktur von der OpenOffice.org-Wiki-Seite https://wiki.openoffice.org/wiki/Documentation/BASIC_Guide/StarDesktop (Stand 2010). Es öffnet ein Dokument in einem Frame mit einem bestimmten Namen. Das Makro geht davon aus, dass ein neuer Frame mit dem Suchnamen erzeugt wird, falls der gesuchte nicht gefunden wird. Die darauf folgende Suche nach einem Frame dieses Namens ist nun auf jeden Fall erfolgreich, und wiederum wird ein neues Dokument darin geöffnet. Im Hinblick auf das veränderte Verhalten von OOo habe ich leichte Anpassungen vorgenommen. Der Name des neuen Frames wird explizit vergeben, wenn er ein leerer String sein sollte.

Im Falle, dass Sie mit dem folgenden Makro experimentieren, achten Sie darauf, vorher Ihre Dokumente zu speichern, denn wenn einer der geöffneten Frames der Suche entspricht, wird das neue Dokument ohne weitere Speichernachfrage in eben diesen Frame geladen.

Listing 273. *Öffnet ein Dokument in einem existierenden Frame.*

```
Sub UseAnExistingFrame
  Dim noArgs()          'Leeres Array für die Argumente
  Dim vDoc              'Die geladene Komponente
  Dim sURL As String    'URL des zu ladenden Dokuments
  Dim nSearch As Long   'Suchoptionen
  Dim sFName As String 'Framename
  Dim vFrame            'Dokument-Frame
  Dim s As String       'Ausgabestring

  REM Globale Suche
  nSearch = com.sun.star.frame.FrameSearchFlag.GLOBAL + _
            com.sun.star.frame.FrameSearchFlag.CREATE

  REM Ich könnte dafür auch eine reale Datei öffnen, aber ich weiß nicht, welche
  REM Dateien auf Ihrem Rechner sind, also erstelle ich ein neues Textdokument.
  'sURL = "file:///home/volker/doc1.odt"
  sURL = "private:factory/swriter"

  REM Sucht einen Frame mit dem Namen MeinFrame statt _default.
  REM Falls kein solcher Frame existiert, wird ein neuer erstellt.
  REM Dieser sollte MeinFrame heißen, neuere AOO- und LO-Versionen
  REM vergeben jedoch einen leeren String als Namen.
  sFName = "MeinFrame"
  vFrame = ThisComponent.CurrentController.Frame
  vDoc = vFrame.loadComponentFromUrl(sURL, sFName, nSearch, noArgs())
  If IsNull(vDoc) Then
    Print "Dokument konnte nicht erstellt werden"
    Exit Sub
  End If

  REM Der Name des Frames hat nichts mit dem Titel zu tun!
  REM Korrektur, falls der Name ein leerer String sein sollte.
  vDoc.CurrentController.Frame.Name = sFName
  s = "Ein Dokument wurde erstellt im Frame " & sFName & "." & Chr$(10) & _
      "Wenn Sie 'OK' drücken, erscheint darin ein neues Dokument."
  MsgBox s

  REM Die folgende Suchoption ohne CREATE führt in manchen AOO- und LO-Versionen
  REM dennoch dazu, dass ein neuer Frame erstellt wird, auch wenn die Suche
  REM nach einem benannten Frame erfolglos ist.
  nSearch = com.sun.star.frame.FrameSearchFlag.GLOBAL
  'sURL = "file:///home/volker/doc2.ods"
  sURL = "private:factory/scalc"
```

```
    vDoc = vFrame.loadComponentFromUrl(sURL, sFName, nSearch, noArgs())
    If IsNull(vDoc) Then
      Print "Dokument konnte nicht erstellt werden"
      Exit Sub
    End If
    s = "Ein neues Dokument wurde erstellt im Frame " _
        & vDoc.CurrentController.Frame.Name & "."
    MsgBox s
  End Sub
```

12.4.1. Benannte Argumente

Das letzte Argument zur Methode loadComponentFromUrl() ist ein Struct-Array vom Typ com.sun.star.beans.PropertyValue. Jede Eigenschaft besteht aus einem Namen und einem Wert. Mit diesen Eigenschaften werden benannte Argumente als Direktiven für OOo beim Öffnen des Dokuments übergeben. Tabelle 107 enthält eine kurze Beschreibung der unterstützten benannten Argumente.

Tipp In der Tabelle 107 stehen nur Argumente, die nicht veraltet sind. Auf der API-Website über den Service com.sun.star.document.MediaDescriptor finden Sie eine komplette Liste, veraltete und aktuelle. Manche Eigenschaften werden nicht gezeigt, zum Beispiel Aborted, weil sie während des Ladevorgangs gesetzt werden. Aborted wird zum Beispiel gesetzt, wenn ein falsches Passwort verwendet wurde.

Tabelle 107. *Gültige benannte Argumente zum Öffnen und Speichern von Dokumenten.*

Argument	Beschreibung
AsTemplate	True erzeugt ein neues Dokument ohne Titel, auch wenn es keine Dokumentvorlage ist. Als Standard wird die Dokumentvorlage zur Bearbeitung geladen.
Author	Der aktuelle Autor, wenn das Dokument gespeichert wird, vorausgesetzt, die Komponente kann den Autor der aktuellen Fassung ermitteln.
CharacterSet	Der Zeichensatz für Zeichen, die aus nur einem Byte bestehen.
Comment	Ähnlich wie das Argument Author, setzt aber die Dokumentbeschreibung für die Versionskontrolle.
ComponentData	Erlaubt komponentenspezifische Eigenschaften.
DocumentBaseURL	Die Basis-URL des Dokuments zur Auflösung relativer Verknüpfungen.
DocumentTitle	Der Dokumenttitel.
EncryptionData	Verschlüsselungsinformation zum Sperren/Entsperren von Dokumenten. Leider kann ich kein Beispiel dafür zeigen.
FilterData	Zusätzliche Filtereigenschaften, falls erforderlich.
FilterName	Name des Filters zum Öffnen oder Speichern der Komponente, falls keine OOo-Typen verwendet werden.
FilterOptions	Zusätzliche Filtereigenschaften, falls erforderlich. Diese Werte müssen Strings sein und vom Filter unterstützt werden. Für Optionen, die keine Strings sind, ist FilterData da.
Frame	Der Frame, der das Dokument enthält.
Hidden	False lädt das Dokument im Verborgenen. Tun Sie das nicht, wenn das Dokument nach dem Öffnen sichtbar werden soll.
HierarchicalDocumentName	Der hierarchische Pfad vom obersten Container bis zum eingebetteten Dokument, zum Beispiel sind Datenbankformulare im Datenbankdokument gespeichert.
InputStream	Sie können einen existierenden Input-Stream zum Öffnen eines Dokuments angeben – zum Beispiel, wenn Sie das Dokument im Arbeitsspeicher haben und es nicht erst auf die Platte schreiben wollen.

InteractionHandler	Übergibt einen interaktiven Error-Handler und enthält, falls nötig, ein Passwort.
JumpMark	Sprung nach dem Öffnen der Komponente auf eine markierte Position. Man kann das auch erreichen, wenn man an das Ende des URL, der das zu öffnende Dokument identifiziert, das Zeichen „#“ und die Sprungmarke anschließt. Die „#“-Syntax ist in den meisten URL-Schemata nicht spezifiziert.
LockContentExtraction	True verhindert, dass irgendwelcher Inhalt kopiert oder verschoben werden kann. Die Menübefehle „Kopieren“ und „Ausschneiden“ werden deaktiviert, ebenso das Kopieren einer Auswahl in die Zwischenablage wie auch das Verschieben mit der Maus. Seit LibreOffice 6.4
LockEditDoc	True verhindert, dass im Nur-Lesen-Modus in den Bearbeitungsmodus geschaltet werden kann. Seit LibreOffice 6.4
LockExport	True verhindert den Export des Dokumentinhalts in eine Datei. Die Menübefehle „Exportieren“, „Senden“, „Speichern unter“ werden deaktiviert, ebenso das Exportieren einzelner Grafiken wie auch Serienbriefe. Seit LibreOffice 6.4
LockPrint	True verhindert jegliche Druckfunktion, einschließlich Drucker-Einstellungen. Seit LibreOffice 6.4
LockSave	True deaktiviert die Funktion „Speichern“. Seit LibreOffice 6.4
MacroExecutionMode	Dieser numerische Wert bestimmt, ob Makros beim Öffnen des Dokuments ausgeführt werden, s. Tabelle 108.
MediaType	Der MIME-Typ der zu ladenden Daten.
OpenNewView	True erzwingt, dass für die Komponente ein neues Fenster geöffnet wird, auch wenn das Dokument schon geladen ist. Manche Komponenten unterstützen Mehrfachansichten derselben Daten. Wenn das geöffnete Dokument keine Mehrfachansicht unterstützt, wird ein neues Fenster geöffnet. Das ist dann keine Ansicht, das Dokument wird einfach noch einmal geöffnet.
OutputStream	Wenn dieses Argument beim Speichern eines Dokuments angegeben ist, muss der Stream zum Schreiben der Daten genutzt werden. Wenn kein Stream zur Verfügung steht, wird ein Stream automatisch erzeugt, der die anderen Eigenschaften nutzt.
Overwrite	True überschreibt eine vorhandene Datei mit demselben Namen.
Password	Passwort zum Öffnen oder Speichern eines Dokuments. Wenn ein Dokument, das ein Passwort verlangt, beim Öffnen kein Passwort erhält, wird es nicht geöffnet.
PickListEntry	False verhindert, dass das geöffnete Dokument in die Liste der zuletzt verwendeten Dokumente aufgenommen wird. Standardwert: True. Seit LibreOffice 5.1
PostData	Sendet Daten an eine HTTP-Adresse und lädt die Antwort als Dokument. PostData verwendet man normalerweise, um die Rückgabe eines Webformulars einer Website zu übernehmen.
Preview	True bestimmt, dass das Dokument in der Seitenansicht geöffnet wird. Es gibt einige Optimierungen beim Öffnen eines Dokuments im Seitenansichtsmodus.
ReadOnly	Öffnet das Dokument schreibgeschützt. Schreibgeschützte Dokumente können über die Benutzeroberfläche nicht geändert werden, wohl aber mittels der API (also mit einem Makro).
Referer	Ein URL, der auf den Referenzierer weist, der das Dokument geöffnet hat. Ohne Referenzierer wird ein Dokument, das Sicherheitskontrollen verlangt, zurückgewiesen. (Ja, der Name des Arguments ist „Referer“, obwohl es englisch korrekt „Referrer“ heißt).
RepairPackage	Dokumente werden in einem komprimierten ZIP-Format gespeichert. Der Wert True führt dazu, dass der Versuch gemacht wird, von einer beschädigten ZIP-Datei Informationen wiederherzustellen.

StartPresentation	Wenn das Dokument eine Präsentation ist, wird sie umgehend gestartet.
StatusIndicator	Ein Objekt, das als Fortschrittsanzeige beim Öffnen oder Speichern eines Dokuments zu verwenden ist.
SuggestedSaveAsDir	URL des Verzeichnisses, das beim nächsten „Speichern unter“-Dialog verwendet wird.
SuggestedSaveAsName	Der Dateinamensvorschlag, der beim nächsten „Speichern unter“-Dialog verwendet wird.
TemplateName	Der Name der Dokumentvorlage statt des URL. TemplateRegionName darf nicht fehlen.
TemplateRegionName	Pfad zur Dokumentvorlage statt des URL. TemplateName darf nicht fehlen.
Unpacked	OOo speichert Dokumente in einem gezippten Format. Der Wert True speichert die Datei in einem Verzeichnis, falls das für die Komponente unterstützt wird. Das hat seit langem nicht funktioniert, bis hin zu meinem letzten Test mit LO 6.0.4.2.
UpdateDocMode	Der numerische Wert legt fest, wie das Dokument aktualisiert wird. Weitere Informationen auf der API-Seite der Konstantengruppe com.sun.star.document.UpdateDocMode.
URL	Vollständiger URL des zu öffnenden Dokuments, einschließlich der Sprungadresse, falls erforderlich.
Version	Wenn die Komponente die Versionskontrolle unterstützt, gibt dieses Argument die Version an, die zu öffnen oder zu speichern ist. Wenn keine Version angegeben ist, wird das Hauptdokument geöffnet oder gespeichert.
ViewControllerName	Der Name des Ansichts-Controllers (View Controller), der beim Laden eines Dokuments in einen Frame erzeugt werden soll. Das heißt, dass die Eigenschaft an die Methode createViewController des Dokuments weitergereicht wird. Wenn das geöffnete Dokument nicht das Interface XModel2 unterstützt, wird die Eigenschaft ignoriert.
ViewData	Der Wert dieses Arguments ist komponentenspezifisch und wird normalerweise vom Controller des Frames bereitgestellt.
ViewId	Manche Komponenten unterstützen verschiedene Ansichten derselben Daten. Dieser Wert steht für die Ansicht, die nach dem Öffnen genutzt wird. Der Standardwert null gilt für die Standardansicht.

12.4.2. Eine Dokumentvorlage öffnen

Beim Öffnen eines Dokuments mit loadComponentFromUrl() behandelt die API alle Dokumente gleich. Sie können eine OOo-Dokumentvorlage zur Bearbeitung öffnen (und sie danach als neue Dokumentvorlage speichern) oder Sie können eine Dokumentvorlage als Grundlage für ein neues Dokument nehmen. Das benannte Argument AsTemplate weist OOo an, das angegebene Dokument als Vorlage zu verwenden und nicht als zu bearbeitendes Dokument. Das Makro im Listing 274 öffnet ein Dokument als Mustervorlage.

Listing 274. *Öffnet ein Dokument auf der Grundlage einer Dokumentvorlage.*

```
Sub UseTemplate
  REM Dies ist ein Array mit nur einem Element (0 To 0) als Typ PropertyValue
  Dim args(0) As New com.sun.star.beans.PropertyValue
  Dim sURL As String  'URL des zu öffnenden Dokuments

  sURL = "file:///home/andy/doc1.ott"
  args(0).Name  = "AsTemplate"
  args(0).Value = True
  StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, args())
End Sub
```

Die Methode loadComponentFromUrl() geht davon aus, dass der Speicherort des Dokuments durch einen URL gekennzeichnet ist, also nicht durch eine betriebssystemspezifische Form. Obwohl ich das Makro im Listing 274 auf meinem Linux-Rechner getestet habe, wird es auch auf einem Windows-Rechner funktionieren. Hier müssen Sie jedoch noch den Laufwerksbuchstaben ergänzen:

```
sURL = "file://c:/home/andy/doc1.ott"
```

Tipp Mit den Funktionen ConvertFromURL und ConvertToURL konvertieren Sie zwischen der systemspezifischen und der URL-Notation.

12.4.3. Makros beim Öffnen eines Dokuments freigeben

Wenn ein Dokument, das ein Makro enthält, aus der Benutzerumgebung heraus geöffnet wird, erscheint eine Sicherheitsabfrage, ob die Makros erlaubt sein sollen. Wenn ein Dokument aus einem Makro heraus mittels der API geöffnet wird, sind Makros in dem Dokument deaktiviert. Manche Makros laufen aufgrund von Ereignissen, die im Dokument geschehen. Wenn Makros nun deaktiviert sind, kann man die enthaltenen Makros dennoch manuell aufrufen, jedoch werden Makros, die von einem Ereignis des Dokuments gestartet werden, niemals ausgeführt.

Das benannte Argument MacroExecutionMode weist OOo an, wie Makros zu behandeln sind, wenn ein Dokument geöffnet wird. In der Tabelle 108 finden Sie die gültigen Werte des Arguments MacroExecutionMode.

Tabelle 108. *Die Konstantengruppe com.sun.star.document.MacroExecMode.*

#	Name	Beschreibung
0	NEVER_EXECUTE	Überhaupt keine Makroausführung.
1	FROM_LIST	Stillschweigende Ausführung von Makros, die in einer Sicherheitsliste stehen.
2	ALWAYS_EXECUTE	Ausführung eines jeden Makros. Solche mit Sicherheitszertifikat und solche, die in einer Sicherheitsliste stehen, werden stillschweigend ausgeführt.
3	USE_CONFIG	Beachtung der Konfiguration der Makrosicherheitseinstellungen. Wenn der Nutzer gefragt werden soll, erscheint der Abfragedialog.
4	ALWAYS_EXECUTE_NO_WARN	Makros sollten immer ohne weiteren Hinweis ausgeführt werden.
5	USE_CONFIG_REJECT_CONFIRMATION	Beachtung der Konfiguration der Makrosicherheitseinstellungen. Im Falle, dass der Nutzer gefragt werden soll, wird das Makro zurückgewiesen.
6	USE_CONFIG_APPROVE_CONFIRMATION	Beachtung der Konfiguration der Makrosicherheitseinstellungen. Im Falle, dass der Nutzer gefragt werden soll, wird das Makro ausgeführt.
7	FROM_LIST_NO_WARN	Nur Makros, die in einer Sicherheitsliste stehen, werden ausgeführt, alle anderen nicht.
8	FROM_LIST_AND_SIGNED_WARN	Es werden Makros ausgeführt, die in einer Sicherheitsliste stehen oder die ein Sicherheitszertifikat haben.
9	FROM_LIST_AND_SIGNED_NO_WARN	Es werden Makros ausgeführt, die in einer Sicherheitsliste stehen oder die ein Sicherheitszertifikat haben. Es werden keine Warnungen oder Hinweise ausgegeben.

Das Makro im Listing 275 öffnet ein Dokument als Mustervorlage und erlaubt die Ausführung von Makros während des Ladevorgangs. In diesem Makro sehen Sie auch, dass mehrere benannte Argumente gleichzeitig angegeben werden können.

Listing 275. *Öffnet ein Dokument als Mustervorlage und gibt enthaltene Makros frei.*

```
Sub UseTemplateRunMacro
  REM Dies ist ein Array mit zwei Elementen (0 To 1) als Typ PropertyValue
  Dim args(1) As New com.sun.star.beans.PropertyValue
  Dim sURL As String  'URL des zu öffnenden Dokuments

  Print com.sun.star.document.MacroExecMode.USE_CONFIG
  sURL = "file:///home/andy/doc1.odt"
  args(0).Name  = "AsTemplate"
  args(0).Value = True
  args(1).Name  = "MacroExecutionMode"
  args(1).Value = com.sun.star.document.MacroExecMode.ALWAYS_EXECUTE_NO_WARN
  StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, args())
End Sub
```

12.4.4. Importieren und exportieren

OOo hat einen Mechanismus, der den Typ eines Dokuments beim Öffnen ermittelt. Dieser Mechanismus ist für die begrenzte Zahl der Dokumenttypen verlässlich, mit denen ich täglich umgehe. Manchmal muss man jedoch den Filternamen angegeben, wenn ein Dokument importiert wird. Zum Export eines Dokuments muss man allerdings den Filtertyp immer spezifizieren. Der Code im Listing 276 öffnet ein MS-Word-Dokument. Meine Tests haben ergeben, dass die Dokumente immer korrekt geöffnet wurden, auch wenn der Filtername fehlte. Denn OOo kann sehr gut raten.

Listing 276. *Angabe des Filternamens beim Öffnen eines Dokuments.*

```
Sub LoadDocFile
  Dim noArgs(0) As New com.sun.star.beans.PropertyValue
  Dim sURL As String
  noArgs(0).Name = "FilterName"
  noArgs(0).Value = "Microsoft Word 97/2000/XP"
  sURL = "file:///home/andy/one.doc"
  StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, noArgs())
End Sub
```

12.4.5. Namen der Import- und Exportfiler

Regelmäßig werden neue Import- und Exportfilter zu OOo hinzugefügt. So lohnt es sich nicht, hier eine Liste der unterstützten Filter aufzuführen, wenn ein Makro die aktuelle Liste erstellen kann. Das folgende Makro erstellt ein neues Tabellendokument und listet darin die unterstützten Filter auf. Zur Formatierung der Daten wird ein aufgezeichnetes Makro eingesetzt.

UIName ist der allgemeine oder der gemäß dem aktuellen Gebietsschema lokalisierte Name. Die Spalte Name enthält den internen Namen, den man verwenden muss, wenn man einen Filternamen für den Import oder Export angibt. Die Spalten Import und Export informieren darüber, ob der Filter für Import oder Export zur Verfügung steht. Die Spalte Optionen zeigt, ob dem Filter Steuerungsoptionen beigegeben werden können.

Listing 277. *Listet die unterstützten Filter in einem Tabellenblatt auf.*

```
Sub FiltersToCalc
  Dim oFF           ' Der Service FilterFactory
  Dim oFilterNames  ' Array der Filter-Namen
  Dim oDoc          ' Neu erstelltes Dokument zur Ausgabe der Filternamen
  Dim oSheet        ' Tabellenblatt, das die Filterinformationen enthalten wird
  Dim oCell         ' Wechselnde Ausgabezelle
  Dim i%            ' Indexvariable
```

```
Dim fProps()       ' Die Eigenschaften jedes einzelnen Filters
Dim fp%            ' Index der Filtereigenschaften
Dim fpCol%         ' Spalte für die jeweiligen Filterdaten
Dim s$             ' Arbeitsablage für manche Eigenschaftswerte
Dim iUserData%     ' Index für Arrays von Eigenschaftswerten
Dim oProp          ' Einzelne Eigenschaft
Dim oArrayData     ' Array mit User-Eigenschaften
Dim sPropNames(7) As String
Dim flags As Long
sPropNames(0) = "Name"
sPropNames(1) = "Import"
sPropNames(2) = "Export"
sPropNames(3) = "Options"
sPropNames(4) = "Visible"
sPropNames(5) = "Flags"
sPropNames(6) = "DocumentService"
sPropNames(7) = "UIName"

oFF = CreateUnoService("com.sun.star.document.FilterFactory")
oFilterNames = oFF.getElementNames()

' Erzeugt ein Tabellendokument zur Aufnahme der Filternamen.
oDoc = StarDesktop.loadComponentFromURL( _
                   "private:factory/scalc", "_blank", 0, Array())
oSheet = oDoc.getSheets().getByIndex(0)

' Schreibt die Filternamen in das Tabellendokument.
For i = LBound(oFilterNames) To UBound(oFilterNames)
  fProps() = oFF.getByName(oFilterNames(i))
  For fp = LBound(fProps) To UBound(fProps)
    oProp = fProps(fp)
    fpCol = FindInArrayAdd(oProp.Name, sPropNames)
    If fpCol > UBound(sPropNames) Then
      ReDim Preserve sPropNames(fpCol) As String
      sPropNames(fpCol) = oProp.Name
    End If

    oCell = oSheet.getCellByPosition(fpCol, i + 1)
    If Not IsArray(oProp.Value) Then
      oCell.setString(oProp.Value)
      If oProp.Name = "Flags" Then
        flags = oProp.Value
        oCell.setString(CStr(flags) & " = " & Flags_int2str(flags)
        If flags And 1 Then
          oCell = oSheet.getCellByPosition(FindInArrayAdd("Import", sPropNames), _
                                           i + 1)
          oCell.setString("X")
        End If
        If flags And 2 Then
          oCell = oSheet.getCellByPosition(FindInArrayAdd("Export", sPropNames), _
                                           i + 1)
          oCell.setString("X")
        End If
        If flags And 128 Then
          oCell = oSheet.getCellByPosition(FindInArrayAdd("Options", sPropNames), _
                                           i + 1)
          oCell.setString("X")
        End If
```

```
            If ((flags And 4096) Or (flags And 8192)) Then
              oCell = oSheet.getCellByPosition(FindInArrayAdd("Visible", sPropNames), _
                                               i + 1)
              oCell.setString("X")
            End If
          End If
        ElseIf LBound(oProp.Value) <= UBound(oProp.Value) Then
          s = ""
          oArrayData = oProp.Value
          For iUserData = LBound(oArrayData) To UBound(oArrayData)
            If VarType(oArrayData(iUserData)) = 8 Then
              s = s & "(String:" & oArrayData(iUserData) & ")"
            Else
              s = s & "(" & oArrayData(iUserData).Name & ":" & _
                  oArrayData(iUserData).Value & ")"
            End If
          Next
          oCell.setString(s)
        End If
      Next
    Next
    For fp = LBound(sPropNames) To UBound(sPropNames)
      oCell = oSheet.getCellByPosition(fp, 0)
      oCell.setString(sPropnames(fp))
    Next
    FinalFormat(oDoc)
End Sub

Function Flags_int2str(flags As Long) As String
    Dim cFlags As Long
    Dim strVal As String
    Dim curFlag As Long

    cFlags = flags
    strVal = ""

    If cFlags And &H00000001 Then
        strVal = strVal & " Import"
        cFlags = cFlags - 1
    End If

    If cFlags And &H00000002 Then
        strVal = strVal & " Export"
        cFlags = cFlags - 2
    End If

    If cFlags And &H00000004 Then
        strVal = strVal & " Template"
        cFlags = cFlags - 4
    End If

    If cFlags And &H00000008 Then
        strVal = strVal & " Internal"
        cFlags = cFlags - 8
    End If
```

```
If cFlags And &H00000010 Then
    strVal = strVal & " TemplatePath"
    cFlags = cFlags - 16
End If

If cFlags And &H00000020 Then
    strVal = strVal & " Own"
    cFlags = cFlags - 32
End If

If cFlags And &H00000040 Then
    strVal = strVal & " Alien"
    cFlags = cFlags - 64
End If

If cFlags And &H00000080 Then
    strVal = strVal & " UseOptions"
    cFlags = cFlags - 128
End If

If cFlags And &H00000100 Then
    strVal = strVal & " Default"
    cFlags = cFlags - 256
End If

If cFlags And &H00000200 Then
    strVal = strVal & " Executable"
    cFlags = cFlags - 512
End If

If cFlags And &H00000400 Then
    strVal = strVal & " SupportSelection"
    cFlags = cFlags - 1024
End If

If cFlags And &H00000800 Then
    strVal = strVal & " MapToAppPlug"
    cFlags = cFlags - 2048
End If

If cFlags And &H00001000 Then
    strVal = strVal & " NotInFileDialog"
    cFlags = cFlags - 4096
End If

If cFlags And &H00002000 Then
    strVal = strVal & " NotInChooser"
    cFlags = cFlags - 8192
End If

If cFlags And &H00004000 Then
    strVal = strVal & " Acynchronas"
    cFlags = cFlags - 16384
End If

If cFlags And &H00008000 Then
    strVal = strVal & " Creator"
    cFlags = cFlags - 32768
```

```
End If

If cFlags And &H00010000 Then
    strVal = strVal & " Readonly"
    cFlags = cFlags - 65536
End If

If cFlags And &H00020000 Then
    strVal = strVal & " NotInstalled"
    cFlags = cFlags - 131072
End If

If cFlags And &H00040000 Then
    strVal = strVal & " ConsultService"
    cFlags = cFlags - 262144
End If

If cFlags And &H00080000 Then
    strVal = strVal & " 3rdPartyFilter"
    cFlags = cFlags - 524288
End If

If cFlags And &H00100000 Then
    strVal = strVal & " Packed"
    cFlags = cFlags - 1048576
End If

If cFlags And &H00200000 Then
    strVal = strVal & " SilentExport"
    cFlags = cFlags - 2097152
End If

If cFlags And &H00400000 Then
    strVal = strVal & " BrowserPreferred"
    cFlags = cFlags - 4194304
End If

If cFlags And &H00800000 Then
    strVal = strVal & " [H00800000]"
    cFlags = cFlags - 8388608
End If

If cFlags And &H01000000 Then
    strVal = strVal & " [H01000000]"
    cFlags = cFlags - 16777216
End If

If cFlags And &H02000000 Then
    strVal = strVal & " [H02000000]"
    cFlags = cFlags - 33554432
End If

If cFlags And &H10000000 Then
    strVal = strVal & " Preferred"
    cFlags = cFlags - 268435456
End If
```

```
    If cFlags <> 0 Then
        strVal = strVal & _
                 " !!! ACHTUNG: nicht unterstütztes Flag [" & cFlags & "] entdeckt !!!"
    End If

    Flags_int2str = strVal
End Function

Function FindInArrayAdd(sName$, nameArray) As Integer
  Dim i As Integer
  For i = LBound(nameArray()) To UBound(nameArray())
    If nameArray(i) = sName Then
       Exit For
    End If
  Next
  FindInArrayAdd = i
End Function

Sub FinalFormat(oDoc)
rem ----------------------------------------------------------------------
rem define variables
Dim document   As Object
Dim dispatcher As Object
rem ----------------------------------------------------------------------
rem get access to the document
document   = oDoc.CurrentController.Frame
dispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")

rem ----------------------------------------------------------------------
Dim args1(0) As New com.sun.star.beans.PropertyValue
args1(0).Name = "ToPoint"
args1(0).Value = "$A$1"

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

rem ----------------------------------------------------------------------
Dim args2(1) As New com.sun.star.beans.PropertyValue
args2(0).Name = "By"
args2(0).Value = 1
args2(1).Name = "Sel"
args2(1).Value = True

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

rem ----------------------------------------------------------------------
Dim args3(2) As New com.sun.star.beans.PropertyValue
args3(0).Name = "FontHeight.Height"
args3(0).Value = 12
args3(1).Name = "FontHeight.Prop"
args3(1).Value = 100
args3(2).Name = "FontHeight.Diff"
args3(2).Value = 0

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

rem ----------------------------------------------------------------------
Dim args4(0) As New com.sun.star.beans.PropertyValue
args4(0).Name = "Bold"
```

```
args4(0).Value = True

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

rem ----------------------------------------------------------------------
Dim args5(0) As New com.sun.star.beans.PropertyValue
args5(0).Name = "HorizontalAlignment"
args5(0).Value = com.sun.star.table.CellHoriJustify.CENTER

dispatcher.executeDispatch(document, ".uno:HorizontalAlignment", "", 0, args5())

rem ----------------------------------------------------------------------
Dim args6(1) As New com.sun.star.beans.PropertyValue
args6(0).Name = "By"
args6(0).Value = 1
args6(1).Name = "Sel"
args6(1).Value = True

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

rem ----------------------------------------------------------------------
Dim args7(0) As New com.sun.star.beans.PropertyValue
args7(0).Name = "aExtraWidth"
args7(0).Value = 254

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

rem ----------------------------------------------------------------------
Dim args8(8) As New com.sun.star.beans.PropertyValue
args8(0).Name = "ByRows"
args8(0).Value = True
args8(1).Name = "HasHeader"
args8(1).Value = True
args8(2).Name = "CaseSensitive"
args8(2).Value = False
args8(3).Name = "IncludeAttribs"
args8(3).Value = True
args8(4).Name = "UserDefIndex"
args8(4).Value = 0
args8(5).Name = "Col1"
args8(5).Value = 7
args8(6).Name = "Ascending1"
args8(6).Value = True
args8(7).Name = "Col2"
args8(7).Value = 1
args8(8).Name = "Ascending2"
args8(8).Value = True

dispatcher.executeDispatch(document, ".uno:DataSort", "", 0, args8())
dispatcher.executeDispatch(document, ".uno:GoToCell", "", 0, args1())
End Sub
```

12.4.6. Dokumente laden und speichern

Ein Dokument wird über den Desktop oder einen Frame geladen. Zum Speichern eines Dokuments nehmen Sie die Methode storeToURL(), die im Dokumentobjekt eingebunden ist. Das erste Argu-

ment dieser Methode ist der URL als Speicheradresse des Dokuments, das zweite Argument ein Array benannter Argumente, s. Tabelle 107. Um ein Dokument zu einem anderen Typ zu exportieren, müssen Sie den entsprechenden Export-Filternamen angeben. Der Code im Listing 278 exportiert ein Textdokument in eine PDF-Datei.

Listing 278. *Exportiert das aktuelle Dokument zu PDF (setzt ein Textdokument voraus).*

```
Dim args(0) As New com.sun.star.beans.PropertyValue
args(0).Name = "FilterName"
args(0).Value = "writer_pdf_Export"
ThisComponent.storeToURL("file:///test.pdf", args())
```

Tipp Bei Filternamen muss die Groß-/Kleinschreibung beachtet werden.

Manche Import- und Exportfilter für Calc unterstützen die Angabe von Optionen: zum Beispiel nutzen die Filter DIF, dBase und Lotus alle dieselben Filteroptionen, nämlich einen numerischen Index für den Zeichensatz bei 1-Byte-Zeichen.

Listing 279. *Die Filter DIF, dBase und Lotus nutzen alle dieselben Filteroptionen.*

```
Dim args(1) As New com.sun.star.beans.PropertyValue
args(0).Name = "FilterName"
args(0).Value = "dBase"
args(1).Name = "FilterOptions"  'Zeichensatz für 1-Byte-Zeichen
args(1).Value = 0               'Zeichensatz des Systems
```

Die Beschreibung des CSV-Filters legt zugrunde, dass Sie mit dem CSV-Dateiformat vertraut sind. Wenn Sie es nicht sind, können Sie gerne diesen Abschnitt überspringen. Der Text-CSV-Filter nutzt im Gegensatz zum dBase-Filter einen komplizierten String mit fünf Musterelementen (Tokens). Jeder Token kann aus mehreren Werten bestehen, die durch einen Schrägstrich (/) getrennt werden. Die Tokens wiederum werden durch ein Komma getrennt.

Listing 280. *Die Optionen für den CSV-Filter sind kompliziert.*

```
Dim args(1) As New com.sun.star.beans.PropertyValue
args(0).Name = "FilterName"
args(0).Value = "scalc: Text - txt - csv (StarCalc)"
args(1).Name = "FilterOptions"
args(1).Value = "44,34,0,1,1/5/2/1/3/1/4/1"
```

- Der erste Token enthält die Feldtrenner als ASCII-Werte. Das Komma zum Beispiel hat den ASCII-Wert 44, s. Listing 280. Als Information, dass manche Felder durch ein Leerzeichen (ASCII 32) und andere durch einen Tabulator (ASCII 9) getrennt werden, würde Listing 280 als Filteroptionen „32/9,34,0,1,1/5/2/1/3/1/4/1“ verwenden.
- In einer CSV-Datei werden die Textteile gewöhnlich mit doppelten Anführungszeichen (") oder einfachen Anführungszeichen (') begrenzt. Im Listing 280 steht, dass Textteile durch doppelte Anführungszeichen (ASCII 34) identifiziert sind. Wenn mehrere Texttrenner verwendet werden sollen, müssen sie mit einem Schrägstrich getrennt werden.
- Der dritte Token bestimmt den zu verwendenden Zeichensatz. Es ist derselbe Wert, der auch in den Filtern DIF, dBase und Lotus angewendet wird. Im Code des Listing 280 ist es die Zeichensatzkennung Null.
- Der vierte Token bestimmt die erste Importzeile – normalerweise Zeile eins. Listing 280 legt fest, dass der Import bei Zeile eins beginnt.
- Der letzte Token beschreibt das Format jeder Spalte der CSV-Datei. Die Spalten können entweder als durch Trenner definierter Text (s. Listing 281) oder als Text fester Länge formatiert sein, s. Listing 282.

Listing 281. *Durch Trenner definierter Spaltentext für den CSV-Filter.*

```
<field_num>/<format>/<field_num>/<format>/<field_num>/<format>/...
```

<field_num> ist eine Ganzzahl als Feldzähler: 1 ist das Feld ganz links. Gehen wir einmal von den Feldern „eins“, „zwei“ und „drei“ aus, dann bezieht sich die 2 als field_num auf das Feld „zwei“. <format> ist eine Ganzzahl für das Feldformat (s. Tabelle 109). Der Code im Listing 280 legt das erste Feld als Datum im Format YY/MM/DD (5) fest, die anderen drei als Standardformat (1).

Tabelle 109. *Formatwerte für CSV-Felder.*

Format	Beschreibung
1	Standard
2	Text
3	MM/DD/YY
4	DD/MM/YY
5	YY/MM/DD
9	Das Feld nicht importieren, sondern ignorieren!
10	Importiert eine Zahl im US-englischen Format, ohne Beachtung des aktuellen Gebietsschemas.

Listing 282. *Strings fester Spaltenbreite für den CSV-Filter.*

```
FIX/<start>/<format>/<start>/<format>/<start>/<format>/...
```

Um dem Filter zu sagen, dass eine Datei mit fester Spaltenbreite daherkommt, setzen Sie dem letzten Token den String „FIX“ voran. In CSV-Dateien mit fester Spaltenbreite ist abzuzählen, an welchem Zeichen eine Spalte beginnt, s. Listing 282. Der Wert <start> bezieht sich auf das erste Zeichen eines Felds. Ein Startwert von 0 ist also das Textzeichen ganz links. Der Wert <format> ist eine Ganzzahl für das Textformat, s. Tabelle 109.

12.4.7. Fehlerbehandlung während des Ladens eines Dokuments

Während des Öffnens eines Dokuments wird kein Laufzeitfehler erzeugt. Fehler werden nur angezeigt, wenn ein interaktiver Handler als benanntes Argument angegeben ist. Leider kann man mit Basic keinen interaktiven Handler einsetzen. Der OOo Developer’s Guide zeigt Beispiele für Error-Handlers in anderen Sprachen. Deutlich gesagt, mit Basic können Sie keine Fehler während der Ladephase eines Dokuments abfangen. Das Dokument wird einfach nicht geladen und es wird Null zurückgegeben.

Die grafische Oberfläche (GUI) stellt einen Error-Handler bereit, der mit dem Nutzer interagiert. Im Falle eines Fehlers zeigt er eine Meldung an und fordert den Nutzer, falls nötig, zur Eingabe eines Passworts auf. Wenn als benanntes Argument kein Error-Handler angegeben ist, wird der Standard-Handler verwendet, der einfach die meisten Fehler ignoriert und dem Nutzer wenig an Informationen bietet.

12.5. Fazit

Der Desktop wirkt als die Hauptanwendung, die OOo steuert. Wenn Sie also auf etwas zugreifen müssen, das sich global auf Dokumente und Frames bezieht, denken Sie an den Desktop. Die globalen Variablen StarDesktop und ThisComponent bieten einen einfachen Zugriff auf das Desktop-Objekt beziehungsweise auf das aktuelle Dokument. Dieses Kapitel hat Techniken gezeigt, wie man auf Container mit mehreren Objekten zugreift. Machen Sie sich mit dem Grundpotenzial zum Öffnen von Dokumenten vertraut, mit dem Import und Export verschiedener Dateitypen und mit den Möglichkeiten und Beschränkungen von OOo. Somit werden Sie in der Lage sein, eine große Breite an Dateitypen in der OOo-Umgebung zu verwenden und zu erzeugen.

13. Allgemeine Dokument-Methoden

OOo umfasst sechs primäre Dokumenttypen: Textdokument, Tabellendokument, Zeichnung, Formel, Datenbank und Präsentation. Trotz ihrer Unterschiede gibt es Funktionalitäten und Interfaces, die für alle Dokumenttypen gleich sind. Dazu gehören der Zugriff auf das Dokument-Modell, das Drucken und das Speichern. In diesem Kapitel werden die allen Dokumenttypen gemeinsamen Dienste vorgestellt.

Jedes Dokument enthält Daten, die geändert und gedruckt werden können. Die wesentlichen Daten in einem Writer-Dokument bestehen zum Beispiel aus Text, es können aber auch Tabellen und Grafiken sein. Das Daten-Modell besteht aus diesen zugrunde liegenden Daten, die unabhängig von ihrer Darstellung änderbar sind. Wenn man diese Daten mit einem Makro ändert, so ändert man sie direkt im Daten-Modell. Auch wenn man mit Hilfe des Dispatchers die Daten indirekt verarbeiten kann, so werden die Änderungen doch direkt im Modell vorgenommen. Der Dispatcher wird häufig zum Einfügen der Zwischenablage in das Dokument verwendet (s. Kapitel 11. Der Dispatcher).

Das Daten-Modell enthält ein Controller-Objekt, das die visuelle Darstellung der Daten vornimmt. Der Controller ändert die Daten nicht, er kontrolliert nur, wie die Daten ausgegeben werden. Der Controller greift direkt auf die Benutzerschnittstelle zu, um festzulegen, wo sich der visuelle Cursor befinden, welche Seite dargestellt wird, und um Teile des Dokuments zu markieren. Mit dem Controller werden anzeigebezogene Informationen erfasst wie die aktuelle Seite oder der aktuell markierte Text.

Tipp In OOo gibt es einen Modus namens „headless“, der kein Startfenster kennt, kein Standarddokument, keine Benutzerschnittstelle und auch keine Benutzereingaben. (Mit der Kommandozeileneingabe „soffice -?“ erhalten Sie eine Liste der unterstützten Modi.) Im Headless-Modus gibt es keine Anzeigekomponente. Daher wird auch kein Controller benötigt, und es kann auch sein, dass er gar nicht existiert. Wenn Ihr Makro auch im Headless-Modus laufen könnte, müssen Sie erst prüfen, ob der Controller Null ist, bevor er genutzt wird.

13.1. Service-Manager

OOo hat einen allgemeinen globalen Service-Manager, mit dem man Instanzen allgemeiner UNO-Services erzeugt und referenziert. Ein Service-Manager erhält einen String, der einen Objekttyp repräsentiert, und gibt eine Instanz dieses Objekts zurück. Der globale Service-Manager wird in Basic über die Funktion CreateUnoService(String) aufgerufen. Er gibt solche allgemeinen Objekte zurück wie den Dispatch-Helper oder ein Objekt für den Simple File Access.

Listing 283. *Anwendung des globalen Service-Managers.*

```
oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")
oSimpleFileAccess = CreateUnoService("com.sun.star.ucb.SimpleFileAccess")
```

Auch Dokumente haben einen Service-Manager, um Objekte zu erzeugen, die Bestandteile der Dokumente sind (oder sich direkt darauf beziehen). Zum Beispiel kann ein Writer-Dokument eine Texttabelle oder ein Textfeld erzeugen, die in das Dokument eingefügt werden können. Ganz allgemein können Objekte, die nicht von einem Dokument erzeugt wurden, nicht in das Dokument eingefügt werden. Ein Dokument ist nicht in der Lage, globale Objekte zurückzugeben, und der globale Service-Manager ist nicht in der Lage, Objekte zurückzugeben, die dann in ein Dokument eingefügt werden.

Listing 284. *Ein Dokument als Service-Manager.*

```
REM Das Dokument erzeugt die Texttabelle.
oTable = oDoc.createInstance("com.sun.star.text.TextTable")
oTable.initialize(3, 2) 'Drei Zeilen, zwei Spalten
REM Nun wird die Texttabelle am Dokumentende eingefügt.
oDoc.Text.insertTextContent(oDoc.Text.getEnd(), oTable, False)
```

13.2. Services und Interfaces

Stellen Sie sich eine Komponente als Fenster vor, das dem Desktop gehört. Mit dem Desktop-Objekt listen Sie die Komponenten auf. Das sind die Basic-IDE und das Hilfefenster gemeinsam mit den aktuellen Dokumenten. Wenn eine Komponente das Interface com.sun.star.frame.XModel unterstützt, ist sie ein Dokument (und nicht die Basic-IDE und auch nicht das Hilfefenster). Jedes Dokument unterstützt das Interface XModel, und jeder Dokumenttyp unterstützt einen ganz speziellen eigenen Service (s. Tabelle 93). Testen Sie zuerst mit HasUnoInterfaces, ob das Objekt das Interface XModel unterstützt, und bestimmen Sie dann den Dokumenttyp mit der Methode supportsService (s. Listing 219). Im Abschnitt 10.4. Inspizierung von Universal Network Objects haben Sie die Zusammenhänge schon kennengelernt.

Tipp Die Methode supportsService() wird vom Interface com.sun.star.lang.XServiceInfo definiert. Dieses Interface definiert auch die Methode getImplementationName(), die einen den Objekttyp eindeutig beschreibenden String zurückgibt.

Zahlreiche Interfaces werden von vielen (wenn nicht sogar allen) Dokumenttypen genutzt. Betrachten Sie die (unvollständige) Liste der gebräuchlichen Interfaces.

Tabelle 110. *Einige der in vielen Dokumenttypen nutzbaren Interfaces.*

Interface	Beschreibung
com.sun.star.beans.XPropertySet	Holt und setzt Objekteigenschaften.
com.sun.star.container.XChild	Holt und setzt das elterliche Objekt, für Objekte, die nur ein Elternteil haben.
com.sun.star.datatransfer.XTransferable	Holt Daten für einen Datentransfer, zum Beispiel zum Kopieren in die Zwischenablage.
com.sun.star.document.XDocumentPropertiesSupplier	Greift auf die Dokumenteigenschaften zu, zum Beispiel auf Autor und Erstellungsdatum.
com.sun.star.document.XDocumentEventBroadcaster	Stelle, die bei auftretenden Ereignissen benachrichtigt wird.
com.sun.star.document.XEventsSupplier	Holt eine Liste der Ereignisse, die dieses Objekt unterstützt.
com.sun.star.document.XLinkTargetSupplier	Holt eine Liste der Linkziele in einem Dokument.
com.sun.star.document.XViewDataSupplier	Holt Eigenschaften, die die offenen Ansichten eines Dokuments beschreiben.
com.sun.star.drawing.XDrawPagesSupplier	Holt Folien für Dokumente, die mehrfache Folien unterstützen, zum Beispiel Zeichnungsdokumente oder Präsentationen.
com.sun.star.frame.XLoadable	Funktionalität zum Laden von Dokumenten.
com.sun.star.frame.XModel	Repräsentation einer Ressource (Dokument) in dem Sinne, dass sie von einer Ressource erzeugt/geladen wurde. Nebenbei gesagt, ich kenne kein Dokument, das XModel nicht einbindet. Dieses Interface stellt auch den Zugriff auf die Komponenten des Modells zur Verfügung.
com.sun.star.frame.XStorable	Methoden zum Speichern eines Dokuments.
com.sun.star.lang.XComponent	Erlaubt einem Objekt, ein eigenes Objekt zu besitzen und sich davon zu trennen, zum Beispiel eine Texttabelle.
com.sun.star.lang.XEventListener	Grundlegendes Interface für alle Listeners, stellt die Methode disposing zur Verfügung.
com.sun.star.lang.XMultiServiceFactory	Von Factories eingebunden, um Objekte zu erzeugen und um festzulegen, welche Objekte die Factory erzeugen kann.
com.sun.star.lang.XServiceInfo	Legt fest, welche Services ein Objekt unterstützt.

Interface	Beschreibung
com.sun.star.lang.XTypeProvider	Legt die Typen (Interfaces) fest, die ein Objekt unterstützt.
com.sun.star.script.XStarBasicAccess	Veraltet, bietet Zugriff auf Bibliotheken.
com.sun.star.style.XStyleFamiliesSupplier	Holt die von einem Dokument unterstützten Vorlagenfamilien, zum Beispiel Absatzvorlagen.
com.sun.star.util.XCloseBroadcaster	Erlaubt einem Objekt, eine Schließen-Anforderung zurückzuweisen.
com.sun.star.util.XCloseable	Bittet, ein Dokument zu schließen (close). Ist der Methode dispose() vorzuziehen.
com.sun.star.util.XModifiable	Stellt fest, ob ein Dokument verändert wurde.
com.sun.star.util.XModifyBroadcaster	Erfassung und Benachrichtigung, wenn ein Dokument verändert wurde.
com.sun.star.util.XNumberFormatsSupplier	Holt die Zahlenformate des Dokuments.
com.sun.star.view.XPrintJobBroadcaster	Erfassung und Benachrichtigung von Druckprozessen.
com.sun.star.view.XPrintable	Funktionalität zum Drucken eines Dokuments.
com.sun.star.view.XRenderable	Funktionalität zum Rendern eines Dokuments.

Ironischerweise wird die Suchfunktionalität nicht von allen Dokumenttypen unterstützt. Denn der Suchprozess ist stark abhängig vom Dokument. Zum Beispiel unterscheidet sich die Suche in einem Textdokument – was die Suchoptionen betrifft – sehr von der Suche in einem Tabellenblatt. In Calc wird die Suche vom einzelnen Blatt statt vom Dokument bereitgestellt.

Ein Calc-Dokument besteht aus mehreren Tabellenblättern. Ein wesentlicher Anteil der Funktionalität steckt daher in den Tabellenblättern und nicht im übergeordneten Calc-Dokument. Die Textsuche oder der Zugriff auf Folien existieren zum Beispiel in Tabellenblattobjekten – Folien werden detailliert im Kapitel 16. Zeichnungs- und Präsentationsdokumente vorgestellt.

Die API-Informationsseiten bieten ausführliche und detaillierte Hilfe zu den meisten Services und Interfaces. Die Internetadresse der html-Informationsseiten von AOO beginnt immer mit „http://www.openoffice.org/api/“, die von LO mit „https://api.libreoffice.org/docs/idl/ref/“. Wenn Sie das OOo-SDK-Paket installiert haben, können Sie auch die Dateien von Ihrer Festplatte nehmen. Die Adresse beginnt dann mit „file:///Installationspfad/sdk/“. Daran schließen sich „docs/common/ref/“ und der Interface-Name an. Zum Beispiel wird die Internetsite über das Interface com.sun.star.beans.XPropertySet unter den folgenden Adressen erreicht:

```
http://www.openoffice.org/api/docs/common/ref/com/sun/star/beans/XPropertySet.html
https://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1beans_1_1XPropertySet.html
```

13.3. Eigenschaften setzen und lesen

Die Eigenschaft dbg_properties ist ein String, der aus einer Liste der Eigenschaften besteht, die das Objekt unterstützt. Basic stellt diese Eigenschaften automatisch für den direkten Zugriff bereit. In anderen Sprachen mag das nicht so sein. Das Interface com.sun.star.beans.XPropertySet stellt Methoden zur Verfügung zum Setzen, Lesen und Listen der Objekteigenschaften, s. Tabelle 111.

Tabelle 111. *Methoden im Interface com.sun.star.beans.XPropertySet.*

Objektmethode	Beschreibung
getPropertySetInfo()	Gibt ein Objekt zurück, das das Interface com.sun.star.beans.XPropertySetInfo unterstützt. Dieses Objekt beschreibt die Objekteigenschaften, kann aber auch Null sein.
setPropertyValue(name, value)	Setzt den Wert der genannten Eigenschaft. Ein Listener darf diese Änderung zurückweisen.
getPropertyValue(name)	Gibt den Wert der genannten Eigenschaft zurück.

Objektmethode	Beschreibung
addPropertyChangeListener(name, listener)	Fügt der genannten Eigenschaft einen XPropertyChangeListener hinzu. Ein Leerstring als name setzt den Listener auf alle Eigenschaften an.
removePropertyChangeListener(name, listener)	Entfernt einen XPropertyChangeListener.
addVetoableChangeListener(name, listener)	Fügt der genannten Eigenschaft einen XVetoableChangeListener hinzu. Ein Leerstring als name setzt den Listener auf alle Eigenschaften an.
removeVetoableChangeListener(name, listener)	Entfernt einen XVetoableChangeListener.

In Basic wird auf Eigenschaften normalerweise direkt zugegriffen. Listing 285 zeigt zwei Wege, aus einem Writer-Dokument auf die Eigenschaft CharFontName zuzugreifen – beide Wege geben den Schriftartnamen zurück.

Listing 285. *Zwei Wege, den Namen der Schriftart zu entnehmen.*

```
Sub GetCharFontName
  Print ThisComponent.CharFontName
  Print CStr(ThisComponent.getPropertyValue("CharFontName"))
End Sub
```

Der direkte Zugriff auf die Eigenschaft ist mit Basic die einfachste Art, aber es gibt auch Gründe, die Methoden zu verwenden, die vom Interface XPropertySetInfo definiert sind. Einige Eigenschaften sind als optional definiert, so dass nicht jedes Dokument jede Eigenschaft besitzt. Das Interface XPropertySetInfo definiert die Objektmethode hasPropertyByName(), mit der man die Existenz einer Eigenschaft vor ihrer Verwendung überprüfen kann. Fehler kann man immer noch mit Error-Handler-Routinen abfangen. Des Weiteren kann man alle enthaltenen Eigenschaften mit ihren möglichen Werten auflisten, wie im Listing 286 zu sehen ist. Bild 88 zeigt einige Eigenschaften eines Writer-Dokuments, ausgegeben vom Makro im Listing 286.

Listing 286. *Ausgabe allgemeiner Dokumenteigenschaften.*

```
Sub GetPropertyValues
  Dim vPropInfo  'Das Objekt PropertySetInfo
  Dim vProps     'Array der Eigenschaften
  Dim vProp      'com.sun.star.beans.Property
  Dim v          'Der Wert einer einzelnen Eigenschaft
  Dim i%         'Indexvariable
  Dim s$         'Ausgabestring
  Dim nCount%

  REM Das Objekt bindet das Interface com.sun.star.beans.XPropertySetInfo ein.
  vPropInfo = ThisComponent.getPropertySetInfo()
  vProps = vPropInfo.getProperties()
  For i = 0 To UBound(vProps)
    If nCount = 30 Then
      nCount = 0
      MsgBox s, 0, "Eigenschaften"
      s = ""
    End If
    nCount = nCount + 1
    vProp = vProps(i)  'com.sun.star.beans.Property
    s = s & vProp.Name & " = "
    v = ThisComponent.getPropertyValue(vProp.Name)
    If IsNull(v) Then
      s = s & "Null"
```

```
    ElseIf IsEmpty(v) Then
      s = s & "Leer"
    ElseIf VarType(v) < 9 Then
      s = s & CStr(v)
    Else
      s = s & "Objekt oder Array"
    End If
    s = s & Chr$(10)
  Next
  MsgBox s, 0, "Eigenschaften"
End Sub
```

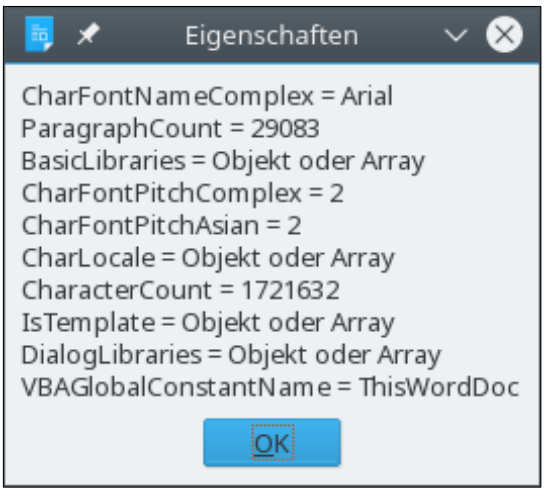


Bild 88. *Zehn Eigenschaften von ThisComponent.*

Tipp	Falls Ihnen die Bedeutung des Makros im Listing 286 nicht präsent ist: Es erlaubt Ihnen generell, jedes Objekt zu inspizieren, das das Interface XPropertySet unterstützt. Die Möglichkeit, ein Objekt zu inspizieren, ist außerordentlich wertvoll, wenn Sie nicht sicher wissen, was Sie mit einem Objekt tun können.

Wenn man ein Objekt mit den Informationen über die verfügbaren Eigenschaften (PropertySetInfo) erstellt hat,

- Erhält man mit getProperties() ein Array der Property-Objekte.
- Stellt man mit hasProperty(name) fest, ob eine Property existiert.
- Holt man mit getPropertyByName(name) eine einzelne Property.

13.4. Dokumenteigenschaften

Dokumenteigenschaften umfassen Informationen, die zum Dokument als Einheit gehören, wie Autor und Erstellungszeitpunkt. Im Gegensatz dazu stehen die im vorigen Abschnitt behandelten Eigenschaften, die zum Dokumentinhalt gehören, wie die aktuelle Schriftart.

Frühere Methoden zum Zugriff auf die Dokumenteigenschaften sind veraltet, zum Beispiel die Verwendung von getDocumentInfo zum Lesen und Verändern der Benutzerdaten. Über das Menü **Datei | Eigenschaften** öffnen Sie den entsprechenden Dialog. Mit getDocumentProperties erhalten Sie das Objekt DocumentProperties.

Listing 287. *Der Gebrauch des Objekts DocumentProperties.*

```
Sub GetThisDocProperties()
  ' com.sun.star.document.DocumentProperties
  PrintDocProperties(ThisComponent.getDocumentProperties())
End Sub
```

```
Sub PrintDocProperties(oDocProps)
  Dim s$
  Dim oData
  On Local Error Goto LO
  Dim iErrCount%

  'Die Fehlerbehandlung bezieht sich darauf, dass in AOO
  'das Struct com.sun.star.uti.DateTime eine Eigenschaft
  'HundredthSeconds hat, in LO aber nicht, stattdessen jedoch die
  'Eigenschaft NanoSeconds, deren Zahlenwert entsprechend größer ist.
  s = "Author : " & oDocProps.Author & Chr$(10) & _
      "AutoloadSecs : " & oDocProps.AutoloadSecs & Chr$(10) & _
      "AutoloadURL : " & oDocProps.AutoloadURL & Chr$(10) & _
      "DefaultTarget : " & oDocProps.DefaultTarget & Chr$(10) & _
      "Description : " & oDocProps.Description & Chr$(10) & _
      "EditingCycles : " & oDocProps.EditingCycles & Chr$(10) & _
      "EditingDuration : " & _
                SecondsAsPrettyTime(oDocProps.EditingDuration) & Chr$(10) & _
      "Generator : " & oDocProps.Generator & Chr$(10) & _
      "ModifiedBy : " & oDocProps.ModifiedBy & Chr$(10) & _
      "Title : " & oDocProps.Title & Chr$(10) & _
      "Language : " & oDocProps.Language.Country & ":" & _
                      oDocProps.Language.Language & Chr$(10) & _
      "ModificationDate : " & oDocProps.ModificationDate.Day & "." & _
                      oDocProps.ModificationDate.Month & "." & _
                      oDocProps.ModificationDate.Year &  " " & _
                      oDocProps.ModificationDate.Hours &  ":" & _
                      oDocProps.ModificationDate.Minutes &  ":" & _
                      oDocProps.ModificationDate.Seconds &  "."
  s = s & oDocProps.ModificationDate.HundredthSeconds & Chr$(10)
  s = s & "PrintDate : " & oDocProps.PrintDate.Day & "." & _
                      oDocProps.PrintDate.Month & "." & _
                      oDocProps.PrintDate.Year &  " " & _
                      oDocProps.PrintDate.Hours &  ":" & _
                      oDocProps.PrintDate.Minutes &  ":" & _
                      oDocProps.PrintDate.Seconds &  "."
  s = s & oDocProps.PrintDate.HundredthSeconds & Chr$(10)
  s = s & "PrintedBy : " & oDocProps.PrintedBy & Chr$(10) & _
      "Subject : " & oDocProps.Subject & Chr$(10) & _
      "TemplateDate : " & oDocProps.TemplateDate.Day & "." & _
                      oDocProps.TemplateDate.Month & "." & _
                      oDocProps.TemplateDate.Year &  " " & _
                      oDocProps.TemplateDate.Hours &  ":" & _
                      oDocProps.TemplateDate.Minutes &  ":" & _
                      oDocProps.TemplateDate.Seconds &  "."
  s = s & oDocProps.TemplateDate.HundredthSeconds & Chr$(10)
  s = s & "TemplateName : " & oDocProps.TemplateName & Chr$(10) & _
      "TemplateURL : " & oDocProps.TemplateURL & Chr$(10) & _
      "Title : " & oDocProps.Title & Chr$(10)
  MsgBox s, 0, "Dokumenteigenschaften"

  Dim i%
  oData = oDocProps.DocumentStatistics
  s = ""
  For i = LBound(oData) To UBound(oData)
    s = s & oData(i).Name & " : " & oData(i).Value & Chr$(10)
```

```
    Next
    MsgBox s, 0, "Dokumentstatistik"

    oData = oDocProps.Keywords
    s = ""
    For i = LBound(oData) To UBound(oData)
      s = s & oData(i) & Chr$(10)
    Next
    MsgBox s, 0, "Schlüsselwörter"
    Exit Sub

    LO:
      iErrCount = iErrCount + 1
      Select Case iErrCount
      Case 1
        s = s & oDocProps.ModificationDate.NanoSeconds & Chr$(10)
      Case 2
        s = s & oDocProps.PrintDate.NanoSeconds & Chr$(10)
      Case 3
        s = s & oDocProps.TemplateDate.NanoSeconds & Chr$(10)
      End Select
      Resume Next
End Sub
```

13.4.1. Dokumenteigenschaften eines nicht geöffneten Dokuments

Die Dokumenteigenschaften eines nicht geöffneten Dokuments sind leicht auszulesen.

Listing 288. *Liest die Dokumenteigenschaften aus einem nicht geöffneten Dokument.*

```
Sub LoadExternalProperties
  Dim sPath$
  Dim oDocProps
  sPath = ConvertToUrl("/andrew0/home/andy/MoveFigsFromFrames.odt")
  oDocProps = CreateUnoService("com.sun.star.document.DocumentProperties")
  oDocProps.loadFromMedium(sPath, Array())
  PrintDocProperties(oDocProps)
End Sub
```

13.4.2. Benutzerdefinierte Eigenschaften

Man kann eigene Dokumenteigenschaften erzeugen und entfernen. Bei der Erzeugung einer Eigenschaft steuern die PropertyAttribute-Konstanten der Tabelle 112 das Verhalten der erzeugten Eigenschaften.

Tabelle 112. *Die Konstantengruppe com.sun.star.beans.PropertyAttribute.*

Konstante	Wert	Beschreibung
MAYBEVOID	1	Der Wert der Eigenschaft darf fehlen.
BOUND	2	Bei einer Änderung der Eigenschaft wird ein PropertyChangeEvent an alle angemeldeten Eigenschaftsänderungslisteners gesendet.
CONSTRAINED	4	Bei einer Änderung der Eigenschaft wird ein PropertyChangeEvent an alle angemeldeten Änderungslisteners mit Vetorecht gesendet.
TRANSIENT	8	Der Eigenschaftswert wird nicht mit dem Dokument gespeichert.
READONLY	16	Der Eigenschaftswert ist schreibgeschützt.
MAYBEAMBIGUOUS	32	Der Eigenschaftswert kann mehrdeutig sein.
MAYBEDEFAULT	64	Der Eigenschaftswert kann als Standard gesetzt werden.

Konstante	Wert	Beschreibung
REMOVEABLE	128	Der Eigenschaftswert kann entfernt werden. Hieß früher REMOVABLE.
OPTIONAL	256	Die Eigenschaft ist optional.

Mit addProperty(Name, Attribute, Standardwert) erzeugen Sie eine neue Eigenschaft und mit removeProperty(Name) entfernen Sie eine Eigenschaft aus den Dokumenteigenschaften. Wenn Sie eine Eigenschaft erzeugen, wird sie den Dokumenteigenschaften hinzugefügt, so dass Sie das Objekt PropertySetInfo nutzen können.

Listing 289. *Fügt eine neue Dokumenteigenschaft hinzu.*

```
Sub AddNewDocumentProperty
  Dim oUDP  'Benutzerdefinierte Eigenschaft
  oUDP = ThisComponent.getDocumentProperties().UserDefinedProperties
  'Nachname des Benutzers
  If Not oUDP.getPropertySetInfo().hasPropertyByName("AuthorLastName") Then
    oUDP.addProperty("AuthorLastName", _
      com.sun.star.beans.PropertyAttribute.MAYBEVOID + _
      com.sun.star.beans.PropertyAttribute.REMOVEABLE + _
      com.sun.star.beans.PropertyAttribute.MAYBEDEFAULT, _
      "Default Last Name")
  End If
End Sub
```

13.4.3. Das veraltete Dokumentinfo-Objekt

Mit der veralteten Methode getDocumentInfo griff man früher auf die Benutzerdaten zu. Das Makro im Listing 290 setzt den Wert eines Benutzerfelds und gibt den Wert aus.

Tipp DocumentInfo und StandaloneDocumentInfo sind veraltet und ersetzt durch DocumentProperties.

Listing 290. *Veraltete Methode zum Lesen und Setzen der benutzerdefinierten Daten.*

```
Sub GetUserInfoFields
  Dim vDocInfo  'Objekt Dokumentinformation
  Dim s$        'Ausgabestring
  Dim i%        'Indexvariable
  vDocInfo = ThisComponent.getDocumentInfo()
  vDocInfo.setUserFieldValue(1, "My special user value")
  For i% = 0 To vDocInfo().getUserFieldCount() - 1
    s$ = s$ & vDocInfo.getUserFieldName(i) & " = " &_
         CStr(vDocInfo.getUserFieldValue(i)) & Chr$(10)
  Next
  MsgBox s$, 0, "Info-Felder"
End Sub
```

13.5. Ereignisse auflisten

Während OOo läuft, erzeugt es Ereignisse (Events), die Listeners darüber informieren, dass etwas geschehen ist. Das System von Event-Listener und -Informant sieht vor, dass beim Eintreten eines bestimmten Ereignisses ein Makro aufgerufen oder ein Listener informiert werden kann – zum Beispiel wenn ein Dokument geöffnet oder modifiziert wurde, oder wenn sich die Textauswahl ändert. Jeder Dokumenttyp unterstützt die beiden Interfaces com.sun.star.document.XEventBroadcaster und com.sun.star.document.XEventsSupplier und ermöglicht ihnen die Unterstützung ereignisbezogener

Aktivitäten – zum Beispiel Ereignisse an Listener zu melden und eine Liste der unterstützten Ereignisse bereitzustellen.

Das Makro im Listing 291 listet die Event-Listeners auf, die beim aktuellen Dokument und für OOo registriert sind. Außerdem listet dieses Makro die von einem Dokument unterstützten Ereignisse auf. Auch wenn keine Listeners registriert sind, bietet das Makro eine informative Liste der unterstützten Ereignistypen.

Listing 291. *Liste der Ereignisse des Dokuments.*

```
Sub DisplayAvailableEvents
  Dim oGEB    ' GlobalEventBroadcaster
  Dim oDoc
  Dim s$
  Dim oText
  oDoc = StarDesktop.loadComponentFromUrl("private:factory/swriter", _
                                          "_blank", 0, Array())
  oGEB = CreateUnoservice("com.sun.star.frame.GlobalEventBroadcaster")
  'Liste der globalen Ereignisse
  s = Join(oGEB.Events.getElementNames(), Chr$(13))
  oText = oDoc.Text
  oText.insertString(oText.End, "===Globale Ereignisse" & Chr$(13), False)
  oText.insertString(oText.End, s, False)

  'Liste der Writer-Dokumentereignisse
  s = Join(oDoc.Events.getElementNames(), Chr$(13))
  oText.insertString(oText.End, Chr$(13) & Chr$(13) & _
                     "===Writer-Ereignisse" & Chr$(13), False)
  oText.insertString(oText.End, s, False)
End Sub
```

Tabelle 113. *Eine Liste der Ereignisse.*

Global	Writer-Dokument
OnCopyTo	OnCopyTo
OnCopyToDone	OnCopyToDone
OnCopyToFailed	OnCopyToFailed
OnCreate	OnCreate
OnFocus	OnFieldMerge
OnLoad	OnFieldMergeFinished
OnLoadFinished	OnFocus
OnModeChanged	OnLayoutFinished
OnModifyChanged	OnLoad
OnNew	OnLoadFinished
OnPrepareUnload	OnMailMerge
OnPrepareViewClosing	OnMailMergeFinished
OnPrint	OnModeChanged
OnSave	OnModifyChanged
OnSaveAs	OnNew
OnSaveAsDone	OnPageCountChange
OnSaveAsFailed	OnPrepareUnload
OnSaveDone	OnPrepareViewClosing
OnSaveFailed	OnPrint
OnStorageChanged	OnSave
OnTitleChanged	OnSaveAs
OnUnfocus	OnSaveAsDone
OnUnload	OnSaveAsFailed
OnViewClosed	OnSaveDone
OnViewCreated	OnSaveFailed
OnVisAreaChanged	OnStorageChanged
	OnTitleChanged
	OnUnfocus
	OnUnload
	OnViewClosed
	OnViewCreated
	OnVisAreaChanged

Tipp Nicht nur Dokumente, auch viele andere Objekte unterstützen Event-Listeners. Sie können zum Beispiel einen Listener auf eine einzelne Zelle in einer Tabellenkalkulation setzen.

13.5.1. Einen eigenen Listener anmelden

Das Makro im Listing 291 greift mit Hilfe des GlobalEventBroadcaster auf Event-Listeners der OOo-Ebene zu. Das Makro im Listing 292 ersetzt den OnSave-Listener der Dokumente durch ein Makro mit dem Namen MySave im Module1 der Standard-Bibliothek in „Meine Makros“. Beim Speichern des Dokuments wird dann automatisch MySave aufgerufen. Diese Verbindung bleibt auch bestehen, wenn das Dokument geschlossen und wieder geöffnet wird.

Listing 292. *Registrierung Ihres eigenen Listeners.*

```
Sub StealAnEvent
  Dim mEventProps(1) As New com.sun.star.beans.PropertyValue
  mEventProps(0).Name = "EventType"
  mEventProps(0).Value = "StarBasic"
  mEventProps(1).Name = "Script"
  mEventProps(1).Value = "macro:///Standard.Module1.MySave()"

  ThisComponent.Events.replaceByName("OnSave", mEventProps())
End Sub
```

Achtung Mit der Adresse „macro://hello/Standard.Module1.MySave()“ benennen Sie ein Makro im Dokument „hello.odt“. Obwohl das gut funktioniert, geht die Verbindung nicht verloren, wenn das Dokument geschlossen ist.

Ein Ereignis kann genau so gut auf der OOo-Ebene mit dem GlobalEventBroadcaster gespeichert werden.

```
oGlobalEventBroadcaster = CreateUnoservice("com.sun.star.frame.GlobalEventBroadcaster")
oGlobalEventBroadcaster.Events.replaceByName("OnStartApp", mEventProps())
```

Verwechseln Sie die als Wert angegebene Adresse nicht mit der gleichen URL-Syntax, mit der eine Formularkontrolle an ein Makro gebunden wird.

```
"vnd.sun.star.script:Standard.Module1.LocalMySave?language=Basic&location=document"
```

13.5.2. Dispatch-Befehle abfangen

Ein Listener-Ereignis zu ersetzen ist nicht dasselbe, wie den Dispatch-Befehl **Datei > Speichern** abzufangen, denn das ist nur ein Listener, der aufgerufen wird, nachdem der Befehl ausgeführt wurde. Einen Dispatch abzufangen sieht eher aus wie der folgende Code von Paolo Mantovani. Eines der Probleme damit besteht darin, dass kein Status-Listener verwendet wird, so dass beim Abschalten eines Befehls das Menü dieses Kommando nicht als abgeschaltet ausweist.

Achtung Gehen Sie sehr achtsam vor, wenn Sie Listeners schreiben und Dispatch-Befehle abfangen. Rechnen Sie damit, dass sich etwas ändert oder dass Sie etwas falsch gemacht haben und OOo daraufhin abstürzt. Ich habe Sie gewarnt.

Listing 293. *Ersetzt ein paar Menübefehle.*

```
Global oDispatchInterceptor
Global oSlaveDispatchProvider
Global oMasterDispatchProvider
Global oFrame
Global bDebug As Boolean

Dim oCopyDispatch

Sub RegisterInterceptor
  oFrame = ThisComponent.CurrentController.Frame
  oDispatchInterceptor = CreateUnoListener("ThisFrame_", _
    "com.sun.star.frame.XDispatchProviderInterceptor")
  oFrame.registerDispatchProviderInterceptor(oDispatchInterceptor)
End Sub

Sub ReleaseInterceptor()
  On Error Resume Next
  oFrame.releaseDispatchProviderInterceptor(oDispatchInterceptor)
End Sub

Function ThisFrame_queryDispatch (oUrl, _
  sTargetFrameName As String, lSearchFlags As Long) As Variant

Dim oDisp
Dim sUrl As String

  'Slot Protocol lässt OOo abstürzen...
  If oUrl.Protocol = "slot:" Then
    Exit Function
  End If
```

```
  If bDebug Then
    Print oUrl.Complete
  End If

  'Hier kommt Ihre Dispatch-Behandlung:
  Select Case oUrl.Complete

    Case ".uno:Copy"
      oDisp = GetCopyDispatch 'Ersetzt den originalen Dispatch
    Case ".uno:Paste"
      oDisp = GetCopyDispatch 'Ersetzt den originalen Dispatch
    Case ".uno:Save"
      oDisp = GetCopyDispatch 'Ersetzt den originalen Dispatch
    Case ".uno:Undo"
      oDisp = GetCopyDispatch 'Ersetzt den originalen Dispatch
    'Case ".uno:blabla"
      'Irgendwas
    Case Else
      oDisp = _
      oSlaveDispatchProvider.queryDispatch(oUrl, sTargetFrameName, lSearchFlags)

  End Select

  ThisFrame_queryDispatch = oDisp
End Function

Function ThisFrame_queryDispatches (mDispatches) As Variant
  'ThisFrame_queryDispatches = mDispatches()
End Function

Function ThisFrame_getSlaveDispatchProvider () As Variant
  ThisFrame_getSlaveDispatchProvider = oSlaveDispatchProvider
End Function

Sub ThisFrame_setSlaveDispatchProvider (oSDP)
  oSlaveDispatchProvider = oSDP
End Sub

Function ThisFrame_getMasterDispatchProvider ()  As Variant
  ThisFrame_getMasterDispatchProvider = oMasterDispatchProvider
End Function

Sub ThisFrame_setMasterDispatchProvider (oMDP)
  oMasterDispatchProvider = oMDP
End Sub

Sub ToggleDebug
  bDebug = Not bDebug
End Sub

Function GetCopyDispatch()
  If Not IsNull(oCopyDispatch) Then
    oCopyDispatch = _
      CreateUnoListener("MyCustom_", "com.sun.star.frame.XDispatch")
  End If

  GetCopyDispatch = oCopyDispatch
End Function
```

```
Sub MyCustom_dispatch(URL, Arguments)
  Select Case URL.Complete
    Case ".uno:Copy"
      MsgBox "Sorry, der originale Dispatch " & URL.Complete & _
        " wurde von Paolo M. gestohlen", 48

    Case ".uno:Paste"
      ThisComponent.CurrentSelection(0).String = _
        "**** WILLKÜRLICHER INHALT DER ZWISCHENABLAGE VON PAOLO M. ****"
    Case ".uno:Save"
      MsgBox "Sorry, der originale Dispatch " & URL.Complete & _
        " wurde von Paolo M. gestohlen", 48

    Case ".uno:Undo"
      MsgBox "Rückgängig: Wie? Was????!!!???", 16

    Case Else

  End Select
End Sub

Sub MyCustom_addStatusListener(Control, URL)
End Sub

Sub MyCustom_removeStatusListener (Control, URL)
End Sub
```

Eine ähnliche Methode zum Abfangen von Kontextmenüs gibt es mit XContextMenuInterceptor.

13.6. Verknüpfungsziele

Mit Verknüpfungszielen, auch Sprungmarken genannt, kann man direkt an eine bestimmte Stelle springen. Der Dokumentnavigator enthält eine Liste von Verknüpfungszielen. Die Objektmethode getLinks() – definiert vom Interface com.sun.star.document.XLinkTargetSupplier – bietet den Zugriff auf die Verknüpfungsziele. Sie gibt ein Objekt zurück, das das Interface XNameAccess unterstützt. Mit anderen Worten, Sie können auf die Links mit den Methoden getByName(), getElementNames(), hasByName() und hasElements() zugreifen.

Das von der Methode getLinks() zurückgegebene Objekt kann nicht direkt auf die Links zugreifen, sondern bietet stattdessen den Zugang zu anderen Objekten, die das können. Um auf alle individuellen Links Zugriff zu erhalten, starten Sie über die Methode getLinks() mit der Liste der Linkfamilien. Die einzelne Linkfamilie erreichen Sie über den Familiennamen. Um zum Beispiel ein Objekt zu erhalten, das auf alle Tabellenlinks zugreifen kann, schreiben Sie:

```
oDoc.getLinks().getByName("Tables")
```

Wenn Sie nun die Linkfamilie haben, können Sie auch die einzelnen Links über ihre Namen erhalten. Mit diesem letzten Schritt haben Sie Zugriff sowohl auf den Namen des Links als auch auf das verlinkte Objekt. Das Makro im Listing 294 beschafft alle Linkfamilien mit ihren enthaltenen Links und gibt dann die Linknamen in einem neuen Textdokument aus.

Listing 294. *Die Sprungziele des aktuellen Dokuments.*

```
Sub GetJumpTargets
  Dim sLinkNames  'Tabellen, Textrahmen, Überschriften, Lesezeichen, usw...
  Dim vOneLink    'Ein Linktyp
  Dim i%          'Indexvariable
  Dim s$          'Ausgabestring
  Dim vTemp
  Dim noArgs()          'Ein leeres Array für die Argumente
  Dim vComp             'Die geöffnete Komponente
```

```
  Dim sURL As String    'URL des zu öffnenden Dokuments
  Dim oText

  sURL = "private:factory/swriter"
  vComp = StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, noArgs())
  oText = vComp.Text
  oText.insertString(oText.End, "LINKTYPEN" & Chr$(13), False)
  sLinkNames = ThisComponent.getLinks().getElementNames()
  oText.insertString(oText.End, Join(sLinkNames, Chr$(13)), False)
  oText.insertString(oText.End, Chr$(13) & Chr$(13) & "SPRUNGZIELE" & Chr$(13), False)
  For i = 0 To UBound(sLinkNames)
    vOneLink = ThisComponent.getLinks().getByName(sLinkNames(i))
    s = s & sLinkNames(i) & " = "
    If IsEmpty(vOneLink) Then
      s = s & "Leer"
    Else
      s = s & sLinkNames(i) & " : " & _
          Join(vOneLink.getElementNames(), Chr$(13) & sLinkNames(i) & " : ")
      REM Um das Linkobjekt selbst zu erhalten, etwa eine
      REM Texttabelle oder ein Grafikobjekt, machen Sie folgendes:
      REM vTemp = vOneLink.getElementNames()
      REM vObj = vOneLink.getByName(vTemp(0))
    End If
    s = s & Chr$(13)
  Next
  oText.insertString(oText.End, s, False)
End Sub
```

Sie können mit Sprungmarken (Verknüpfungszielen) direkt an eine bestimmte Stelle springen, wenn ein Dokument geladen wird. Wie man beim Öffnen eines Dokuments den Cursor an eine bestimmte Stelle setzt, zeigt Listing 295. Die Eigenschaft JumpMark setzt den Namen auf einen der Werte, die in dem vom letzten Makro erzeugten Dokument stehen. Der Name „Table1|table“ zum Beispiel bewirkt einen Sprung zu dieser Tabelle.

Listing 295. *Sprung auf ein Linkziel mit der Eigenschaft JumpMark.*

```
Dim Props(0)
Props(0).Name = "JumpMark"
Props(0).Value = "Table1|table"
sUrl = "file://c:/docs/Special_doc.odt"
vDoc = StarDesktop.loadComponentFromUrl(sUrl, "_blank", 0, Props())
```

Man kann eine Sprungmarke auch als Teil des URL angeben (s. Listing 296). Dazu wird sie nach einem Doppelkreuz (#) an das Ende des URL gesetzt. Sollte die Sprungmarke Sonderzeichen enthalten, etwa Leerzeichen, müssen sie mit der URL-Standardnotation kodiert werden. Ein Leerzeichen beispielsweise wird als %20 kodiert.

Listing 296. *Sprung auf ein Linkziel mit der Sprungmarke als Teil des URL.*

```
sUrl = "file://c:/docs/Special_doc.odt#Table1|table"
vDoc = StarDesktop.loadComponentFromUrl(sUrl, "_blank", 0, Props())
```

Tipp Das Zeichen „#“ hat viele Namen, Doppelkreuz, Gartenzaun, Nummernzeichen, Raute, Hashzeichen, um nur einige zu nennen.

13.7. Zugriff auf die Ansichtsdaten: XViewDataSupplier

Wenn Sie OOo nutzen, öffnen Sie ein Dokument, editieren es oder ändern etwas anderes (Sie wechseln zum Beispiel auf eine andere Seite oder ändern den Vergrößerungsfaktor), dann schließen Sie das Dokument und öffnen es wieder. Wenn Sie das tun, wird das Dokument an derselben Stelle auf dem Bildschirm, in derselben Größe und mit demselben Vergrößerungsfaktor geöffnet wie zu dem Zeitpunkt, als es zuletzt gespeichert wurde. Diese Information wird mit dem Dokument gespeichert und ist abrufbar durch das Interface com.sun.star.document.XViewDataSupplier. Dieses Interface stellt zwei Objektmethoden zur Verfügung, getViewData und setViewData. Das Makro im Listing 297 gibt die Ansichtsdaten für ThisComponent aus (s. Bild 89).

Listing 297. *Zeigt die Ansichtsdaten des aktuellen Dokuments.*

```
Sub GetViewData
  Dim vViewData 'Das Objekt View Data
  Dim i%        'Indexvariable
  Dim j%        'Indexvariable
  Dim s$        'Ausgabestring
  Dim vTemp     'Ein bestimmter Ansichtsaspekt als Objekt
  vViewData = ThisComponent.getViewData()
  REM Für jeden Ansichtsaspekt der Ansicht
  For i = 0 To vViewData.getCount() - 1
    vTemp = vViewData.getByIndex(i)
    For j = 0 To UBound(vTemp)
      s = s & vTemp(j).Name & " = " & CStr(vTemp(j).Value) & Chr$(10)
    Next
    MsgBox s, 0, "Ansichtsdaten"
  Next
End Sub
```

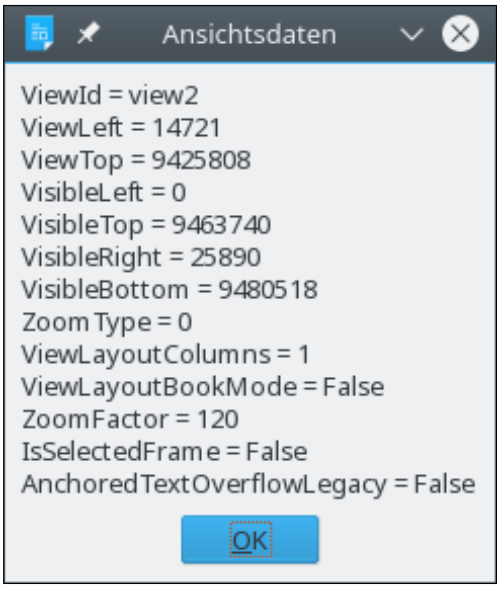


Bild 89. *Ansichtsdaten eines Dokuments.*

13.8. Ein Dokument schließen: XCloseable

Seit OOo 1.1.0 unterstützt jeder Dokumenttyp das Interface com.sun.star.util.XCloseable. Zum Schließen solcher Objekte rufen Sie die Objektmethode close(bForce) auf. Ist bForce True, muss das Objekt geschlossen werden. Es darf sich also nicht gegen das Schließen wehren. Wenn bForce hingegen False ist, darf sich das Objekt weigern, geschlossen zu werden.

Bevor ein Dokument geschlossen werden soll, wird eine Nachricht an alle registrierten Listeners gesendet, die dadurch die Gelegenheit erhalten, das Schließen zu verhindern. Wenn es kein Veto von

einem der Listeners auf die Schließen-Anforderung gibt, wird eine Nachricht an alle registrierten Listeners gesendet mit der Information, dass das Dokument geschlossen wird. Das Interface XCloseBroadcaster stellt Methoden zum Anlegen und Entfernen eines Schließen-Listeners zur Verfügung (s. Tabelle 114).

Tabelle 114. *Methoden im Interface com.sun.star.util.XCloseBroadcaster.*

Objektmethode	Beschreibung
addCloseListener(XCloseListener)	Fügt einen Listener für „Close“-Ereignisse hinzu, mit oder ohne Veto.
removeCloseListener(XCloseListener)	Entfernt ein Objekt, das mit addCloseListener() registriert wurde.

Das Makro im Listing 298 zeigt, wie ein Dokument unabhängig von der OOo-Version geschlossen wird. Für Versionen vor 1.1.0 gibt es die Methode close() nicht, stattdessen musste man die Methode dispose() wählen. Die Methode dispose() wirkt bedingungslos und sollte besser nicht zum Schließen eines Dokuments verwendet werden (mit Dokument meine ich natürlich jedes Objekt, das sowohl über close als auch dispose verfügt), denn sie bietet nicht die Möglichkeit, dass ein registrierter Nutzer, der gerade das Dokument bearbeitet, das Schließen ablehnen und die momentane Bearbeitung beenden kann.

Listing 298. *Der sichere Weg, ein Dokument zu schließen.*

```
If HasUnoInterfaces(oDoc, "com.sun.star.util.XCloseable") Then
  oDoc.close(True)
Else
  oDoc.dispose()
End If
```

Achtung Verwenden Sie nicht dispose, um ein Dokument zu schließen. Die Methode existiert nur noch zur Rückwärtskompatibilität. Nehmen Sie zum Beispiel an, dass Sie den Ausdruck eines Dokuments starten und dann sofort das Dokument mit dispose() schließen. Das zum Druck gegebene Dokument existiert plötzlich nicht mehr, und OOo stürzt ab.

13.9. Folien: XDrawPagesSupplier

13.9.1. Draw und Impress

Draw und Impress sind in Bezug auf die unterstützten Interfaces nahezu identisch. Die Spezialität von Draw sind unabhängige Grafikobjekte, wohingegen es bei Impress um geschäftliche Effekte und Präsentationen geht. Die Zeichnungsfunktionalitäten von Draw und Impress sind jedoch identisch. Die Grafikobjekte werden auf „Folien“ (draw pages) gezeichnet und dargestellt. Das Konzept von Draw wie auch von Impress sieht multiple Folien vor. Die Funktionalität zum Abruf eines DrawPage-Objekts wird vom Interface com.sun.star.drawing.XDrawPagesSupplier definiert. Das Interface com.sun.star.drawing.XDrawPages definiert Methoden, einzelne Folien abzurufen, einzufügen und zu entfernen (s. Tabelle 115).

Tabelle 115. *Methoden im Interface com.sun.star.drawing.XDrawPages.*

Objektmethode	Beschreibung
InsertNewByIndex(index)	Erzeugt eine neue Folie oder Masterfolie und fügt sie ein.
remove(XDrawPage)	Entfernt eine Folie oder Masterfolie.
getCount()	Gibt die Anzahl der Folien zurück.
getByIndex(index)	Gibt eine bestimmte Folie zurück.
hasElements()	Gibt True zurück, wenn Folien vorhanden sind.

Das Makro im Listing 299 zeigt, wie man durch die Liste der Folien iteriert. Jede Folie wird als JPG-Datei exportiert. Der Exporttyp wird durch die Eigenschaft MediaType bestimmt.

Listing 299. *Exportiert jede Grafikfolie als JPG.*

```
OFilter = CreateUnoService("com.sun.star.drawing.GraphicExportFilter")
Dim args(1) As New com.sun.star.beans.PropertyValue
Dim oPage, oFilter, sName&
For i = 0 To oDoc.getDrawPages().getCount() - 1
  oPage = oDoc.getDrawPages().getByIndex(i)
  sName = oPage.Name
  oFilter.setSourceDocument(oPage)
  args(0).Name = "URL"
  args(0).Value = "file:///c|/" & sName & ".JPG"
  args(1).Name = "MediaType"
  args(1).Value = "image/jpeg"
  oFilter.filter(args())
Next
```

Tipp Der Index zum Zugriff auf die Folien ist nullbasiert. Das heißt, dass die erste Folie die Position 0 hat. Wenn ein Dokument vier Folien enthält, werden sie von 0 bis 3 gezählt. Daher wird die Schleife im Listing 299 durch „For i = 0 To oDoc.getDrawPages().getCount() - 1“ bestimmt.

Das Makro im Listing 300 erzeugt eine neue Präsentation und fügt dann eine Grafik in die erste Folie ein. Die Bildgröße wird auf das vorhandene Seitenverhältnis angepasst.

Listing 300. *Fügt eine Grafik im richtigen Seitenverhältnis in eine Folie ein.*

```
Sub AddProportionalGraphic
  Dim oDoc       'Neues Präsentationsdokument
  Dim oDrawPage  'Die Folie, die die Grafik erhalten soll
  Dim oGraph     'Die neu erzeugte Grafik

  REM Erzeugt ein Impress-Präsentationsdokument
  oDoc = StarDesktop.loadComponentFromURL("private:factory/simpress",_
                                          "_default", 0, Array())

  REM Falls gewünscht, wird eine neue Folie eingefügt,
  REM wobei die erste Folie erhalten bleibt.
  REM Für die Objektmethode getDrawPages() steht in Basic auch die Eigenschaft
  REM DrawPages zur Verfügung:
  REM oDrawPage = oDoc.DrawPages.insertNewByIndex(1)
  REM oder oDrawPage = oDoc.getDrawPages().insertNewByIndex(1)
  REM In diesem Fall wird einfach die erste Folie genommen.
  oDrawPage = oDoc.getDrawPages().getByIndex(0)

  REM Erzeugt ein Grafikobjekt, das in das Dokument eingefügt werden soll.
  oGraph = oDoc.createInstance("com.sun.star.drawing.GraphicObjectShape")

  REM Setzt den URL der Grafik, so dass sie eingefügt werden kann.
  oGraph.GraphicURL = "http://www.openoffice.org/images/AOO_logos/orb.jpg"
  oDrawPage.add(oGraph)

  REM Wenn ich hier aufhöre, wird die Grafik sehr klein oben links
  REM im Dokument erscheinen. Das ist ziemlich sinnlos.
  REM Obwohl ich einfach die Grafik auf die Größe der Bitmap skalieren könnte,
  REM habe ich sie lieber so skaliert, dass sie so groß wie möglich
  REM ohne Änderung des Seitenverhältnisses wird.
  REM Dazu muss das Verhältnis von Höhe zu Breite sowohl für das Bild
```

```
  REM als auch für die Folie bestimmt werden.

  Dim oNewSize As New com.sun.star.awt.Size    'Neue Bildgröße
  Dim oBitmapSize As New com.sun.star.awt.Size 'Bitmap-Größe

  Dim dImageRatio As Double     'Bild-Seitenverhältnis Höhe zu Breite
  Dim dPageRatio As Double      'Folien-Seitenverhältnis Höhe zu Breite

  oBitmapSize = oGraph.GraphicObjectFillBitmap.getSize()
  dImageRatio = CDbl(oBitmapSize.Height) / CDbl(oBitmapSize.Width)
  dPageRatio = CDbl(oDrawPage.Height) / CDbl(oDrawPage.Width)

  REM Vergleich der Seitenverhältnisse: welches ist relativ gesehen breiter?
  If dPageRatio > dImageRatio Then
    oNewSize.Width  = oDrawPage.Width
    oNewSize.Height = CLng(CDbl(oDrawPage.Width) * dImageRatio)
  Else
    oNewSize.Width  = CLng(CDbl(oDrawPage.Height) / dImageRatio)
    oNewSize.Height = oDrawPage.Height
  End If

  REM Zentriert das Bild auf der Präsentationsfolie.
  Dim oPosition As New com.sun.star.awt.Point
  oPosition.X = (oDrawPage.Width - oNewSize.Width) / 2
  oPosition.Y = (oDrawPage.Height - oNewSize.Height) / 2

  oGraph.SetSize(oNewSize)
  oGraph.SetPosition(oPosition)
End Sub
```

Wie schon gesagt, Impress und Draw sind sehr ähnlich hinsichtlich der unterstützten API. Das Makro im Listing 301 zeichnet Linien in einem Zeichnungsdokument (s. Bild 90).

Listing 301. *Zeichnet Linien in einem neuen Zeichnungsdokument.*

```
Sub DrawLinesInDrawDocument
  Dim oDoc       'Neues Zeichnungsdokument
  Dim oDrawPage  'Die Folie, die die Grafik erhalten soll
  Dim oShape     'Die einzufügende Form

  REM Erstellt ein neues Zeichnungsdokument.
  oDoc = StarDesktop.loadComponentFromURL("private:factory/sdraw",_
                                          "_default", 0, Array())

  REM Die erste Folie wird gewählt
  oDrawPage = oDoc.getDrawPages().getByIndex(0)

  Dim i As Long
  Dim oPos As New com.sun.star.awt.Point   'Position eines Punktes
  Dim oSize As New com.sun.star.awt.Size   'Grafikgröße
  Dim dStepSize As Double                  'Schrittweite
  dStepSize = CDbl(oDrawPage.Height) / 10

  For i = 0 To 10
    'Alle Maßangaben sind in Hundertstel mm.
    oShape = oDoc.createInstance("com.sun.star.drawing.LineShape")
    oShape.LineColor = RGB(0, 255 - 20 * i, 20 * i)  'Linienfarbe
```

```
        oShape.LineWidth = 250              'Linienstärke

        oPos.X = 0                          'Abstand von links
        oPos.Y = CLng(CDbl(i) * dStepSize)  'Abstand von oben
        oShape.setPosition(oPos)

        oSize.Width = oDrawPage.Width                'Breite
        oSize.Height= oDrawPage.Height - 2 * oPos.Y  'Höhe
        oShape.setSize(oSize)
        oDrawPage.add(oShape)
    Next
End Sub
```

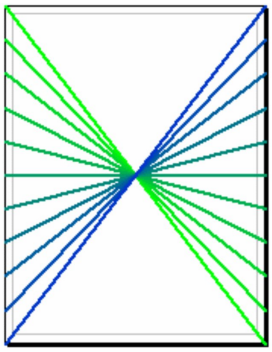


Bild 90. *Sich kreuzende Linien in einem Zeichnungsdokument.*

13.9.2. Linien mit Pfeilen zeichnen in Calc

Jedes Calc-Dokument enthält eine einzelne Folie für jedes Tabellenblatt. In Calc ist jedes Tabellenblatt wie eine transparente Ebenenschicht, bei der Zeichnungsdaten über den Standard-Dokumentdaten liegen. Das folgende Makro zeigt ein paar nützliche Dinge:

- Jede Zelle hat eine bestimmte Position auf der Folie. Dieses Makro zeichnet eine Linie zwischen zwei Zellen und nutzt dabei das Zellattribut Position. Die Linie wird von der oberen linken Ecke der Zelle B2 bis zur oberen linke Ecke der Zelle D4 gezeichnet.
- Die Form LineShape wird vom Dokument erzeugt.
- Danach werden die LineShape-Attribute gesetzt, zum Beispiel Position, Size und LineColor.
- Die Form LineShape wird der Folie hinzugefügt.
- Pfeile und andere Linienenden werden hinzugefügt, *nachdem* die Form der Folie hinzugefügt wurde – wenigstens ist das so in OOo 3.3.0 notwendig.

Pfeile als Linienenden werden durch LineEndName und LineStartName gesetzt. Mir ist es nicht gelungen, eine Liste der unterstützten Pfeilnamen zu finden, also habe ich die Namen verwendet, die ich im GUI finden konnte. Mit den wenigen, die ich ausprobiert habe, funktionierte es. Dann habe ich den Quellcode durchsucht und die Dateien filter/source/msfilter/escherex.cx und svx/source/dialog/sdstring.src entdeckt, in denen die folgenden Namen stehen:

- Arrow
- Arrow concave
- Circle
- Dimension Lines
- Double Arrow
- Line Arrow

- Rounded large Arrow
- Rounded short Arrow
- Small Arrow
- Square
- Square 45
- Symmetric Arrow

Bei meinen Tests funktionierten die US-Namen auch in anderen Gebietsschemata. Es geht auch mit spezifischen lokalen Benennungen, aber der Wert wird immer unter dem US-Namen gespeichert. Wenn man also mit einem französischen Gebietsschema den Namen „Double flèche“ benutzt, wird der Wert als „Double Arrow“ gesetzt.

Listing 302. *Zeichnet eine Linie in einem Calc-Dokument.*

```
Sub InsertLineInCalcDocument
  Dim oLine
  Dim oCell1
  Dim oCell2
  Dim oSheet

  Dim oPos As New com.sun.star.awt.Point    'Position eines Punktes
  Dim oSize As New com.sun.star.awt.Size    'Größe eines Rechtecks
  Dim oPage

  oSheet = ThisComponent.Sheets(0)          'Erstes Tabellenblatt
  oCell1 = oSheet.getCellByPosition(1, 1)   'Zelle B2
  oCell2 = oSheet.getCellByPosition(3, 3)   'Zelle D4

  oLine = ThisComponent.createInstance("com.sun.star.drawing.LineShape")

  oPos.X = oCell1.Position.X    'linke obere Ecke, Abstand von links
  oPos.Y = oCell1.Position.Y    'linke obere Ecke, Abstand von oben
  oLine.Position = oPos

  oSize.Width = oCell2.Position.X - oCell1.Position.X    'Breite
  oSize.Height = oCell2.Position.Y - oCell1.Position.Y   'Höhe
  oLine.Size = oSize

  oLine.LineWidth = 4                   'Linienstärke
  oLine.LineColor = RGB(128, 0, 0)      'Linienfarbe
  oPage = oSheet.getDrawPage()

  oPage.add(oLine)                      'Die Linie wird in die Folie eingefügt.

  REM Muss NACH dem Einfügen der Liniene in die Folie getan werden.
  oLine.LineEndName = "Arrow"
  oLine.LineStartName = "Double Arrow"
End Sub
```

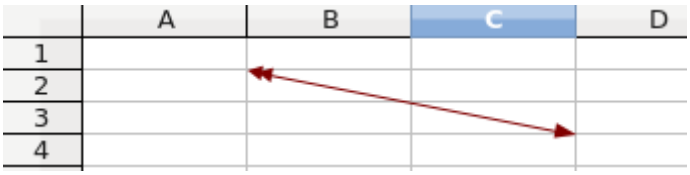


Bild 91. *Eine Linie in einem Calc-Dokument zeichnen.*

13.9.3. Writer

Jedes Writer-Dokument enthält eine einzige Folie für das gesamte Dokument. Im Writer ist eine Seite wie eine transparente Ebenenschicht, bei der Zeichnungsdaten über den Standard-Dokumentdaten liegen.

Writer-Dokumente unterstützen nicht das Interface XDrawPagesSupplier, weil sie nur eine einzige Folie enthalten. Allerdings unterstützen sie das Interface XDrawPageSupplier, das nur eine Objektmethode definiert: getDrawPage().

Das Makro im Listing 301 nutzt optionale Folieneigenschaften, nämlich die Höhe und die Breite. Die Folie eines Writer-Dokuments hat diese Eigenschaften nicht, dafür aber andere Eigentümlichkeiten. Wenn man zum Beispiel der Folie Linien hinzufügt – wie im Listing 301 –, werden sie an der Cursorposition als Zeichen eingefügt, statt dass die Positionen als spezifische Lage im Dokument verstanden werden. Das Makro im Listing 303 zeichnet Linien, die dieses Verhalten demonstrieren (s. auch Bild 92).

Listing 303. *Zeichnet Linien in ein Writer-Dokument.*

```
Sub DrawLinesInWriterDocument
  Dim oDoc       'Neues Writer-Dokument
  Dim oDrawPage  'Die Folie, die die Grafik erhalten soll
  Dim oShape     'Die einzufügende Form

  REM Erzeugt ein neues Writer-Dokument.
  oDoc = StarDesktop.loadComponentFromURL("private:factory/swriter",_
                                          "_default", 0, Array())

  oDrawPage = oDoc.getDrawPage()

  Dim i As Long
  Dim oSize As New com.sun.star.awt.Size  'Rechteckgröße
  Dim dStepSize As Double                 'Schrittweite
  dStepSize = 800

  For i = 0 To 10
    'Alle Maßangaben sind in Hundertstel mm.
    oShape = oDoc.createInstance("com.sun.star.drawing.LineShape")
    oShape.LineColor = RGB(255, 255 - 20 * i, 20 * i)  'Linienfarbe
    oShape.LineWidth = 50                              'Linienstärke
    'Linie im umschriebenen Rechteck: Breite und Höhe
    oSize.Width = CLng(dStepSize / 5 * i - dStepSize)
    oSize.Height= dStepSize
    oShape.setSize(oSize)
    oDrawPage.add(oShape)
  Next
End Sub
```

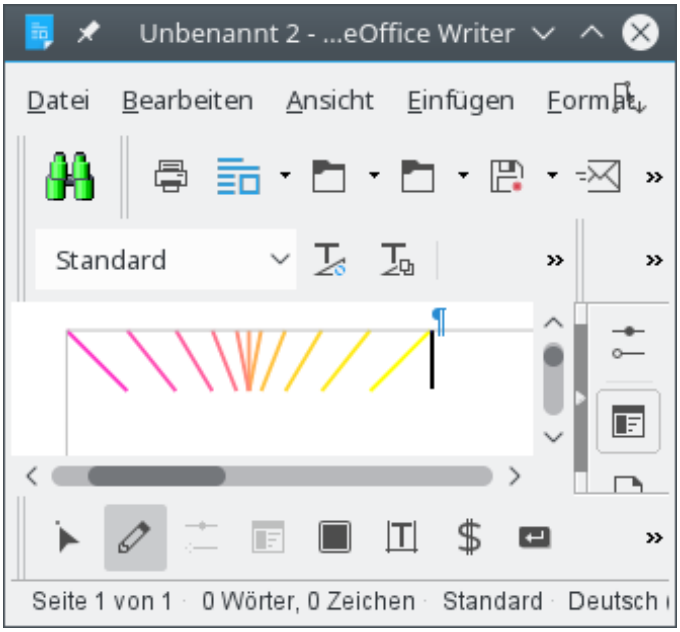


Bild 92. *Die in ein Writer-Dokument gezeichneten Linien werden als Zeichen behandelt.*

13.10. Das Modell

XModel ist das wichtigste Interface, mit dem man eine Komponente als Dokument erkennt, im Gegensatz zur Basic-IDE oder den inklusiven Hilfeseiten. Objekte, die das Interface com.sun.star.frame.XModel einbinden, stellen eine Komponente dar, die von einem URL erzeugt wurde. OOo-Dokumentobjekte können von einem Controller angesteuert werden, den man auch als Ansicht des Dokuments ansehen kann. Das Interface XModel definiert die Objektmethoden in der Tabelle 116.

Tabelle 116. *Methoden im Interface com.sun.star.frame.XModel.*

Objektmethode	Beschreibung
getURL()	Gibt den URL des Dokuments als String zurück.
getArgs()	Gibt eine Kopie des Service com.sun.star.document.MediaDescriptor für dieses Modell (Dokument) zurück.
lockControllers()	Hält manche Ansichtsaktualisierungen zurück – Makros können dadurch schneller ablaufen. Achten Sie darauf, zum Ende des Makros die Controller wieder zu entsperren.
unlockControllers()	Entsperrt die Ansichtsaktualisierung. Wird einmal für jeden lockControllers()-Befehl aufgerufen.
hasControllersLocked()	Gibt es noch wenigstens eine Aktualisierungssperre?
getCurrentController()	Der Controller, der aktuell dieses Modell ansteuert.
getCurrentSelection()	Die aktuelle Auswahl mit dem aktuellen Controller.

13.10.1. Dokumentargumente

Beim Laden eines Dokuments können Argumente zur Steuerung des Ladevorgangs übergeben werden, um zum Beispiel ein Dokument schreibgeschützt zu öffnen. Mit getArgs() erhält man den Mediadeskriptor, der im Einzelnen dokumentiert, wie das Dokument geöffnet wurde. Der Service MediaDescriptor kann als eine Reihe von optionalen Eigenschaften oder als Array von Eigenschaften abgerufen werden, s. Listing 304 und Bild 93.

Listing 304. *Gibt den Mediadeskriptor des Dokuments aus.*

```
Sub PrintDocumentArgs()
  REM Alle Zeilen, die einen Fehler produzieren, werden ignoriert.
  REM Einen Wert auszulesen, kann einen Fehler verursachen, aber
  REM das ist kein Problem. Der Wert wird einfach nicht ausgegeben.
  On Error Resume Next

  Dim vArgs 'Der Mediadeskriptor als com.sun.star.beans.PropertyValue-Array
  Dim s$    'Ausgabestring
  Dim i%    'Indexvariable

  REM Holt den Mediadeskriptor.
  REM Dieser kann aus einem Array von PropertyValue-Structs bestehen.
  vArgs = ThisComponent.getArgs()

  For i = 0 To UBound(vArgs)        'Jede einzelne Eigenschaft:
    s = s & vArgs(i).Name & " = "   'Eigenschaftsname und Gleichheitszeichen.
    s = s & vArgs(i).Value          'Den Wert auszulesen kann zu einem Error führen!
    s = s & Chr$(10)                'Abschließender Zeilenumbruch
  Next
  MsgBox s, 0, "Args"
End Sub
```

Bild 93. *Von der Objektmethode getArgs() zurückgegebene Eigenschaften.*

Eine Textdatei ist keine normale Writer-Datei, sondern eine einfache Datei, die normalerweise von einem Editor wie Notepad erstellt wurde. Textdateien enthalten keine Informationen über die verwendete Schriftart oder über die Art des Zeilenumbruchs. Textdateien haben häufig die Dateinamenserweiterung TXT. Beim Öffnen einer Textdatei zeigt OOo einen Dialog und stellt ein paar Fragen über die Datei, um ihre Kodierung zu erfahren. Man hat mich gefragt, wie eine Textdatei mit einem Makro geöffnet wird, ohne den Dialog, mit dem die spezifischen Importwerte festgelegt werden. Auch wenn ich in der Lage war, den Import-Filternamen korrekt zu bestimmen, so hatte ich doch keine Idee, wie die benötigten Filteroptionen zu setzen waren – bis ich die Objektmethode getArgs() fand.

Tipp Wenn Sie wissen, wie man ein Dokument über die GUI öffnet, aber nicht sicher sind, welche Argumente zu setzen sind, um das Dokument mit loadComponentFromURL zu laden, dann öffnen Sie das Dokument erst einmal in der GUI und inspizieren den Mediadeskriptor des Dokuments. Ein Hinweis: schauen Sie sich die Filteroptionen an.

Bild 93 zeigt die Argumente für eine importierte Textdatei. Die Eigenschaft FilterName gibt den Namen des Importfilters an, und die Eigenschaft FilterOptions gibt die Filteroptionen an, mit denen die Datei geöffnet wurde. Gibt man diese Eigenschaften der Desktop-Methode loadComponentFromUrl() mit, wird die Datei korrekt geöffnet, ohne dass der Dialog auftaucht. Tabelle 117 enthält eine Liste der Eigenschaften, die von der Objektmethode getArgs() zurückgegeben werden können. Auf der API-Website finden sich noch weitere Eigenschaften, die aber entweder unbedeutend oder veraltet sind.

Tabelle 117. *Eigenschaften im Service com.sun.star.document.MediaDescriptor.*

Eigenschaft	Beschreibung
Aborted	Kann gesetzt sein, wenn der Ladevorgang bei der Passworteingabe abgebrochen wird.
AsTemplate	Angabe, ob das Dokument als Template geöffnet wurde.
Author	Der Autor dieser Dokumentversion, für die Versionierung.
CharacterSet	Zeichensatz des Dokuments für Ein-Byte-Zeichen.
Comment	Kommentar zur aktuellen Dokumentversion, für die Versionierung.
DocumentTitle	Dokumenttitel, falls vorhanden.
FilterName	Name des Filters, mit dem das Dokument importiert oder gespeichert wird.
FilterOptions	Filteroptionen für den Import des Dokuments.
FilterData	Weitere Importeigenschaften, wenn der String FilterOptions nicht ausreicht.
Hidden	Angabe, ob beim Laden das Argument Hidden (Verborgen) gesetzt wurde.
HierarchicalDocumentName	Der hierarchische Pfad von der obersten Ebene zum eingebetteten Dokument.
InputStream	Der InputStream, wenn er während des Ladens gesetzt wurde.
InteractionHandler	Error-Handler für Fehler, die während des Imports auftreten.
JumpMark	Sprungziel, auf das nach dem Öffnen des Dokuments gesprungen wird.
MediaType	MIME-Typ des Dokuments.
OpenNewView	Öffnet eine neue Ansicht für ein schon geladenes Dokument, statt das Dokument noch einmal zu öffnen. Es werden also zwei Ansichten für dieselben Daten angefordert.
OutputStream	Der OutputStream, der beim Speichern des Dokuments verwendet wird.
Overwrite	Angabe, ob beim Speichern eine vorhandene Datei überschrieben wird.
Password	Passwort zum Öffnen oder Speichern des Dokuments.
Preview	Angabe, ob das Dokument im Vorschaumodus (optimiert) geöffnet wird.
ReadOnly	Angabe, ob das Dokument schreibgeschützt geöffnet wird. Der Controller wird das Dokument nicht verändern.
Referer	URL des Referenzgebers – zum Beispiel, wenn das Dokument durch Klick auf einen HTTP-Link geöffnet wird. (Achtung: Referer, NICHT Referrer).
RepairPackage	Angabe, ob das Dokument im Reparaturmodus geöffnet wird.
StartPresentation	Angabe, ob direkt nach dem Öffnen eines Impress-Dokuments die Präsentation starten soll.
StatusIndicator	Ein Statusindikator, wenn er beim Laden des Dokuments angegeben wurde.
Unpacked	Angabe, ob ein OOo-Dokument als Verzeichnis und nicht als ZIP-Datei gespeichert wird.

Eigenschaft	Beschreibung
URL	URL des Dokuments.
Version	Aktuelle Dokumentversion, für die Versionierung.
ViewData	Die zu nutzenden Ansichtsfestlegungen.
ViewId	Die ID der Startansicht.
MacroExecutionMode	Angabe, wie Makros beim Laden des Dokuments behandelt werden.

13.11. Ein Dokument speichern

Die Stelle, an der ein Dokument gespeichert wird, heißt Uniform Resource Locator (URL) – mit anderen Worten, sein Dateiname. Der URL einer Datei besteht normalerweise aus dem vollständigen Dateipfad. Wenn beim Dateinamen vom URL die Rede ist, sieht er so ähnlich aus wie „file:///c:/meinedatei.odt“ und nicht wie „c:\meinedatei.odt“. Ein URL ist eine allgemein gültige Form der Speicherortsbenennung, eine, die bequem erweitert werden kann, um in hersteller- und rechnerunabhängiger Weise einen breiten Rahmen von Speicherortstypen abzudecken. Basic bietet die Funktionen ConvertToURL und ConvertFromURL zur Konvertierung zwischen den beiden Notationen. Das Interface XStorable definiert Objektmethoden zum Speichern eines Dokuments in einen URL (s. Tabelle 118).

Tabelle 118. *Methoden im Interface com.sun.star.frame.XStorable.*

Objektmethode	Beschreibung
hasLocation()	True, wenn das Dokument einen Speicherort hat. False, wenn es ein neues, noch nicht gespeichertes Dokument ist.
getLocation()	Gibt den URL zurück, an dem das Objekt mit storeAsURL() gespeichert wurde.
isReadonly()	Man kann die Methode store() nicht aufrufen, wenn die Datei von einem schreibgeschützten Speicherort geöffnet wurde.
store()	Speichert die Daten an dem aktuellen URL.
storeAsURL(URL, args)	Speichert das Dokument am angegebenen URL, der dann zum aktuellen URL wird.
storeToURL(URL, args)	Speichert das Dokument am angegebenen URL, der aktuelle URL ändert sich jedoch nicht.

Mit der Objektmethode hasLocation() ermitteln Sie, ob ein Dokument den eigenen Speicherort kennt, und mit der Methode store() speichern Sie es am aktuellen URL. Das Makro im Listing 305 verwendet sowohl von XStorable als auch von XModifiable definierte Methoden zum Speichern eines Dokuments auf dem Speichermedium. Das Dokument wird nur gespeichert, wenn es seinen Speicherort kennt, wenn es geändert wurde und wenn es nicht schreibgeschützt ist.

Listing 305. *Korrekte Methode, ein Dokument zu speichern.*

```
If ThisComponent.isModified() Then
  If (ThisComponent.hasLocation() And (Not ThisComponent.isReadOnly())) Then
    ThisComponent.store()
  Else
    REM Entweder hat das Dokument keinen Speicherort oder es kann nicht
    REM gespeichert werden, weil der Speicherort schreibgeschützt ist.
    ThisComponent.setModified(False)
  End If
End If
```

Direkt nachdem ein Dokument erzeugt wurde, hat es noch keinen Speicherort. Ein Dokument hingegen, das von der Festplatte geöffnet wurde, hat einen bekannten Speicherort. Mit den Methoden storeAsURL() oder storeToURL() speichern Sie ein Dokument am angegebenen Ort. Der Unterschied zwischen den beiden Methoden besteht darin, dass storeAsURL() den aktuellen Ort (URL)

neu festlegt, storeToURL() aber nicht. Die Abfolge der Aktionen in der Tabelle 119 macht den Unterschied deutlich.

Tabelle 119. *Der Unterschied zwischen storeToURL und storeAsURL.*

Schritt	Aktion	Kommentar
1	Dokument erzeugen.	Die Methode store() kann nicht aufgerufen werden, weil das Dokument noch keinen Speicherort hat.
2	StoreToURL aufrufen	Das Dokument wird gespeichert. Die Methode store() kann aber nicht aufgerufen werden, weil es noch keinen Speicherort hat.
3	StoreAsURL aufrufen	Die Methode store() kann aufgerufen werden, weil das Dokument nun einen Speicherort hat.
4	StoreToURL aufrufen	Das Dokument wird gespeichert, aber der Speicherort ist noch derselbe wie nach der Stufe 3.

Tipp Die Methode storeAsURL() kennen Sie aus der Menüoption Datei | Speichern unter..., wodurch der aktuelle Speicherort geändert wird. Die Methode storeToURL() kommt normalerweise beim Export eines Dokuments zur Anwendung, so dass sich der Speicher-URL der Datei nicht ändert und er keine OOo-fremde Namenserweiterung erhält.

Die beiden Objektmethoden zum Speichern eines Dokuments – storeAsURL() und storeToURL() – verwenden dieselben Argumente. Lernen Sie, eine zu benutzen, und sie können es auch mit der anderen.

```
ThisComponent.storeAsURL(url, args())
ThisComponent.storeToURL(url, args())
```

Das zweite Argument ist ein Array von Eigenschaftswerten (s. Tabelle 117), die bestimmen, wie das Dokument gespeichert wird (s. Listing 306). Die Dateien können auch genauso gut ohne Argumente gespeichert werden (s. Listing 307).

Listing 306. *Speichert ein Dokument an einem neuen Ort.*

```
Dim args(0) As New com.sun.star.beans.PropertyValue
Dim sUrl As String
sUrl = "file:///c:/My%20Documents/test_file.odt"
args(0).Name = "Overwrite"  'Diese Eigenschaft ist in der Tabelle 117 beschrieben.
args(0).Value = False       'Ein existierendes Dokument wird nicht überschrieben.
ThisComponent.storeAsURL(sUrl, args())
```

Tipp Das Interface com.sun.star.frame.XComponentLoader definiert die Objektmethode loadComponentFromUrl(), mit der eine Datei geöffnet wird. Die verschiedenen Dokumenttypen binden dieses Interface nicht ein, wohl aber sowohl der Dokument-Frame als auch der Desktop. Die Methode loadComponentFromUrl() nutzt auch die Werte der Tabelle 117, um zu bestimmen, wie die Datei geöffnet wird.

Listing 307. *Speichert das Dokument mit einer unpassenden Namenserweiterung.*

```
ThisComponent.storeToURL("file:///c:/two.xls", Array())
```

Achtung Das Makro im Listing 307 verwendet die Namenserweiterung „xls“, die normalerweise von Microsoft Excel benutzt wird. Die Datei wird dadurch aber nicht im Microsoft-Excel-Format gespeichert. Sie behält das Standard-OOo-Dateiformat bei, wenn kein Exportfilter angegeben ist.

Wenn Sie eine Datei öffnen, prüft OOo, ob die Datei das Standard-OOo-Dateiformat hat. Wenn nicht, wird der Dateityp basierend auf der Namenserweiterung bestimmt. Ich kann nicht mehr zählen, wie oft ich gefragt wurde, warum OOo keine komma-separierte Textdatei öffnen kann. Die übliche

Antwort ist, dass eine solche Datei die Namenserweiterung CSV haben muss, ansonsten kann sie OOo nicht erkennen. Obwohl die Namenserweiterung wichtig ist, wenn eine Datei aus dem GUI geöffnet wird, so spielt sie beim Speichern keine Rolle. Wenn Sie eine Datei in einem vom originären OOo abweichenden Format speichern wollen, müssen sie OOo das andere Dateiformat explizit mitteilen (s. Listing 308).

Listing 308. *Exportiert ein Dokument in das angegebene Microsoft-Excel-Dateiformat.*

```
Dim args(0) As New com.sun.star.beans.PropertyValue
args(0).Name = "FilterName"          'Welcher Filter soll es denn wohl sein?
args(0).Value = "MS Excel 97"        'Aha, das Excel-97-Format!
ThisComponent.storeToURL("file:///c:/one.xls",args())
```

Tipp Obwohl die Filternamen für den Export dieselben sind wie für den Import, können nicht alle Importfilter exportieren und umgekehrt auch nicht alle Exportfilter importieren.

Impress und Draw, mit denen grafische Inhalte bearbeitet werden, unterstützen multiple Folien. Der Export einer Folie in ein spezifisches grafisches Format erfordert einen Grafik-Exportfilter (s. Listing 309). Ohne die Angabe des Medientyps wird der Export aber scheitern.

Listing 309. *Exportiert die erste Folie als JPG-Datei.*

```
Dim oFilter
Dim args(1) As New com.sun.star.beans.PropertyValue
oFilter = CreateUnoService("com.sun.star.drawing.GraphicExportFilter")
oFilter.setSourceDocument(ThisComponent.drawPages(0))
args(0).Name = "URL"                  'Ort, an dem die Datei gespeichert wird
args(0).Value = "file:///c:/one.JPG" 'Der Zielpfad als URL
args(1).Name = "MediaType"            'Welcher Dateityp?
args(1).Value = "image/jpeg"          'Dieser Dateityp
oFilter.filter(args())
```

13.12. Bearbeitung von Formatvorlagen

Formatvorlagen sind Mechanismen, Formatinformationen zu gruppieren. Eine Absatzvorlage zum Beispiel definiert die Schriftart, die Zeichengröße, die Randeinstellungen und noch viele andere Formatoptionen. Die Änderung einer Formatvorlage bewirkt eine Änderung aller Objekte, die diese Vorlage nutzen. Das Interface com.sun.star.style.XStyleFamiliesSupplier bietet Zugang zu den von einem Dokument verwendeten Formatvorlagen. Das Makro im Listing 310 gibt alle Formatvorlagen des aktuellen Dokuments aus. Bild 94 zeigt die Vorlagen in einem meiner Dokumente.

Listing 310. *Gibt die in einem Dokument verwendeten Vorlagen aus.*

```
Sub DisplayAllStyles
  Dim oFamilies     'Familien mit Interface com.sun.star.container.XNameAccess
  Dim oFamilyNames  'Namen der Familientypen. Stringarray
  Dim oStyleNames   'Namen der Vorlagen. Stringarray
  Dim oStyles       'Vorlagen mit Interface com.sun.star.container.XNameAccess
  Dim oStyle        'Eine einzelne Vorlage
  Dim s As String   'Ausgabestring
  Dim n As Integer  'Indexvariable
  Dim i As Integer  'Indexvariable

  oFamilies = ThisComponent.StyleFamilies
  oFamilyNames = oFamilies.getElementNames()

  REM Als erstes werden die Vorlagenfamilien und
  REM die Anzahl der Vorlagentypen ausgegeben.
  For n = LBound(oFamilyNames) To UBound(oFamilyNames)
    oStyles = oFamilies.getByName(oFamilyNames(n))
```

```
    s = s & oStyles.getCount() & " " & oFamilyNames(n) & Chr$(10)
  Next
  MsgBox s, 0, "Vorlagenfamilien"

  REM Nun werden alle einzelnen Vorlagennamen ausgegeben.
  For n = LBound(oFamilyNames) To UBound(oFamilyNames)
    s = ""
    oStyles = oFamilies.getByName(oFamilyNames(n))
    oStyleNames = oStyles.getElementNames()
    For i = LBound(oStyleNames) To UBound (oStyleNames)
      s = s + i + " : " + oStyleNames(i) + Chr$(10)
      If ((i + 1) Mod 30 = 0) Then
        MsgBox s, 0, oFamilyNames(n)
        s = ""
      End If
    Next i
    If Len(s) <> 0 Then MsgBox s, 0, oFamilyNames(n)
  Next n
End Sub
```

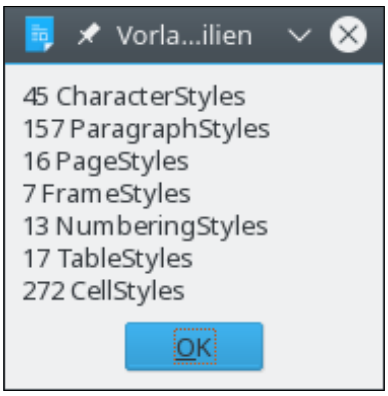


Bild 94. *Die Vorlagenfamilien in diesem Textdokument.*

Die verschiedenen Dokumenttypen enthalten unterschiedliche Vorlagentypen. Bild 94 zeigt die Vorlagenfamilien eines Textdokuments. Calc-Dokumente haben die Vorlagenfamilien CellStyles und PageStyles, Impress-Dokumente die Familien Graphics und Default, und Draw-Dokumente die Familie Graphic. Obwohl sich die Vorlagentypen unterscheiden, haben sie doch Gemeinsamkeiten. Zum Beispiel bindet jede Vorlage sowohl den Service com.sun.star.style.Style als auch das Interface com.sun.star.style.XStyle ein. Die gemeinsamen Methoden und Eigenschaften stellen sehr rudimentäre Funktionalitäten zur Verfügung (s. Tabelle 120).

Tabelle 120. *Methoden und Eigenschaften im Service com.sun.star.style.Style.*

Methode oder Eigenschaft	Beschreibung
isUserDefined()	Ist diese Vorlage benutzerdefiniert? Wenn nicht, ist sie von OOo bereitgestellt.
isInUse()	Wird diese Vorlage im Dokument verwendet?
getParentStyle()	Der Name der elterlichen Vorlage. Kann ein leerer String sein.
setParentStyle(name)	Legt die elterliche Vorlage fest.
IsPhysical	Wurde die Vorlage physisch erzeugt?
FollowStyle	Name der Vorlage für den Folgeabsatz. Bei einer Überschriftsvorlage hätte ich gerne den nächsten Absatz als normalen Text.
DisplayName	Name der Vorlage zur Angabe in der Benutzeroberfläche.

Methode oder Eigenschaft	Beschreibung
IsAutoUpdate	Wenn die Eigenschaften eines Objekts, das diese Vorlage nutzt, geändert werden (zum Beispiel ein Wechsel der Schriftart), sollen diese Änderungen automatisch auch in die Vorlage übernommen werden?

Tabelle 120 zeigt Methoden und Eigenschaften, mit denen die allgemeine Frage beantwortet werden kann: „Wie erhalte ich eine Liste der Formatvorlagen, die momentan von einem Dokument genutzt werden?“ Siehe Listing 311 und Bild 95.

Tipp Listing 310 greift auf die Formatvorlagen über ihren Namen, Listing 311 über ihren Index zu. Beachten Sie, dass wenn getCount() 10 zurückgibt, 10 Elemente enthalten sind, auf die man mit dem Indexwert 0 bis 9 zugreift.

Listing 311. *Ausgabe aller genutzten Absatzvorlagen.*

```
Sub DisplayAllUsedParagraphStyles
  Dim oStyles       'Vorlagen mit Interface com.sun.star.container.XNameAccess
  Dim oStyle        'Eine einzelne Vorlage
  Dim s As String   'Ausgabestring
  Dim i As Integer  'Indexvariable

  oStyles = ThisComponent.StyleFamilies.getByName("ParagraphStyles")

  REM Wenn getCount() 10 Vorlagen meldet, heißt das von 0 bis 9 oder von 1 bis 10.
  For i = 1 To oStyles.getCount()
    oStyle = oStyles.getByIndex(i - 1)
    If oStyle.isInUse() Then s = s & oStyle.DisplayName & Chr$(10)
  Next
  MsgBox s, 0, "Verwendete Absatzvorlagen"
End Sub
```

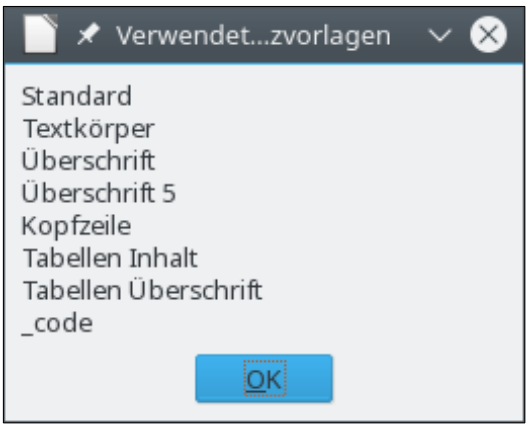


Bild 95. *In einem Textdokument verwendete Absatzvorlagen.*

Anfangs war ich von den Ergebnissen wie im Bild 95 verwirrt, denn es wurden Absatzvorlagen angezeigt, die ich für unbenutzt hielt. Ich nahm an, dass ich im Dokument einen Fehler gemacht hätte, und startete eine Suche nach diesen irrtümlich verwendeten Vorlagen über den Dialog Suchen & Ersetzen (Menü Bearbeiten). Wenn man das Kontrollkästchen „Suche nach Vorlagen“ (erreichbar nach einem Klick auf „Mehr Optionen“) aktiviert, wird im Feld „Suchen nach“ eine Aufklappliste aller in diesem Dokument genutzten Vorlagen angeboten. Mit dieser Methode suchte ich nach der Absatzvorlage „Standard“ (s. Bild 95), die aber nicht gefunden wurde, genauso wenig wie „Überschrift“. Nach kurzer Verwirrung – ungefähr fünf Minuten – erkannte ich, dass ich ganz und gar nicht auf einen neuen Bug gestoßen war, sondern ein interessantes Verhalten entdeckt hatte, das mir vorher nicht aufgefallen war. Wenn in einem Dokument eine Vorlage genutzt wird, so wird gleichermaßen die mit ihr verknüpfte Vorlage mit angezeigt, auch wenn sie nicht direkt genutzt wird. In meinem

Dokument ist zum Beispiel „Textkörper“ mit „Standard“ verknüpft, und „Überschrift 5“ ist mit „Überschrift“ verknüpft.

Die verschiedenen Vorlagentypen enthalten die für ihren Typ passenden Methoden und Eigenschaften. Die zu Vorlagen gehörenden allgemeinen Services, Interfaces, Structs und Konstanten finden Sie unter den Adressen
AOO: http://api.openoffice.org/docs/common/ref/com/sun/star/style/module-ix.html
LO: https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1style.html

Die dort angezeigten Typen bilden die Basis, auf der andere Vorlagentypen aufgebaut sind. Zum Beispiel haben die beiden Services com.sun.star.text.TextPageStyle und com.sun.star.sheet.TablePageStyle gleichermaßen den Service com.sun.star.style.PageStyle als Grundlage. Um ein Gefühl für eine Vorlage zu bekommen, lohnt es sich häufig, mit der Inspizierung des Objekts zu beginnen.

```
MsgBox  vObj.dbg_methods
MsgBox  vObj.dbg_supportedInterfaces
MsgBox  vObj.dbg_properties
```

Die Vorlagenobjekte binden wie so viele andere Objekte das Interface XPropertySet ein. Das Makro im Listing 312 gibt mit Hilfe dieses Interface eine Liste von Eigenschaften aus, die in der Absatzvorlage „OOoTextBody“ stecken. Listing 312 zeigt nicht die Werte der einzelnen Eigenschaften, nur deren Namen. Als interessante Übung können Sie ja das Listing 312 so modifizieren, dass der Wert jeder Eigenschaft angezeigt wird, vorausgesetzt, es ist ein Standard-Datentyp – eine Eigenschaft kann auch aus einem komplexen Objekt bestehen.

Listing 312. *Ausgabe der Eigenschaften einer Absatzvorlage.*

```
Sub StyleProperties
  Dim oStyles       'Vorlagen mit Interface com.sun.star.container.XNameAccess
  Dim s As String   'Ausgabestring
  Dim i As Integer  'Indexvariable
  Dim Props         'Array von Eigenschaften
  REM Jede Vorlage unterstützt das Interface com.sun.star.beans.XPropertySet,
  REM das die Methode getPropertySetInfo() bereitstellt, die
  REM eine Enumeration der enthaltenen Eigenschaften ermöglicht.
  REM getProperties gibt ein Array von com.sun.star.beans.Property-Werten zurück.
  oStyles = ThisComponent.StyleFamilies.getByName("ParagraphStyles")
  Props = oStyles.getByName("OOoTextBody").getPropertySetInfo().getProperties()

  For i = 0 To UBound(Props)          'Für jede einzelne Eigenschaft
    s = s & Props(i).Name & Chr$(10) 'Der Name der Eigenschaft und ein Zeilenumbruch
    If (i + 1) Mod 30 = 0 Then              'Wenn der String zu lang wird,
      MsgBox s, 0, "Vorlageneigenschaften"  'wird er ausgegeben
      s = ""                               'und die Liste neu begonnen.
    End If
  Next
  REM Für den Fall, dass die gesamte Liste noch nicht ausgegeben wurde.
  If Len(s) <> 0 Then MsgBox s, 0, "Vorlageneigenschaften"
End Sub
```

Nach meiner Erfahrung muss man nur selten Vorlagen inspizieren. Noch seltener ist der Fall, dass man eine Vorlage aus einem Makro heraus modifiziert. Und doch kommt es vor. Zum Beispiel wird die Seitengröße von der aktuellen Seitenvorlage bestimmt. Mit dem aktuellen Controller finden Sie die aktuelle Seitenvorlage, aus der Sie die Seitenabmessungen auslesen. Listing 313 gibt die Seitengröße, die Randeinstellungen und die aktuelle Cursorposition auf der Seite aus. Bild 96 zeigt das Ergebnis.

Tipp Der häufigste Grund, den ich zur Modifizierung einer Seitenvorlage gesehen habe, ist, der aktuellen Seitenvorlage eine Kopf- oder Fußzeile hinzuzufügen.

Teile des folgenden Codes sind spekulativ und haben sich geändert, seit ich den Code ursprünglich schrieb. Der aktuelle Controller gibt die aktuelle Cursorposition auf der Basis der linken oberen Ecke der ersten Seite des Dokuments aus. Auf der Seite 200 wird daher die y-Koordinate sehr groß sein. Der folgende Code versucht, das zu kompensieren, tut es aber nur dann, wenn alle benutzten Seitenvorlagen dieselben Höhenmaße verwenden. Dabei ist zu beachten, dass der Y-Wert der Cursorposition sich nicht nur über die Summe der Einzelseiten aufbaut, sondern offenbar auch den schmalen Steg zwischen den Seiten einbezieht, dessen Höhe anscheinend von der verwendeten Seitengröße abhängt. In diesem Dokument (Seitengröße: DINA4) ist der Steg 5,013 mm hoch (herausgefunden durch Ausprobieren), im amerikanischen Original (Seitengröße: Letter) 5,0093 mm. Der X-Wert der Cursorposition ist abhängig vom aktuellen Maßstab der Dokumentdarstellung in Bezug auf die aktuelle Fensterbreite. Er liefert nur dann den korrekten Wert, wenn die Dokumentbreite größer oder gleich der Fensterbreite ist.

Listing 313. *Ausgabe der Seiteninformationen.*

```
Sub PrintPageInformation
  Dim oViewCursor     'Aktueller Viewcursor
  Dim oStyle          'Aktuelle Seitenvorlage
  Dim lHeight As Long 'Seitenhöhe aus der Seitenvorlage in 1/100 mm
  Dim lWidth  As Long 'Seitenbreite aus der Seitenvorlage in 1/100 mm
  Dim s As String     'Ausgabestring
  REM Der aktuelle Controller bildet die Schnittstelle mit einem Menschen - mit Ihnen!
  REM Nun, ich hoffe jedenfalls, Sie sind ein solcher.
  REM Holt den aktuellen Viewcursor vom Controller. Das ist das Ding,
  REM das weiß, wo sich der aktuelle Cursor letzten Endes befindet.
  oViewCursor = ThisComponent.CurrentController.getViewCursor()

  REM Dieser Viewcursor kennt eine Menge, auch die aktuelle Seitenvorlage.
  REM Mit dem Namen der Seitenvorlage erhalten Sie eine Referenz
  REM auf die aktuelle Seitenvorlage.
  s = oViewCursor.PageStyleName
  oStyle = ThisComponent.StyleFamilies.getByName("PageStyles").getByName(s)
  s = "Seitenvorlage = " & s & Chr$(10)

  lHeight = oStyle.Height  'Seitenhöhe in 1/100 mm
  lWidth  = oStyle.Width   'Seitenbreite in 1/100 mm

  REM Seitenabmessungen in mm, Zoll und Pica.
  s = s & "Die Seitenabmessungen sind " & Chr$(10) &_
      "     " & Format(lWidth / 100.0, "##,##0.#") & " x " &_
      Format(lHeight / 100.0, "##,##0.#") & " mm" & Chr$(10) &_
      "     " & Format(lWidth / 2540.0, "##,##0.#") & " x " &_
      & Format(lHeight / 2540.0, "##,##0.#") & " Zoll" & Chr$(10) &_
      "     " & Format(lWidth * 72.0 / 2540.0, "##,##0.#") & " x " &_
      & Format(lHeight * 72.0 / 2540.0, "##,##0.#") & " Pica" & Chr$(10)

  Dim dCharHeight As Double 'Zeichengröße in mm
  Dim iCurPage As Integer   'Aktuelle Seite

  Dim dXCursor As Double    'Abstand des Cursors von der linken Kante in mm
  Dim dYCursor As Double    'Abstand des Cursors von der oberen Kante in mm
  Dim dXRight  As Double    'Abstand des Cursors von der rechten Kante in mm
  Dim dYBottom As Double    'Abstand des Cursors von der unteren Kante in mm
  Dim dBottom As Double     'Unterer Seitenrand in mm
```

```
    Dim dLeft   As Double     'Linker Seitenrand in mm
    Dim dRight  As Double     'Rechter Seitenrand in mm
    Dim dTop    As Double     'Oberer Seitenrand in mm

    dCharHeight = oViewCursor.CharHeight / 72.0 * 25.4 'Umrechnung Punkte in mm
    iCurPage = oViewCursor.getPage()                   'Seitenzahl
    s = s & "Aktuelle Seite = " & iCurPage & Chr$(10)

    dBottom = oStyle.BottomMargin / 100.0 : dLeft = oStyle.LeftMargin / 100.0
    dRight  = oStyle.RightMargin  / 100.0 : dTop  = oStyle.TopMargin  / 100.0
    s = s & "Rand (in mm): Links = " & Format(dLeft, "##,##0.#") &_
            "  Rechts = " & Format(dRight, "##,##0.#") & Chr$(10)
    s = s & "Rand (in mm): Oben = " & Format(dTop, "##,##0.#") &_
            "  Unten = " & Format(dBottom, "##,##0.#") & Chr$(10)

    Dim v
    REM Die Koordinaten des Cursors bezogen auf die obere linke Ecke der Seite.
    REM Der Rückgabetyp ist com.sun.star.awt.Point
    REM Die Einheit ist in 1/100 mm.
    v = oViewCursor.getPosition()

    REM Die Seitenzahl vom Viewcursor schließt "Phantom"-Seiten ein, wenn zwei
    REM aufeinander folgende Seiten beide, sagen wir, rechte Seiten sind.
    REM v.Y schließt das nicht ein.
    Dim realPageNumber As Long
    Dim dInterPageSpace As Double  'Der schmale Streifen zwischen den Seiten

    dInterPageSpace = 501.3  '1/100 mm
    realPageNumber = Fix(v.Y / (lHeight + dInterPageSpace))

    Dim realY : realY = Fix(v.Y - realPageNumber * (lHeight + iInterPageSpace))

    REM Die Cursorposition ist der Abstand zum Seitenrand plus der Seitenrand.
    dYCursor = realY / 100.0 + dTop
    dYBottom = (lHeight - realY) / 100.0 - dTop
    dXCursor = v.X / 100.0 + dLeft
    dXRight  = (lWidth - v.X) / 100.0 - dLeft

    s = s & "Der Cursor steht " & Format(dXCursor, "0.#") & " mm von links " & Chr$(10)
    s = s & "Der Cursor steht " & Format(dXRight, "0.#") & " mm von rechts " & Chr$(10)
    s = s & "Der Cursor steht " & Format(dYCursor, "0.#") & " mm von oben " & Chr$(10)
    s = s & "Der Cursor steht " & Format(dYBottom, "0.#") & " mm von unten " & Chr$(10)
    s = s & "Zeichengröße = " & Format(dCharHeight, "0.##") & " mm" & Chr$(10)
    MsgBox s, 0, "Seiteninformationen"
End Sub
```

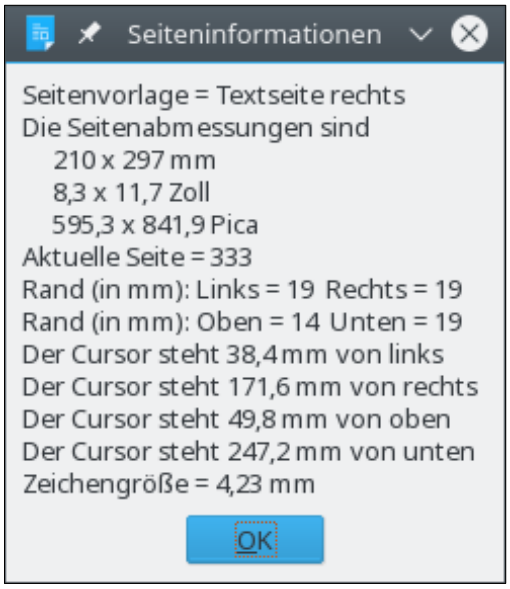


Bild 96. *Informationen aus der Seitenvorlage.*

Calc-Dokumente sind aus Tabellenblättern zusammengesetzt. Jedes Blatt kann eine andere Seitenvorlage haben. Das Makro im Listing 313 holt die aktuelle Seitenvorlage mit Hilfe des Viewcursors. Bei Calc-Dokumenten holen Sie die Seitenvorlage aus dem aktiven Tabellenblatt.

```
REM Die Seitenvorlage erhalten Sie aus dem gerade aktiven Tabellenblatt.
REM In einem Calc-Dokument weiß der aktuelle Controller, welches Tabellenblatt
REM gerade aktiv ist.
Print "Style = " & ThisComponent.CurrentController.getActiveSheet().PageStyle
```

13.12.1. Nützliche Helfer für Formatvorlagen

Obwohl es einfach ist, Formatvorlagen mit Makros zu modifizieren, können doch kleine Details große Probleme bewirken, beispielsweise eine Absatzvorlage zu spezifizieren, die auf dem aktuellen Rechner gar nicht vorhanden ist.

Listing 314. *Prüft, ob ein Dokument eine Absatzvorlage enthält.*

```
Function DocHasParStyle(oDoc, sName$) As Boolean
  Dim oStyles
  oStyles = oDoc.StyleFamilies.getByName("ParagraphStyles")
  DocHasParStyle() = oStyles.hasByName(sName)
End Function
```

Die Überprüfung einer Zeichenvorlage ist gleichermaßen trivial.

Listing 315. *Prüft, ob ein Dokument eine Zeichenvorlage enthält.*

```
Function DocHasCharStyle(oDoc, sName$) As Boolean
  Dim oStyles
  oStyles = oDoc.StyleFamilies.getByName("CharacterStyles")
  DocHasCharStyle() = oStyles.hasByName(sName)
End Function
```

Um herauszufinden, ob in einem Dokument eine bestimmte Schriftart zur Verfügung steht, prüfen Sie die Schriftartdeskriptoren aus dem Containerfenster.

Listing 316. *Prüft, ob eine Schriftart in einem Dokument zur Verfügung steht.*

```
Function DocHasFontName(oDoc, sName$) As Boolean
  Dim oWindow
  Dim oFonts()
  Dim i%
```

```
  oWindow  = oDoc.getCurrentController().getFrame().getContainerWindow()
  oFonts() = oWindow.getFontDescriptors()
  For i = LBound(oFonts()) To UBound(oFonts())
    If oFonts(i).Name = sName Then
      DocHasFontName() = True
      Exit Function
    End If
  Next
  DocHasFontName() = False
End Function
```

Eine Property ist ein Struct mit einem Namen und einem Wert. Das folgende Makro akzeptiert einen Namen und einen Wert und gibt eine Property mit Namen und Wert zurück.

Listing 317. *Erzeugt eine Property mit dem angegebenen Namen und dem angegebenen Wert.*

```
'*************************************************************
'** Erzeugt ein PropertyValue Struct und gibt ihn zurück.
'*************************************************************
Function CreateProperty(Optional cName As String, Optional uValue)
                                        As com.sun.star.beans.PropertyValue
   Dim oPropertyValue As New com.sun.star.beans.PropertyValue
   If Not IsMissing(cName) Then
      oPropertyValue.Name = cName
   End If
   If Not IsMissing(uValue) Then
      oPropertyValue.Value = uValue
   End If
   CreateProperty() = oPropertyValue
End Function
```

Listing 317 bietet die Möglichkeit, Properties direkt in einem Array zu erzeugen.

Listing 318. *Erzeugt Properties, die zur Erzeugung einer Zeichenvorlage genutzt werden.*

```
  REM Basisvorlage für alle.
  REM Computercode, der nicht farbig kodiert ist und in normalem Text vorkommt,
  REM nutzt diese Vorlage.
  oProps() = Array(CreateProperty("CharFontName", sFontName), _
    CreateProperty("CharColor", RGB(0, 0, 0)), _
    CreateProperty("CharNoHyphenation", True))     'Keine automatische Silbentrennung

  CreateCharacterStyle("OOoComputerCode", oProps())

  REM Basisvorlage für normale Listings.
  oProps() = Array(CreateProperty("ParentStyle", "OOoComputerCode"))
  CreateCharacterStyle("_OOoComputerBase", oProps())
```

Der schwierige Part ist zu entscheiden, welche Eigenschaften zu setzen sind und welche nicht. Es hilft sehr, einmal eine manuell erzeugte Vorlage mit den interessierenden Werten zu inspizieren. Eine Eigenschaft, die nicht explizit gesetzt ist, erbt ihren Wert von der elterlichen Vorlage oder dem existierenden Format. Wenn Sie zum Beispiel eine Zeichenvorlage auf fett setzen, aber weder Schriftart noch Schriftgröße angeben, wird eine Zeichenvorlage erzeugt, die weder die Schriftart noch die Schriftgröße beeinflusst, sondern nur die Zeichen auf fett setzt.

Listing 319. *Zeichenvorlagen, die zur Formatierung der Codebeispiele genutzt werden.*

```
'*************************************************************
'** Erzeugt Zeichenvorlagen für StarBasic mit denselben Farben
'** wie die OOo IDE.
```

```
'*************************************************************
Function CreateStarBasicCharStyles()
  Dim oProps()

  REM Wenn Sie keine Rechtschreibprüfung wollen, setzen Sie
  REM die Zeicheneigenschaft CharLocale auf noLocale.
  Dim noLocale As New com.sun.star.lang.Locale
  noLocale.Country = ""
  noLocale.Language = "zxx"

  If Not CreateBaseCharStyles() Then
    CreateStarBasicCharStyles() = False
    Exit Function
  End If

  oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharColor", RGB(76, 76, 76)))
  CreateCharacterStyle("_OOoComputerComment", oProps())

  oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharColor", RGB(255, 0, 0)))
  CreateCharacterStyle("_OOoComputerLiteral", oProps())

  oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharLocale", noLocale), _
    CreateProperty("CharColor", RGB(0, 0, 128)))
  CreateCharacterStyle("_OOoComputerKeyWord", oProps())

  oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerBase"), _
    CreateProperty("CharColor", RGB(0, 128, 0)))
  CreateCharacterStyle("_OOoComputerIdent", oProps())
  CreateStarBasicCharStyles() = True
End Function
```

Das folgende Makro erzeugt eine Zeichenvorlage, falls sie nicht existiert. Es wird gesondert geprüft, ob es die übergeordnete Vorlage gibt, denn sie muss vor der Kindvorlage erzeugt werden.

Listing 320. *Erzeugt eine Zeichenvorlage, falls sie nicht existiert.*

```
Sub CreateCharacterStyle(sStyleName$, oProps())
  Dim i%
  Dim oFamilies
  Dim oStyles
  Dim oStyle

  oFamilies = ThisComponent.StyleFamilies
  oStyles = oFamilies.getByName("CharacterStyles")
  If oStyles.HasByName(sStyleName) Then
    Exit Sub
  End If
  oStyle = ThisComponent.createInstance("com.sun.star.style.CharacterStyle")
  'Überträgt die als Argument oProps übergebenen Eigenschaften auf die neue Vorlage.
  For i = LBound(oProps) To UBound(oProps)
    If oProps(i).Name = "ParentStyle" Then
      If oStyles.HasByName(oProps(i).Value) Then
        oStyle.ParentStyle = oProps(i).Value
      Else
        Print "Die übergeordnete Zeichenvorlage (" & oProps(i).Value & _
              ") existiert nicht. Sie wird ignoriert."
```

```
      End If
      oStyle.ParentStyle = oProps(i).Value
    Else
      oStyle.setPropertyValue(oProps(i).Name, oProps(i).Value)
    End If
  Next
  oStyles.insertByName(sStyleName, oStyle)
End Sub
```

Eine Absatzvorlage ist ein wenig komplizierter, weil einige Elemente besondere Beachtung brauchen, zum Beispiel das Setzen von Tabulatorpositionen.

Listing 321. *Erzeugt eine Absatzvorlage, falls sie nicht existiert.*

```
Sub CreateParStyle(sStyleName$, oProps())
  Dim i%, j%
  Dim oFamilies
  Dim oStyles
  Dim oStyle
  Dim tabStops%

  oFamilies = ThisComponent.StyleFamilies
  oStyles = oFamilies.getByName("ParagraphStyles")
  If oStyles.HasByName(sStyleName) Then
    Exit Sub
  End If
  oStyle = ThisComponent.createInstance("com.sun.star.style.ParagraphStyle")

  'Überträgt die als Argument oProps übergebenen Eigenschaften auf die neue Vorlage.
  For i = LBound(oProps) To UBound(oProps)
    If oProps(i).Name = "ParentStyle" Then
      If oStyles.HasByName(oProps(i).Value) Then
        oStyle.ParentStyle = oProps(i).Value
      Else
        Print "Die übergeordnete Absatzvorlage  (" & oProps(i).Value & _
              ") existiert nicht. Sie wird ignoriert."
      End If
    ElseIf oProps(i).Name = "ParaTabStops" Then
      tabStops = oProps(i).Value  'Struct com.sun.star.style.TabStop
      Dim tab(0 To 19) As New com.sun.star.style.TabStop     '20 sollten reichen.
      For j = LBound(tab) To UBound(tab)
        tab(j).Alignment = com.sun.star.style.TabAlign.LEFT  'Ausrichtung
        tab(j).DecimalChar = Asc(".")                        'Dezimaltrenner
        tab(j).FillChar = 32                                 'Füllzeichen
        tab(j).Position = (j + 1) * tabStops                 'Position vom linken Rand
      Next
      oStyle.ParaTabStops = tab
    ElseIf oProps(i).Name = "FollowStyle" Then               'Folgevorlage
      If oStyles.HasByName(oProps(i).Value) Or oProps(i).Value = sStyleName Then
        oStyle.setPropertyValue(oProps(i).Name, oProps(i).Value)
      Else
        Print "Die Folgevorlage (" & oProps(i).Value & _
              ") existiert nicht. Sie wird ersetzt durch die Vorlage " & sStyleName
      End If
    Else
      oStyle.setPropertyValue(oProps(i).Name, oProps(i).Value)
    End If
```

```
    Next
    oStyles.insertByName(sStyleName, oStyle)
End Sub
```

Dieser Code erzeugt die Hauptabsatzvorlagen zur Codeformatierung in diesem Dokument.

Listing 322. *Properties zur Erzeugung einer Absatzvorlage.*

```
REM Tabulatorpositionen werden in der Absatzvorlage gesetzt.
' 1/4 Zoll
tabStopLoc% = 2540 / 4

oProps() = Array(CreateProperty("ParaTopMargin", CLng(0)), _
  CreateProperty("ParaBottomMargin", CLng(2540 * 0.03)), _
  CreateProperty("ParaLeftMargin", CLng(2540 * 0.20)), _
  CreateProperty("ParaRightMargin", CLng(0)), _
  CreateProperty("ParaFirstLineIndent", CLng(0)), _
  CreateProperty("CharFontName",sFontName), _
  CreateProperty("ParaTabStops", tabStopLoc), _
  CreateProperty("ParaLineNumberCount", False), _
  CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
  CreateProperty("CharAutoKerning", False), _
  CreateProperty("CharHeight", fParNormalCharHeight))
CreateParStyle("_OOoComputerCode", oProps())

oProps() = Array(CreateProperty("ParentStyle", "_OOoComputerCode"), _
  CreateProperty("ParaTopMargin", CLng(0)), _
  CreateProperty("ParaBottomMargin", CLng(2540 * 0.10)), _
  CreateProperty("ParaLeftMargin", CLng(2540 * 0.20)), _
  CreateProperty("ParaRightMargin", CLng(0)), _
  CreateProperty("ParaFirstLineIndent", CLng(0)), _
  CreateProperty("CharFontName",sFontName), _
  CreateProperty("ParaTabStops", tabStopLoc), _
  CreateProperty("ParaLineNumberCount", False), _
  CreateProperty("WritingMode", com.sun.star.text.WritingMode.LR_TB), _
  CreateProperty("CharAutoKerning", False), _
  CreateProperty("CharHeight", fParNormalCharHeight), _
  CreateProperty("FollowStyle", sNextStyle))
CreateParStyle("_OOoComputerCodeLastLine", oProps())
```

13.13. Der Umgang mit dem Gebietsschema (Locale)

Ein Gebietsschema (englisch Locale) beschreibt eine bestimmtes geografisches, politisches oder kulturelles Gebiet. Zahlen- und Datumsangaben sind gebietsabhängig und daher auch mit einem Gebietsschema verknüpft. Über das Menü **Extras | Optionen | Spracheinstellungen | Sprachen** sehen Sie, welches Gebietsschema Sie für Ihren Rechner nutzen. Ein Gebietsschema ist leicht erstellt.

```
Dim aLocale As New com.sun.star.lang.Locale
aLocale.Language = "fr"       'Sprache
aLocale.Country = "FR"        'Land
```

Tipp	OOo mag wohl nicht jedes mögliche Gebietsschema unterstützen, wird aber versuchen, die bestmögliche Übereinstimmung zu finden.

Das Gebietsschema beruht sowohl auf der Sprache als auch auf dem Land. In manchen Ländern werden mehrere Sprachen gesprochen wie auch manche Sprachen in mehreren Ländern. Tabelle 121 enthält den aus zwei Buchstaben bestehenden Sprachencode und Tabelle 122 den auch aus zwei Buchstaben bestehenden Ländercode.

Tipp Obwohl die Groß- und Kleinschreibung für die Codes im Gebietsschema keine Rolle spielt, wird der Sprachencode üblicherweise in Kleinbuchstaben, der Ländercode in Großbuchstaben geschrieben.

Tabelle 121. *Sprachencode, alphabetisch nach dem Code sortiert.*

Code	Sprache	Code	Sprache	Code	Sprache
aa	Afar	ab	Abchasisch	af	Afrikaans
am	Amharisch	ar	Arabisch	as	Assamesisch
ay	Aymara	az	Aserbaidschanisch	ba	Baschkirisch
be	Weißrussisch	bg	Bulgarisch	bh	Bihari
bi	Bislama	bn	Bengalisch; Bangla	bo	Tibetisch
br	Bretonisch	ca	Katalanisch	co	Korsisch
cs	Tschechisch	cy	Walisisch; Kymrisch	da	Dänisch
de	Deutsch	dz	Bhutani; Dzongkha	el	Griechisch
en	Englisch	eo	Esperanto	es	Spanisch
et	Estnisch	eu	Baskisch	fa	Persisch; Farsi
fi	Finnisch	fj	Fidschi	fo	Färöisch
fr	Französich	fy	Friesisch	ga	Irisch
gd	Scots-Gälisch	gl	Galicisch	gn	Guarani
gu	Gujarati	ha	Hausa	he	Hebräisch (früher iw)
hi	Hindi	hr	Kroatisch	hu	Ungarisch
hy	Armenisch	ia	Interlingua	id	Indonesisch (früher in)
ie	Interlingue	ik	Inupiaq	is	Isländisch
it	Italienisch	iu	Inuktitut	ja	Japanisch
jw	Javanisch	ka	Georgisch	kk	Kasachisch
kl	Grönländisch	km	Kambodschanisch	kn	Kannada
ko	Koreanisch	ks	Kashmiri	ku	Kurdisch
ky	Kirgisisch	la	Lateinisch	ln	Lingála
lo	Laotisch	lt	Litauisch	lv	Lettisch
mg	Malagasy	mi	Maori	mk	Mazedonisch
ml	Malayalam	mn	Mongolisch	mo	Moldauisch
mr	Marathi	ms	Malaiisch	mt	Maltesisch
my	Birmanisch	na	Nauruisch	ne	Nepali
nl	Niederländisch	no	Norwegisch	oc	Okzitanisch
om	Oromo	or	Oriya	pa	Panjabi
pl	Polnisch	ps	Paschtunisch	pt	Portugiesisch
qu	Quechua	rm	Rätoromanisch	rn	Kirundi
ro	Rumänisch	ru	Russisch	rw	Kinyarwanda
sa	Sanskrit	sd	Sindhi	sg	Sango
sh	Serbokroatisch	si	Sinhal; Singhalesisch	sk	Slowakisch
sl	Slowenisch	sm	Samoisch	sn	Shona
so	Somali	sq	Albanisch	su	Sundanisch

Code	Sprache	Code	Sprache	Code	Sprache
ss	Siswati	st	Sesotho	sv	Schwedisch
sw	Swahili	ta	Tamil	te	Telugu
tg	Tadschikisch	th	Thai	ti	Tigrinya
tk	Turkmenisch	tl	Tagalog	tn	Setswana
to	Tonga	tr	Türkisch	ts	Tsonga; Xitsonga
tt	Tatar	tw	Twi	ug	Uigurisch
uk	Ukrainisch	ur	Urdu	uz	Usbekisch
vi	Vietnamesisch	vo	Volapük	wo	Wolof
xh	Xhosa	yi	Jiddisch (früher ji)	yo	Yoruba
za	Zhuang	zh	Chinesisch	zu	Zulu

Tabelle 122. *Ländercode, alphabetisch nach Ländern sortiert.*

Code	Land	Code	Land
AF	Afghanistan	EG	Ägypten
AX	Åland	AL	Albanien
DZ	Algerien	AS	Amerikanisch-Samoa
AD	Andorra	AO	Angola
AI	Anguilla	AQ	Antarktis
AG	Antigua und Barbuda	GQ	Äquatorialguinea
AR	Argentinien	AM	Armenien
AW	Aruba	AC	Ascension (Sankt Helena, Ascension und Tristan da Cunha)
AZ	Aserbaidschan	ET	Äthiopien
AU	Australien	BS	Bahamas
BH	Bahrain	BD	Bangladesch
BB	Barbados	BE	Belgien
BZ	Belize	BJ	Benin
BM	Bermuda	BT	Bhutan
BO	Bolivien	BA	Bosnien und Herzegowina
BW	Botsuana	BV	Bouvetinsel
BR	Brasilien	IO	Britisches Territorium im Indischen Ozean
BN	Brunei Darussalam	BG	Bulgarien
BF	Burkina Faso	BI	Burundi
CL	Chile	CN	China
CK	Cookinseln	CR	Costa Rica
DK	Dänemark	DE	Deutschland
DM	Dominica	DO	Dominikanische Republik
DJ	Dschibuti	EC	Ecuador
SV	El Salvador	CI	Elfenbeinküste
ER	Eritrea	EE	Estland
EU	Europäische Union	FK	Falklandinseln (Malwinen)
FO	Färöer	FJ	Fidschi

Code	Land	Code	Land
FI	Finnland	FX	France métropolitaine
FR	Frankreich	GF	Französisch-Guayana
PF	Französisch-Polynesien	TF	Französische Süd- und Antarktisgebiete
GA	Gabun	GM	Gambia
GE	Georgien	GH	Ghana
GI	Gibraltar	GD	Grenada
GR	Griechenland	GL	Grönland
GP	Guadeloupe	GU	Guam
GT	Guatemala	GG	Guernsey
GN	Guinea	GW	Guinea-Bissau
GY	Guyana	HT	Haiti
HM	Heard und die McDonaldinseln	HN	Honduras
HK	Hongkong	IN	Indien
ID	Indonesien	IQ	Irak
IR	Iran	IE	Irland
IS	Island	IM	Isle of Man
IL	Israel	IT	Italien
JM	Jamaika	JP	Japan
YE	Jemen	JE	Jersey
JO	Jordanien	YU	Jugoslawien
VI	Jungferninseln, Amerikanische	VG	Jungferninseln, Britische
KY	Kaimaninseln	KH	Kambodscha
CM	Kamerun	CA	Kanada
CV	Kap Verde	KZ	Kasachstan
QA	Katar	KE	Kenia
KG	Kirgisistan	KI	Kiribati
CC	Kokosinseln (Keelinginseln)	CO	Kolumbien
KM	Komoren	CD	Kongo, Demokratische Republik (früher Zaire)
CG	Kongo, Republik (Kongo-Brazzaville)	HR	Kroation (kroatisch: Hrvatska)
CU	Kuba	KW	Kuwait
LA	Laos	LS	Lesotho
LV	Lettland	LB	Libanon
LR	Liberia	LY	Libyen
LI	Liechtenstein	LT	Litauen
LU	Luxemburg	Mo	Macao
MG	Madagaskar	MW	Malawi
MY	Malaysia	MV	Malediven
ML	Mali	MT	Malta
MA	Marokko	MH	Marshallinseln

Code	Land	Code	Land
MQ	Martinique	MR	Mauretanien
MU	Mauritius	YT	Mayotte
MK	Mazedonien	MX	Mexiko
FM	Mikronesien, Föderierte Staaten von	MD	Moldawien
MC	Monaco	MN	Mongolei
ME	Montenegro	MS	Montserrat
MZ	Mosambik	MM	Myanmar
NA	Namibia	NR	Nauru
NP	Nepal	NC	Neukaledonien
NZ	Neuseeland	NI	Nicaragua
NL	Niederlande	AN	Niederländische Antillen
NE	Niger	NG	Nigeria
NU	Niue	KP	Nordkorea (Demokratische Volksrepublik Korea)
MP	Nördliche Marianen	NF	Norfolkinsel
NO	Norwegen	OM	Oman
AT	Österreich	TL	Osttimor
PK	Pakistan	PS	Palästinensische Autonomiegebiete
PW	Palau	PA	Panama
PG	Papua-Neuguinea	PY	Paraguay
PE	Peru	PH	Philippinen
PN	Pitcairninseln	PL	Polen
PT	Portugal	PR	Puerto Rico
RE	Réunion	RW	Ruanda
RO	Rumänien	RU	Russland
KN	Saint Kitts And Nevis	SB	Salomonen
ZM	Sambia	WS	Samoa
SM	San Marino	SH	Sankt Helena (Sankt Helena, Ascension und Tristan da Cunha)
PM	Sankt-Pierre und Miquelon	ST	São Tomé und Príncipe
SA	Saudi-Arabien	SE	Schweden
CH	Schweiz	SN	Senegal
RS	Serbien	SC	Seychellen
SL	Sierra Leone	ZW	Simbabwe
SG	Singapur	SK	Slowakei
SI	Slowenien	SO	Somalia
ES	Spanien	SJ	Spitzbergen und Jan Mayen
LK	Sri Lanka	LC	St. Lucia
VC	St. Vincent und die Grenadinen	ZA	Südafrika
SD	Sudan	GS	Südgeorgien und die Südlichen Sandwichinseln
KR	Südkorea (Republik Korea)	SS	Südsudan
SR	Suriname	SZ	Swasiland

Code	Land	Code	Land
SY	Syrien	TJ	Tadschikistan
TW	Taiwan	TZ	Tansania
TH	Thailand	TG	Togo
TK	Tokelau	TO	Tonga
TT	Trinidad und Tobago	TD	Tschad
CZ	Tschechien	TN	Tunesien
TR	Türkei	TM	Turkmenistan
TC	Turks- und Caicosinseln	TV	Tuvalu
SU	UdSSR (heute von Russland weiter-verwendet)	UG	Uganda
UA	Ukraine	HU	Ungarn
UM	United States Minor Outlying Islands	UY	Uruguay
UZ	Usbekistan	VU	Vanuatu
VA	Vatikanstadt	VE	Venezuela
AE	Vereinigte Arabische Emirate	US	Vereinigte Staaten
UK	Vereinigtes Königreich (auch GB)	GB	Vereinigtes Königreich (auch UK)
VN	Vietnam	WF	Wallis und Futuna
CX	Weihnachtsinsel	BY	Weißrussland
EH	Westsahara	CF	Zentralafrikanische Republik
CY	Zypern, Republik		

In meinen OOo-Dokumenten setze ich in großem Umfang Formatvorlagen ein. Für meine Codebeispiele habe ich eine besondere Absatzvorlage. Jede Absatzvorlage enthält die Möglichkeit, den im Absatz vorkommenden Zeichen bestimmte Standardattribute zu verleihen. In OOo ist das Gebietsschema ein Zeichenattribut, daher setze ich das Gebietsschema für diese Absatzvorlage auf „keine“, um zu verhindern, dass die Rechtschreibung meiner Codebeispiele überprüft wird.

Um OOo mitzuteilen, dass ein Wort Französisch ist und somit als Französisch geprüft werden soll, setzen Sie das Gebietsschema für die Zeichen auf Französisch. Der Code im Listing 323 durchläuft die Absätze eines Dokuments und setzt in jedem das Gebietsschema auf Französisch.

Listing 323. *Setzt ein einfaches Textdokument auf Französisch als Gebietsschema.*

```
Sub SetDocumentLocale
  Dim aLocale As New com.sun.star.lang.Locale
  aLocale.Language = "fr"    'Setzt das Gebietsschema auf die französische Sprache
  aLocale.Country = "FR"     'Setzt das Gebietsschema auf das Land Frankreich
  Dim oCursor  'Cursor zum Durchlauf durch das Dokument
  Dim oText    'Das Textobjekt des Dokuments

  oText = ThisComponent.Text          'Textdokumente haben ein Textobjekt
  oCursor = oText.createTextCursor() 'Erzeugt einen Textcursor

  REM Setzt den Cursor an den Dokumentanfang. Selektiert dabei keinen Text.
  oCursor.gotoStart(False)

  REM Verschiebt den Cursor an das Absatzende. Selektiert dabei den gesamten Absatz.
  REM gotoNextParagraph() gibt False zurück, wenn der Vorschub nicht möglich ist.
```

```
  Do While oCursor.gotoNextParagraph(True)
    oCursor.CharLocale = aLocale 'Dies kann bei manchen Absatztypen fehlschlagen
    oCursor.goRight(0, False)    'Hebt jede Textselektion auf.
  Loop
End Sub
```

Die Rechtschreibprüfung, die Silbentrennung und der Thesaurus brauchen zum Arbeiten alle ein Gebietsschema. Sie werden jedoch nicht funktionieren, wenn sie nicht ordentlich konfiguriert sind. Das geschieht über das Menü **Extras | Optionen | Spracheinstellungen | Linguistik**. Das Makro im Listing 324 greift auf die Objekte SpellChecker (Rechtschreibprüfung), Hyphenator (Silbentrennung) und Thesaurus zu, die alle eine Locale-Eigenschaft benötigen.

Listing 324. *Rechtschreibprüfung, Silbentrennung und Thesaurus.*

```
Sub SpellCheckExample
  Dim s()          'Enthält die zu prüfenden Wörter
  Dim vReturn      'Von SpellChecker, Hyphenator und Thesaurus zurückgegebener Wert
  Dim i As Integer 'Indexvariable
  Dim msg$         'Ausgabestring

  REM Ich erzeuge ein leeres Argument-Array.
  REM Ich könnte es auch mit Array() machen.
  Dim emptyArgs() As New com.sun.star.beans.PropertyValue

  Dim aLocale As New com.sun.star.lang.Locale
  aLocale.Language = "de"  'Sprache Deutsch
  aLocale.Country = "DE"   'Land Deutschland

  REM Wörter für die Rechtschreibprüfung, die Silbentrennung und den Thesaurus
  s = Array("hallo", "Anästhesiologie",_
      "PNEUMONOULTRAMICROSCOPICSILICOVOLCANOCONIOSIS",_
      "Pitonyak", "Rechtschreibfehler")

  REM ******Beispiel für die Rechtschreibprüfung!
  Dim vSpeller As Variant
  vSpeller = CreateUnoService("com.sun.star.linguistic2.SpellChecker")
  'Mit vReturn = vSpeller.spell(s, aLocale, emptyArgs()) erhalten Sie Optionen!
  For i = LBound(s()) To UBound(s())
    vReturn = vSpeller.isValid(s(i), aLocale, emptyArgs())
    msg = msg & vReturn & " für " & s(i) & Chr$(10)
  Next
  MsgBox msg, 0, "Rechtschreibprüfung"
  msg = ""

  REM ******Beispiel für die Silbentrennung!
  Dim vHyphen As Variant
  vHyphen = CreateUnoService("com.sun.star.linguistic2.Hyphenator")
  For i = LBound(s()) To UBound(s())
    'vReturn = vHyphen.hyphenate(s(i), aLocale, 0, emptyArgs())
    vReturn = vHyphen.createPossibleHyphens(s(i), aLocale, emptyArgs())
    If IsNull(vReturn) Then
      'Die Silbentrennung ist in der Konfiguration möglicherweise abgeschaltet
      msg = msg & " Null für " & s(i) & Chr$(10)
    Else
      msg = msg & vReturn.getPossibleHyphens() & " für " & s(i) & Chr$(10)
    End If
  Next
  MsgBox msg, 0, "Silbentrennung"
  msg = ""
```

```
  REM ******Beispiel für den Thesaurus!
  Dim vThesaurus As Variant
  Dim j As Integer, k As Integer
  vThesaurus = CreateUnoService("com.sun.star.linguistic2.Thesaurus")
  s = Array("hallo", "zug", "kalt")
  For i = LBound(s()) To UBound(s())
    vReturn = vThesaurus.queryMeanings(s(i), aLocale, emptyArgs())
    If UBound(vReturn) < 0 Then
      Print "Der Thesaurus hat nichts gefunden für " & s(i)
    Else
      msg = "Das Wort " & s(i) & " hat folgende Bedeutungen:" & Chr$(10)
      For j = LBound(vReturn) To UBound(vReturn)
        msg = msg & Chr$(10) & "Bedeutung = " & vReturn(j).getMeaning() & Chr$(10)
        msg = msg & Join(vReturn(j).querySynonyms(), " ") & Chr$(10)
      Next
      MsgBox msg, 0, "Andere Bedeutungen"
    End If
  Next
End Sub
```

Es ist möglich, das für OOo konfigurierte Standard-Gebietsschema zu ermitteln. Laurent Godard, ein aktiver OpenOffice.org-Freiwilliger, hat das Makro im Listing 325 zur Ermittlung des aktuell in OOo konfigurierten Gebietsschemas geschrieben.

Listing 325. *Aktuell eingestellte Sprache.*

```
Sub OOoLang()
  'Ermittelt die laufende OOo-Version
  'Autor : Laurent Godard
  'E-Mail : listes.godard@laposte.net
  '
  Dim aSettings, aConfigProvider
  Dim aParams2(0) As New com.sun.star.beans.PropertyValue
  Dim sProvider$, sAccess$
  sProvider = "com.sun.star.configuration.ConfigurationProvider"
  sAccess   = "com.sun.star.configuration.ConfigurationAccess"
  aConfigProvider = CreateUnoService(sProvider)
  aParams2(0).Name = "nodepath"
  aParams2(0).Value = "/org.openoffice.Setup/L10N"
  aSettings = aConfigProvider.createInstanceWithArguments(sAccess, aParams2())

  Dim OOLangue As String
  OOLangue = aSettings.getbyname("ooLocale")
  MsgBox "OOo ist mit dem Gebietsschema " & OOLangue & " konfiguriert", 0, _
         "OOo-Gebietsschema"
End Sub
```

Das Makro oben funktioniert unter LibreOffice, allerdings offenbar nicht unter AOO. Aber das folgende Makro sollte es sowohl unter AOO als auch unter LO tun.

Listing 326. *Nutzung der Bibliothek „Tools“ zur Ermittlung des aktuellen Gebietsschemas.*

```
GlobalScope.BasicLibraries.loadLibrary("Tools")
Print GetRegistryKeyContent("org.openoffice.Setup/L10N", False).getByName("ooLocale")
```

13.14. Auflistung der Drucker

Lange fehlte in OOo die Möglichkeit, Drucker aufzulisten. In den OOo-Versionen 1.x oder 2.x öffnete ich mit dem Dispatch-Befehl .uno:print den Druckdialog und griff danach direkt auf den Dialog mit der Liste der verfügbaren Drucker zu.

Dann wurde der Service PrinterServer eingeführt, wurde aber zuerst nicht richtig konstruiert, so dass die Methoden des Interface XPrinterServer nicht direkt zur Verfügung standen. Der Fehler ist aber mittlerweile behoben, getestet mit AOO 4.1.4 und LO 5.3.

Listing 327. *Ausgabe der verfügbaren Drucker.*

```
Sub PrintAllPrinterNames()
  Dim oPrintServer ' Der Service PrinterServer.
  Dim aNames       ' Liste der Druckernamen.

  oPrintServer = CreateUnoService("com.sun.star.awt.PrinterServer")
  aNames = oPrintServer.getPrinterNames()
  MsgBox Join(aNames, Chr$(10))
End Sub
```

Für den Fall, dass Sie noch mit einer OOo-Version arbeiten, die mit der direkten Methode Schwierigkeiten hat, stelle ich Ihnen eine funktionierende Lösung von Niklas Nebel vor. Sie ist trickreich, zeigt aber, wie man das Problem umgehen kann.

Listing 328. *Ausgabe der verfügbaren Drucker, Workaround.*

```
Sub PrintAllPrinterNamesWorkaround()
  Dim oPrintServer ' Der Service PrinterServer.
  Dim oCore        ' Zugriff auf Klassen und andere Objekte über ihre Namen.
  Dim oClass       ' Die Objektklasse XPrinterServer.
  Dim oMethod      ' Methode getPrinterNames von der Klasse XPrinterServer.
  Dim aNames       ' Liste der Druckernamen.

  ' Erzeugt das Objekt, das erst mit OOo 3.5 direkt verfügbar sein wird.
  oPrintServer = CreateUnoService("com.sun.star.awt.PrinterServer")
  oCore = CreateUnoService("com.sun.star.reflection.CoreReflection")

  ' Die Objektklasse für das Interface XPrinterServer.
  oClass = oCore.forName("com.sun.star.awt.XPrinterServer")

  ' Die Methode getPrinterNames für die Klasse XPrinterServer.
  oMethod = oClass.getMethod("getPrinterNames")

  ' Aufruf der Methode getPrinterNames für das Objekt PrinterServer.
  aNames = oMethod.invoke(oPrintServer, Array())
  MsgBox Join(aNames, Chr$(10))
End Sub
```

Tipp	Falls OOo Ihren Drucker nicht am Namen erkennt, setzen Sie den Druckernamen in Winkelklammern „<Druckername>“, jedenfalls war das früher für manche Drucker nötig.

Vor OOo 3.5 kann der PrinterServer nicht aus Basic heraus verwendet werden, weil XTypeProvider von dieser Klasse nicht als geerbt deklariert wird, obwohl es in der Klasse angelegt ist. Basic kann daher die von dem Objekt unterstützten Methoden nicht erkennen. Der Code im Listing 328 zeigt, wie man die Methode direkt aufruft.

13.15. Dokumente drucken

Die primäre Druckfunktionalität ist bei allen OOo-Dokumenttypen gleich. Das Interface com.sun.star.view.XPrintable definiert drei Methoden, s. Tabelle 123.

Tabelle 123. *Methoden im Interface com.sun.star.view.XPrintable.*

Objektmethode	Beschreibung
getPrinter()	Der Standarddrucker als Array von Eigenschaften (com.sun.star.view.PrinterDescriptor).
setPrinter(properties)	Weist dem Objekt einen neuen Drucker zu (com.sun.star.view.PrinterDescriptor).
print(properties)	Druckt das Dokument (com.sun.star.view.PrintOptions).

Die Objektmethode getPrinter() gibt ein Array der den Drucker beschreibenden Eigenschaften zurück, s. Bild 97. Das Makro im Listing 329 zeigt, wie man auf die einzelnen Eigenschaften zugreift und wie man sie interpretiert, s. Tabelle 124 zu den unterstützten Eigenschaften.

Listing 329. *Ausgabe der Druckereigenschaften.*

```
Sub DisplayPrinterProperties
  Dim oProps 'Array von com.sun.star.beans.PropertyValue
  Dim i%     'Indexvariable
  Dim s$     'Ausgabestring
  Dim v      'Wert einer Property
  Dim sName$ 'Name einer Property
  On Error Resume Next
  oProps = ThisComponent.getPrinter()
  For i = 0 To UBound(oProps)
    sName = oProps(i).Name
    v = oProps(i).Value
    If sName = "PaperOrientation" Then
      s = s & "Ausrichtung  =  "
      REM com.sun.star.view.PaperOrientation.LANDSCAPE wird auch unterstützt
      s = s & IIf(v = com.sun.star.view.PaperOrientation.PORTRAIT,_
              "Hochformat", "Querformat") & "  =  " & CStr(v)
    ElseIf sName = "PaperFormat" Then
      s = s & "Papierformat  =  "
      Select Case v
        Case com.sun.star.view.PaperFormat.A3
          s = s & "A3"
        Case com.sun.star.view.PaperFormat.A4
          s = s & "A4"
        Case com.sun.star.view.PaperFormat.A5
          s = s & "A5"
        Case com.sun.star.view.PaperFormat.B4
          s = s & "B4"
        Case com.sun.star.view.PaperFormat.B5
          s = s & "B5"
        Case com.sun.star.view.PaperFormat.LETTER
          s = s & "LETTER"
        Case com.sun.star.view.PaperFormat.LEGAL
          s = s & "LEGAL"
        Case com.sun.star.view.PaperFormat.TABLOID
          s = s & "TABLOID"
        Case com.sun.star.view.PaperFormat.USER
          s = s & "USER"
```

```
            Case Else
              s = s & "Unbekannt"
          End Select
          s = s & "  =  " & CStr(v)
        ElseIf sName = "PaperSize" Then
          s = s & "Abmessungen  =  "
          REM Typ ist com.sun.star.awt.Size
          REM Die Größe ist in TWIPS (1440 Twips pro Zoll, d.h. 56,69 Twips pro mm)
          s = s & Format(v.Width / 56.69, "0.#") & " x " & _
                  Format(v.Height / 56.69, "0.#") & " (mm)"
        Else
          s = s & sName & "  =  "
          s = s & CStr(v)
        End If
        s = s & Chr$(10)
      Next
      MsgBox s, 0, "Druckereigenschaften"
    End Sub
```

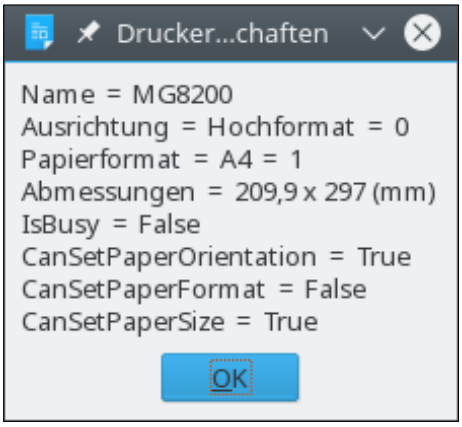


Bild 97. *Eigenschaften des Standarddruckers.*

Tabelle 124. *Eigenschaften im Service com.sun.star.view.PrinterDescriptor.*

Eigenschaft	Beschreibung
Name	Name der Druckerwarteschlange.
PaperOrientation	Papierausrichtung (com.sun.star.view.PaperOrientation).
PaperFormat	Vordefinierte Papierformate (com.sun.star.view.PaperFormat).
PaperSize	Papiergröße in Twips (com.sun.star.awt.Size).
IsBusy	Ist der Drucker beschäftigt?
CanSetPaperOrientation	Kann die Papierausrichtung eingestellt werden?
CanSetPaperFormat	Werden andere Papierformate unterstützt?
CanSetPaperSize	Kann die Papiergröße eingestellt werden?

Wenn Sie die Methode print() ohne Eigenschaften aufrufen, wird eine Einzelkopie des Dokuments auf dem aktuellen Drucker ausgegeben. Alle Dokumenttypen unterstützen die Eigenschaften in der Tabelle 125. Die Eigenschaft Pages unterstützt das Standardformat, das Sie vom Druckdialog kennen. Der Formatstring "1, 3, 4-7, 9-" druckt die Seiten 1, 3, 4 bis 7 und die Seite 9 bis zur letzten Seite.

Tabelle 125. *Eigenschaften im Service com.sun.star.view.PrintOptions.*

Eigenschaft	Beschreibung
CopyCount	Anzahl der zu druckenden Kopien.

Eigenschaft	Beschreibung
FileName	Ausgabe in eine Datei statt an den Drucker.
Collate	Sortiert die gedruckten Seiten (True oder False).
Pages	Zu druckende Seiten und Seitenbereiche.
Wait	Mit dem Wert True wird der Druckauftrag synchron ausgeführt, das heißt, die Kontrolle geht erst an das Makro zurück, wenn der Job erledigt ist.
DuplexMode	Der Duplexmodus wird mit der Konstantengruppe com.sun.star.view.DuplexMode gesetzt. Es gibt die Werte UNKNOWN (0), OFF (1), LONGEDGE (2) und SHORTEDGE (3).

Listing 330. *Druckt die Seiten 30 und 31 des aktuellen Dokuments.*

```
Sub PrintPages30_31()
  Dim oProps(1) As New com.sun.star.beans.PropertyValue
  oProps(0).Name = "Pages" : oProps(0).Value = "30-31"
  oProps(1).Name = "Wait"  : oProps(1).Value = True
  ThisComponent.print(oProps())
End Sub
```

Wenn ein Dokument gedruckt wird, geht die Kontrolle an den Aufrufer zurück, bevor der Druckvorgang beendet ist. Wenn Sie das Dokument vor dem Druckende schließen, wird OOo sehr wahrscheinlich abstürzen, weil OOo im Inneren das Dokument noch nutzt. Man kann einen Listener einsetzen, der auf das Ende des Druckjobs wartet, aber es gibt einen einfacheren Weg. Die Methode print() akzeptiert ein Array von Eigenschaften zur Druckkontrolle. Das Argument „Wait“ mit dem booleschen Wert True hindert die Methode print() daran, vor dem Druckende zurückzukehren. Ein Druck-Listener wird im Abschnitt 13.15.3. Beispiel für einen Druck-Listener in Calc vorgestellt.

Achtung Wenn man ein Dokument schließt, während OOo das Dokument druckt, kann OOo abstürzen. Mit der Eigenschaft „Wait“ verhindern sie das Problem.

Auf Unix-Rechnern werden Drucker für OOo über das Werkzeug „spadmin“ konfiguriert. Danach ist ein Drucker für OOo über seinen Namen erreichbar. Man kann darüber hinaus auch Drucker nutzen, die nicht für OOo konfiguriert wurden, dann aber muss der Druckername in Winkelklammern stehen: „<“ und „>“. Der folgende Code druckt über einen Drucker aus, der nicht der Standarddrucker ist.

```
Public oProps(0) As New com.sun.star.beans.PropertyValue
Public oOpts(1)  As New com.sun.star.beans.PropertyValue
Dim oDoc         'Zu druckendes Dokument.
Dim oPrinter     'Array von Eigenschaften, die den Drucker definieren.
Dim sUrl$        'URL des Dokuments, das geöffnet und gedruckt wird.
Dim sPrinter$    'Name des Druckers.

REM Der Druckername, wie er dem System bekannt ist.
sPrinter = "HP-Color-LaserJet-4650DN"

REM Das Dokument soll im Modus verborgen geöffnet werden, so dass es
REM auf dem Bildschirm nicht sichtbar ist.
oProps(0).Name = "Hidden"
oProps(0).Value = True

REM Öffnet das Dokument.
sUrl = "file:///c:/test_doc.odt"
oDoc = oDesk.loadComponentFromUrl(sUrl, "_blank", 63, oProps())
```

```
REM Kopie des aktuellen Druckerobjekts aus dem Dokument.
REM In Wahrheit ist es ein Array von Eigenschaftswerten.
REM Ändern Sie den Namen des Objekts auf den Namen des Druckers, der genutzt
REM werden soll. Achten Sie darauf, dass der Druckername der des Systems ist.
oPrinter = oDoc.getPrinter()
For i = LBound(oPrinter) To UBound(oPrinter)
  If oPrinter(i).Name = "Name" Then
    oPrinter(i).Value = sPrinter
  End If
Next i

REM Kopiert den Drucker zurück ins Dokument. Das Einzige, das
REM sich geändert hat, ist der Druckername.
oDoc.setPrinter(oPrinter)

REM Nun werden die Druckoptionen für den aktuellen Druckauftrag gesetzt.
REM Achten Sie darauf, dass der Druckername in Winkelklammern eingeschlossen ist.
REM Achten Sie auch darauf, dass die Methode print() so eingestellt ist, dass sie
REM erst nach dem Druckende die Kontrolle zurückgibt.
oOpts(0).Name = "Name"
oOpts(0).Value = "<" & sPrinter & ">"
oOpts(1).Name = "Wait"
oOpts(1).Value = True
oDoc.print(oOpts())
```

Tipp Früher wurde einmal experimentell festgelegt, dass man vor dem Drucken den Zieldrucker im Dokument angeben muss, wenn er nicht der Standarddrucker ist.

13.15.1. Textdokumente drucken

Die unterschiedlichen Dokumenttypen unterstützen zusätzliche Druckoptionen. Textdokumente unterstützen das Interface com.sun.star.text.XPagePrintable, s. Tabelle 126. Dieses Interface enthält eine alternative Methode mit erweiterten Kontrollmöglichkeiten der Druckausgabe. Der Hauptvorteil liegt darin, dass man mehrere Seiten des Dokuments auf einer Druckseite ausgeben kann.

Tabelle 126. *Methoden im Interface com.sun.star.text.XPagePrintable.*

Objektmethoden	Beschreibung
getPagePrintSettings()	Gibt ein Array von Eigenschaften zurück, s. Tabelle 127.
setPagePrintSettings(properties)	Ändert die Einstellungen, s. Tabelle 127.
printPages(properties)	Druckt mit den Eigenschaften aus der Tabelle 125.

Die Objektmethode printPages() akzeptiert dieselben Eigenschaften wie die Methode print() (s. Tabelle 125). Wie man die Eigenschaften des Seitendrucks ausliest und setzt, sehen Sie in der Tabelle 127. Das Makro im Listing 331 liest die aktuellen Seitendruck-Eigenschaften und gibt sie aus, s. Bild 98.

Tabelle 127. *Vom Interface com.sun.star.text.XPagePrintable genutzte Eigenschaften.*

Eigenschaft	Beschreibung
PageRows	Anzahl der Zeilen von Seitenfolgen auf jeder Druckseite.
PageColumns	Anzahl der Spalten von Seitenfolgen auf jeder Druckseite.
LeftMargin	Linker Rand.
RightMargin	Rechter Rand.
TopMargin	Oberer Rand.
BottomMargin	Unterer Rand.

Eigenschaft	Beschreibung
HoriMargin	Zwischenraum zwischen den Zeilen der Seitenfolgen.
VertMargin	Zwischenraum zwischen den Spalten der Seitenfolgen.
IsLandscape	Querformat, True oder False.

Listing 331. *Ausgabe der Seitendruckeigenschaften.*

```
Sub DisplayPagePrintProperties
  Dim oProps 'Array von com.sun.star.beans.PropertyValue
  Dim i%     'Indexvariable
  Dim s$     'Ausgabestring
  If HasUnoInterfaces(ThisComponent, "com.sun.star.text.XPagePrintable") Then
    oProps = ThisComponent.getPagePrintSettings()
    For i = 0 To UBound(oProps)
      s = s & oProps(i).Name & "  =  " & CStr(oProps(i).Value) & Chr$(10)
    Next
    MsgBox s, 0, "Seitendruckeigenschaften"
  Else
    Print "Hoppla, dieses Dokument unterstützt das Interface XPagePrintable nicht."
  End If
End Sub
```

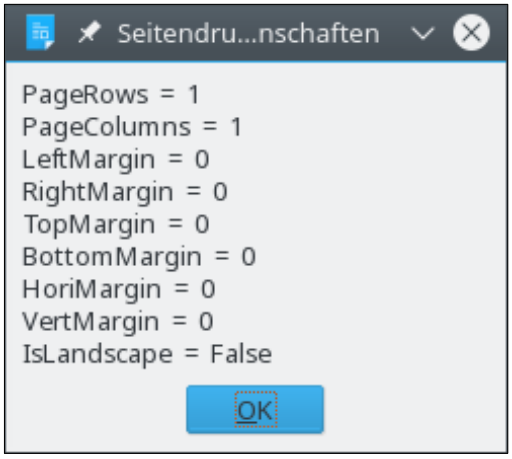


Bild 98. *Seitendruckeigenschaften eines Textdokuments.*

Das Makro im Listing 332 druckt ein Dokument mit zwei Seiten auf jeder Druckseite im Querformat. Meine letzten Versuche mit OOo 3.3 verliefen positiv, das Dokument wird normal gedruckt. Das ist eine Verbesserung gegenüber dem früheren Verhalten: da stürzte OOo ab.

Listing 332. *Druckt zwei Seiten pro Druckseite.*

```
Sub PrintTwoPerPage
  Dim oProps(0 To 1) As New com.sun.star.beans.PropertyValue
  oProps(0).Name = "PageColumns" : oProps(0).Value = 2        '2 Spalten
  oProps(1).Name = "IsLandscape" : oProps(1).Value = True     'Querformat
  If HasUnoInterfaces(ThisComponent, "com.sun.star.text.XPagePrintable") Then
    ThisComponent.setPagePrintSettings(oProps())  '
    ThisComponent.printPages(Array())            'Mit Standardeigenschaften
  Else
    Print "Hoppla, dieses Dokument unterstützt das Interface XPagePrintable nicht."
  End If
End Sub
```

13.15.2. Tabellendokumente drucken

Um besondere Druckfunktionen für ein Textdokument einzusetzen, muss man eine besondere Objektmethode aufrufen. Um besondere Druckfunktionen für ein Tabellendokument einzusetzen, muss man jedoch die Dokumenteigenschaften und Seitenvorlageneigenschaften modifizieren und dann die Standard-print()-Methode aufrufen. Zum Beispiel kommt es oft vor, dass eine Calc-Tabelle zu groß ist und daher nicht auf eine Papierseite passt. Um die Tabelle so zu skalieren, dass sie auf eine bestimmte Anzahl Seiten passt, setzt man die Eigenschaft ScaleToPages auf die passende Seitenmenge. Mit der Eigenschaft PageScale wird die Tabelle einfach prozentual skaliert, s. Listing 333.

Listing 333. *Druckt eine Tabelle in 25% der Größe. Das ist sehr klein.*

```
Sub PrintScaledSpreadsheet
  Dim s$     'Vorlagenname
  Dim oStyle 'Die aktuelle Seitenvorlage

  REM Die Seitenvorlage erhält man über die aktuell aktive Tabelle.
  REM In einem Calc-Dokument weiß der aktuelle Controller,
  REM welche Tabelle aktiv ist.
  s = ThisComponent.CurrentController.getActiveSheet().PageStyle
  oStyle = ThisComponent.StyleFamilies.getByName("PageStyles").getByName(s)
  REM oStyle.PageScale = 100   Der Standardwert ist 100 (für 100%).
  REM oStyle.ScaleToPages = 0  Der Standardwert ist 0, für nicht skalieren.
  oStyle.PageScale = 25         'Skaliert das Dokument auf 25% (sehr sehr sehr klein)
  ThisComponent.print(Array())  'Druckt das Dokument
End Sub
```

Der zweite Aspekt des Tabellendrucks betrifft den Druckbereich zusammen mit den Spalten und Zeilentiteln, s. Tabelle 128.

Tabelle 128. *Methoden im Interface com.sun.star.sheet.XPrintAreas.*

Objektmethode	Beschreibung
getPrintAreas()	Gibt ein Array vom Typ com.sun.star.table.CellRangeAddress zurück.
setPrintAreas(ranges)	Legt die Druckbereiche für die Tabelle über ein Array vom Typ CellRangeAddress fest. Falls nichts festgelegt ist, wird alles gedruckt.
getPrintTitleColumns()	Gibt True zurück, wenn Titelspalten auf allen folgenden Druckseiten wiederholt werden.
setPrintTitleColumns(boolean)	Wird auf True gesetzt, wenn Titelspalten auf allen folgenden Druckseiten wiederholt werden.
getTitleColumns()	Gibt ein Array vom Typ com.sun.star.table.CellRangeAddress zurück.
setTitleColumns(ranges)	Legt die Spalten fest, die als Titel dienen. Zeilen werden ignoriert, es zählen nur Spalten.
getPrintTitleRows()	Gibt True zurück, wenn Titelzeilen auf allen folgenden Druckseiten wiederholt werden.
setPrintTitleRows(boolean)	Wird auf True gesetzt, wenn Titelzeilen auf allen folgenden Druckseiten wiederholt werden.
getTitleRows()	Gibt ein Array vom Typ com.sun.star.table.CellRangeAddress zurück.
setTitleRows(ranges)	Legt die Zeilen fest, die als Titel dienen. Spalten werden ignoriert, es zählen nur Zeilen.

Die Methoden in der Tabelle 128 gelten für Tabellen in einem Calc-Dokument, im Gegensatz zum gesamten Calc-Dokument. Das Makro im Listing 334 legt zwei Druckbereiche fest und druckt das Dokument. Jeder Druckbereich wird auf einer neuen Seite gedruckt.

Listing 334. *Legt mehrere Druckbereiche in einem Tabellendokument fest und druckt sie aus.*

```
Sub PrintSpreadsheetAreas
  Dim oRanges(1) As New com.sun.star.table.CellRangeAddress  'Zellbereiche
  oRanges(0).Sheet = 0
  oRanges(0).StartColumn = 0 : oRanges(0).StartRow = 0 'A1
  oRanges(0).EndColumn = 3   : oRanges(0).EndRow = 4   'D5

  oRanges(1).Sheet = 0
  oRanges(1).StartColumn = 0 : oRanges(1).StartRow = 8 'A9
  oRanges(1).EndColumn = 3   : oRanges(1).EndRow = 10  'D11

  ThisComponent.CurrentController.getActiveSheet().setPrintAreas(oRanges())
  ThisComponent.print(Array())
End Sub
```

13.15.3. Beispiel für einen Druck-Listener in Calc

Die Tabelle1 in einem Calc-Dokument enthält eine Schaltfläche, die Tabelle2 druckt und Tabelle1 als aktive Tabelle belässt. Für den Druck muss Tabelle2 zur aktiven Tabelle gemacht werden. Der Code wurde folgendermaßen strukturiert:

1. Macht Tabelle2 aktiv.
2. Ruft die Druckmethode der Dokumentebene auf.
3. Macht Tabelle1 wieder aktiv.

Die Druckmethode gibt die Kontrolle unmittelbar wieder zurück, und das Dokument wird im Hintergrund gedruckt. Daher wird Tabelle1 zur aktiven Tabelle, bevor der Druck startet, mit der Folge, dass nicht Tabelle2, sondern Tabelle1 gedruckt wird. Die korrekte Lösung dieses Problems ist, einen Druck-Listener einzurichten, der erst nach dem Druckende die Tabelle1 reaktiviert.

Ein Listener wird in einer globalen Variablen gespeichert, so dass er noch lebt, wenn das Makro selbst beendet ist. Aber Achtung: Wenn Sie irgendein Makro bearbeiten, wird die globale Variable gelöscht, aber der Listener bleibt registriert, und es gibt keinen Weg, die Registrierung rückgängig zu machen, außer Sie schließen das Dokument. Also erzeugen Sie erst die Variablen, die den Listener referenzieren. Das Dokument ist auch gespeichert, aber ich könnte genauso gut ThisComponent referenzieren.

Listing 335. *Globale Variablen, die den Listener und das Dokument referenzieren.*

```
Global oPrintListener
Global oPrintJobListenerDoc
```

Die Aktivierung der aktuellen Tabelle geschieht über ein real existierendes Makro. Hier ist es:

Listing 336. *Hilfsroutine zur Aktivierung einer Tabelle eines Calc-Dokuments.*

```
REM ******************************
REM ** oDoc - Zu nutzendes Calc-Dokument. Ohne Fehlerprüfung.
REM ** sSheetName - Name der zu aktivierenden Tabelle. Die Existenz wird geprüft.
REM ******************************
Sub Set_active_sheet(oDoc, sSheetName)
  Dim oSheets
  oSheets = oDoc.Sheets
  If oSheets.hasByName(sSheetName) Then
    oDoc.CurrentController.setActiveSheet(oSheets.getByName(sSheetName))
  End If
End Sub
```

Wenn der Listener abgemeldet werden soll, wird an ihn ein Ereignis gesendet. Als Präfix für alle Routinen, die den Listener einsetzen, wird der Text „print_listener_“ verwendet.

Listing 337. *Die Methode disposing des Druck-Listeners.*

```
REM ******************************
REM ** Der Druckauftrag ist erledigt, also wird der Listener entfernt.
REM ******************************
Sub print_listener_disposing(oEvent)
  On Error Resume Next
  Dim emptyObj
  If Not IsNull(oPrintJobListenerDoc) And Not IsEmpty(oPrintJobListenerDoc) Then
    oPrintJobListenerDoc.removePrintJobListener(oPrintListener)
    oPrintJobListenerDoc = emptyObj
  End If
End Sub
```

Jedes Mal, wenn sich der Status des Druckauftrags ändert, wird der Listener benachrichtigt.

Listing 338. *Das Status-Geändert-Ereignis des Druck-Listeners.*

```
REM ******************************
REM ** Wird bei jeder Änderung des Druckauftragsstatus aufgerufen.
REM ** Bei Fehlern werden Fehlermeldungen ausgegeben.
REM ** Wenn das Ereignis einen Fehler oder das Ende des Druckauftrags mitteilt,
REM ** wird der Druck-Listener abgemeldet, und die Tabelle1 wird aktiviert.
REM ******************************
Sub print_listener_printJobEvent(oPrintJobEvent)
  Dim bCleanup As Boolean ' True, wenn der Listener abgemeldet werden soll.
  Dim sMessage$        ' Wenn der String nicht leer ist, wird eine Meldung ausgegeben.

  REM Alle unterstützten Ereignisstatusänderungen.
  Select Case oPrintJobEvent.State
    Case com.sun.star.view.PrintableState.JOB_STARTED
      ' Das zu druckende Dokument wird gerendert.
      bCleanup = False
    Case com.sun.star.view.PrintableState.JOB_COMPLETED
      ' Das Rendern ist beendet, das Spoolen beginnt.
      bCleanup = False
    Case com.sun.star.view.PrintableState.JOB_SPOOLED
      ' Druckpuffer ist erzeugt!
      'sMessage = "Der Druckauftrag wurde an den Drucker geschickt."
      bCleanup = True
    Case com.sun.star.view.PrintableState.JOB_ABORTED
      sMessage = "Der Druckvorgang wurde abgebrochen."
      bCleanup = True
    Case com.sun.star.view.PrintableState.JOB_FAILED
      sMessage = "Fehler beim Drucken."
      bCleanup = True
    Case com.sun.star.view.PrintableState.JOB_SPOOLING_FAILED
      sMessage = "Das Dokument wurde nicht gedruckt oder nicht zum Drucker geschickt."
      bCleanup = True
    Case Else
      sMessage = "Unbekannter und unerwarteter Druckstatus."
      bCleanup = True
  End Select

  REM Entfernt den Listener, wenn der Druckauftrag beendet ist, und ruft ein
  REM Hilfsmakro auf, um die Tabelle1 zu aktivieren.
  If bCleanup And Not IsNull(oPrintJobListenerDoc) _
```

```
                And Not IsEMpty(oPrintJobListenerDoc) Then
      On Error Resume Next
      Dim emptyObj
      oPrintJobListenerDoc.removePrintJobListener(oPrintListener)
      Call Set_active_sheet(oPrintJobListenerDoc, "Tabelle1")
      oPrintJobListenerDoc = emptyObj
    End If
    If sMessage <> "" Then
      MsgBox sMessage
    End If
End Sub
```

Wenn der Tabellenname existiert, wird die gewünschte Tabelle aktiviert, der Druck-Listener wird erzeugt und registriert, und dann wird der Dokumentdruck eingeleitet.

Listing 339. *Druck der angegebenen Tabelle.*

```
Sub PrintSheet(oDoc, sSheetToPrint)
  Dim sPrefix$  ' Präfix zur Identifizierung der Routinen des Druck-Listeners.
  Dim sService$ ' Der Servicename des Druck-Listeners.

  sPrefix = "print_listener_"
  sService = "com.sun.star.view.XPrintJobListener"
  If Not oDoc.Sheets.hasByName(sSheetToPrint) Then
    MsgBox "Das Dokument enthält keine Tabelle mit dem Namen " & sSheetToPrint
    Exit Sub
  End If

  Call Set_active_sheet(oDoc, sSheetToPrint)

  oPrintListener = CreateUnoListener(sPrefix, sService)
  oDoc.addPrintJobListener(oPrintListener)
  oPrintJobListenerDoc = oDoc
  oDoc.print(Array())
End Sub
```

13.15.4. Druckbeispiele von Vincent Van Houtte

Ich habe ein paar sehr interessante Beispiele von Vincent Van Houtte gefunden, die hier zu veröffentlichen er mir freundlicherweise gestattet hat. Die Highlights der aufgeführten Makros:

- PrintDoc – Druckt ein Dokument in einer bestimmten Papiergröße, mit bestimmten Papierschächten und mit (oder ohne) Hintergrundbild. In zahlreichen der beigefügten Methoden wird diese Routine aufgerufen. Beachten Sie, dass die Methode den Zieldrucker direkt benennt.
- PrintPage – Dasselbe wie printDoc, druckt aber eine einzelne Seite.
- CloseDocument – Schließt das bestimmte Dokument.
- ExportAsPdfAndSendEmail – Export als PDF-Dokument und Versendung einer E-Mail.

Tipp Listing 340 nutzt die sehr beliebte Routine FindCreateNumberFormatStyle, s. Listing 403.

Listing 340. *Druckbeispiele von Vincent Van Houtte (mit kleinen Modifikationen und deutschen Kommentaren vom Übersetzer).*

```
Sub PrintWithCopyStamp()
' ------------------------------
' Dieses Makro fügt einen 'KOPIE'-Stempel ein und druckt das Dokument
```

```
' ohne Hintergrund(-bild) zum Papierschacht tray1.
'
' Autor Vincent Van Houtte (2010)
' ------------------------------
   REM Fügt den Stempel mit dem Sendezeitpunkt ein.
   Dim sActionText As String
   sActionText = "KOPIE"
   InsertDTstamp(sActionText)

   REM Druckt die Seite
   PrintDocWithoutBgToTray1()

   REM Entfernt den Rahmen mit dem Kopie-Stempel.
   RemoveDTstamp()
End Sub

Sub InsertDTstamp(sActionText)
' ------------------------------
' Dieses Makro fügt einen 'Datum/Uhrzeit'-Stempel ein mit sActionText
'
' Autor Vincent Van Houtte (2011)
' ------------------------------
   Dim oCursor, oText, oDoc
   oDoc = ThisComponent
   oText = oDoc.getText()
   oCursor = oText.createTextCursor()
   oCursor.gotoStart(False)

   REM Erzeugt die Datum- und Zeitobjekte.
   Dim oDate, oTime
   oDate = oDoc.createInstance("com.sun.star.text.TextField.DateTime")
   oDate.IsFixed = True
   oDate.IsDate = True
   oDate.NumberFormat = FindCreateNumberFormatStyle("T MMMM JJJJ", oDoc)

   oTime = oDoc.createInstance("com.sun.star.text.TextField.DateTime")
   oTime.IsFixed = True
   oTime.IsDate = False
   oTime.NumberFormat = FindCreateNumberFormatStyle("HH:MM", oDoc)

   REM Erzeugt den Rahmen.
   Dim oFrameDT As Object
   oFrameDT = oDoc.createInstance("com.sun.star.text.TextFrame")
   With oFrameDT
     .setName("FrameDT")
     .AnchorType = com.sun.star.text.TextContentAnchorType.AT_PAGE
     .HoriOrient = com.sun.star.text.HoriOrientation.NONE
     .VertOrient = com.sun.star.text.VertOrientation.NONE
     .HoriOrientPosition = -4900
     .VertOrientPosition = -1600
     .Width = 4000
     .Height = 1500
     .BorderDistance = 100
   End With

   REM Fügt den Rahmen in das Textdokument ein.
   oText.insertTextContent(oCursor, oFrameDT, True)
```

```
    REM Schreibt den Text in den Rahmen.
    Dim oCursor2 As Object
    oCursor2 = oFrameDT.createTextCursor()
    With oCursor2
      .CharHeight = 16
      .CharWeight = com.sun.star.awt.FontWeight.BOLD
      .ParaAdjust = com.sun.star.style.ParagraphAdjust.CENTER
    End With

    oFrameDT.insertString(oCursor2, sActionText, False)

    With oCursor2
      .CharHeight = 9
      .CharWeight = com.sun.star.awt.FontWeight.NORMAL
      .ParaAdjust = com.sun.star.style.ParagraphAdjust.CENTER
    End With

    oFrameDT.insertControlCharacter(oCursor2, _
                        com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
    oFrameDT.insertTextContent(oCursor2, oDate, False)

    oFrameDT.insertControlCharacter(oCursor2, _
                        com.sun.star.text.ControlCharacter.LINE_BREAK, False)
    oFrameDT.insertTextContent(oCursor2, oTime, False)
End Sub

Sub RemoveDTstamp()
' ------------------------------
' Dieses Makro entfernt den mit dem vorigen Makro erzeugten 'Datum/Uhrzeit'-Stempel.
'
' Autor Vincent Van Houtte (2011)
' ------------------------------
    Dim oDoc, oTextFrames, oFrameDT
    oDoc = ThisComponent

    REM Sucht den Stempel-Rahmen und entfernt ihn.
    oTextFrames = oDoc.getTextFrames
    If oTextFrames.hasByName("FrameDT") Then
       oFrameDT = oTextFrames.getByName("FrameDT")
       oFrameDT.dispose()
    End If
End Sub

Sub PrintDocWithBgToTray1
' ------------------------------
' Dieses Makro druckt das Dokument mit dem Hintergrund(-Bild) zum ersten
' Papierschacht. Wird nur gebraucht, wenn am Geschäftspapier gespart werden soll,
' wenn es aufgebraucht ist oder wenn der zweite Papierschacht blockiert ist.
'
' Autor Vincent Van Houtte (2010)
' ------------------------------
    Dim sTray1 As String
    Dim sTray2 As String
    Dim bBg1 As Boolean
    Dim bBg2 As Boolean
    Dim wait As Boolean
    sTray1 = "Tray1"
```

```
    sTray2 = "Tray1"
    bBg1 = True
    bBg2 = False
    wait = True
    printDoc(sTray1, sTray2, bBg1, bBg2, wait)
End Sub

Sub PrintDocWithoutBgToTray1
' ------------------------------
' Dieses Makro druckt das Dokument ohne Hintergrund(-Bild) zum
' ersten Papierschacht. Nützlich für Kopien der versendeten Briefe.
'
' Autor Vincent Van Houtte (2010)
' ------------------------------
    Dim sTray1 As String
    Dim sTray2 As String
    Dim bBg1 As Boolean
    Dim bBg2 As Boolean
    Dim wait As Boolean
    sTray1 = "Tray1"
    sTray2 = "Tray1"
    bBg1 = False
    bBg2 = False
    wait = True
    printDoc(sTray1, sTray2, bBg1, bBg2, wait)
End Sub

Sub PrintDocWithoutBgToTray2_old
' ------------------------------
' Dieses Makro druckt das Dokument ohne Hintergrund(-Bild) zum
' zweiten Papierschacht. Nützlich, wenn vorgedrucktes Geschäftspapier vorhanden ist:
' man kann ein Bild als Hintergrund haben, das zwar nicht gedruckt werden,
' aber bei der Konvertierung nach PDF erscheinen soll.
'
' Autor Vincent Van Houtte (2010)
' ------------------------------
    Dim sTray1 As String
    Dim sTray2 As String
    Dim bBg1 As Boolean
    Dim bBg2 As Boolean
    Dim wait As Boolean
    sTray1 = "Tray2"
    sTray2 = "Tray1"
    bBg1 = False
    bBg2 = False
    wait = True
    printDoc(sTray1, sTray2, bBg1, bBg2, wait)
End Sub

Sub PrintDocWithoutBgToTray2
' ------------------------------
' Dieses Makro druckt die erste Seite (ohne Hintergrundbild) zum
' zweiten Papierschacht und alle anderen Seiten (ohne Hintergrundbild) zum
' ersten Papierschacht.
' Nützlich, wenn vorgedrucktes Geschätspapier vorhanden ist, aber aus Kostengründen
' nur die erste Seite auf dem teuren Geschäftspapier gedruckt werden soll:
' man kann ein Bild als Hintergrund haben, das zwar nicht gedruckt werden,
' aber bei der Konvertierung nach PDF erscheinen soll.
'
' Autor Vincent Van Houtte (2011)
```

```
' ------------------------------

   Dim oDoc As Object
   Dim iPageCount As Integer, n As Integer
   Dim sPage As String
   oDoc = ThisComponent

   REM Zählt die Anzahl der Seiten.
   iPageCount = oDoc.getCurrentController().getPropertyValue("PageCount")

   REM Schleife über jede Seite
   n = 1
   Do Until n > iPageCount

   REM Druckt jede Seite zum richtigen Papierschacht
      If n = 1 Then
         Print_stat(n)
      Else
         Print_plain(n)
      End If
      n = n + 1
   Loop
End Sub

Sub Print_stat(sPageNr As String)
' ------------------------------
' Dieses Makro druckt die erste Seite ohne Hintergrund(-Bild) zum
' zweiten Papierschacht. Nützlich, wenn vorgedrucktes Geschäftspapier vorhanden ist:
' man kann ein Bild als Hintergrund haben, das zwar nicht gedruckt werden,
' aber bei der Konvertierung nach PDF erscheinen soll.
'
' Autor Vincent Van Houtte (2011)
' ------------------------------
   Dim sTray1 As String
   Dim sTray2 As String
   Dim bBg1 As Boolean
   Dim bBg2 As Boolean
   Dim wait As Boolean
   sTray1 = "Tray2"
   sTray2 = "Tray1"
   bBg1 = False
   bBg2 = False
   wait = True
   printPage(sTray1, sTray2, bBg1, bBg2, wait, sPageNr)
End Sub

Sub Print_plain(sPageNr As String)
' ------------------------------
' Dieses Makro druckt die nächste Seite ohne Hintergrund(-Bild) zum
' ersten Papierschacht. Nützlich, wenn man vorgedrucktes Geschäftspapier sparen will:
' man kann ein Bild als Hintergrund haben, das zwar nicht gedruckt werden,
' aber bei der Konvertierung nach PDF erscheinen soll.
'
' Autor Vincent Van Houtte (2011)
' ------------------------------
   Dim sTray1 As String
   Dim sTray2 As String
```

```
    Dim bBg1 As Boolean
    Dim bBg2 As Boolean
    Dim wait As Boolean
    sTray1 = "Tray1"
    sTray2 = "Tray1"
    bBg1 = False
    bBg2 = False
    wait = True
    printPage(sTray1, sTray2, bBg1, bBg2, wait, sPageNr)
End Sub

Sub printDoc(sTray1, sTray2, bBg1, bBg2, wait)
' ------------------------------
' Dieses Makro druckt das Dokument mit den übergebenen Argumenten.
'
' Autor Vincent Van Houtte (2010)
' ------------------------------
    Dim oDoc As Object
    oDoc = ThisComponent

    REM Setzt die Hintergrunddruck-Option der Dokumenteinstellungen auf False oder True
    Dim oSettings As Object
    oSettings = oDoc.createInstance("com.sun.star.text.DocumentSettings")
    oSettings.PrintPageBackground = bBg1

    REM Wahl eines bestimmten Druckers
    Dim mPrinterOpts(2) As New com.sun.star.beans.PropertyValue
    mPrinterOpts(0).Name = "Name"
    mPrinterOpts(0).Value = "MFC8880DN"
    mPrinterOpts(1).Name = "PaperFormat"
    mPrinterOpts(1).Value = com.sun.star.view.PaperFormat.A4
    mPrinterOpts(2).Name = "PaperOrientation"
    mPrinterOpts(2).Value = com.sun.star.view.PaperOrientation.PORTRAIT
    oDoc.Printer = mPrinterOpts()

    REM Setzt den Papierschacht in der Seitenvorlage
    Dim oStyle As Object
    Dim sPageStyle As String
    sPageStyle = oDoc.CurrentController.getViewCursor().PageStyleName
    oStyle = oDoc.StyleFamilies.getByName("PageStyles").getByName(sPageStyle)
    oStyle.PrinterPaperTray = sTray1

    REM Setzt die Druckoptionen
    Dim mPrintOpts(2) As New com.sun.star.beans.PropertyValue
    mPrintOpts(0).Name = "CopyCount"
    mPrintOpts(0).Value = 1
    mPrintOpts(1).Name = "Collate"
    mPrintOpts(1).Value = True
    mPrintOpts(2).Name = "Wait"
    mPrintOpts(2).Value = True

    REM Druckt
    oDoc.print(mPrintOpts())

    REM RESET OPTIONS
    REM Setzt die Hintergrunddruck-Option der Dokumenteinstellungen zurück.
    oSettings.PrintPageBackground = bBg2
    REM Setzt den Papierschacht in der Seitenvorlage zurück
```

```
    oStyle.PrinterPaperTray = sTray2
    REM Druckt 0 Seiten, um den Reset wirksam werden zu lassen
    Dim mPrintOpts2(0) As New com.sun.star.beans.PropertyValue
    mPrintOpts2(0).Name = "CopyCount"
    mPrintOpts2(0).Value = 0
    oDoc.print(mPrintOpts2())
End Sub

Sub printPage(sTray1, sTray2, bBg1, bBg2, wait, sPageNr)
' ------------------------------
' Dieses Makro druckt das Dokument mit den übergebenen Argumenten.
'
' Autor Vincent Van Houtte (2010)
' ------------------------------
    Dim oDoc As Object
    oDoc = ThisComponent

    REM Setzt die Hintergrunddruck-Option der Dokumenteinstellungen auf False oder True
    Dim oSettings As Object
    oSettings = oDoc.createInstance("com.sun.star.text.DocumentSettings")
    oSettings.PrintPageBackground = bBg1
    REM Wahl eines bestimmten Druckers
    Dim mPrinterOpts(3) As New com.sun.star.beans.PropertyValue
    mPrinterOpts(0).Name = "Name"
    mPrinterOpts(0).Value = "MFC8880DN"
    mPrinterOpts(1).Name = "PaperFormat"
    mPrinterOpts(1).Value = com.sun.star.view.PaperFormat.A4
    mPrinterOpts(2).Name = "PaperOrientation"
    mPrinterOpts(2).Value = com.sun.star.view.PaperOrientation.PORTRAIT
    oDoc.Printer = mPrinterOpts()

    REM Setzt den Papierschacht in der Seitenvorlage
    Dim oStyle As Object
    Dim sPageStyle As String
    sPageStyle = oDoc.CurrentController.getViewCursor().PageStyleName
    oStyle = oDoc.StyleFamilies.getByName("PageStyles").getByName(sPageStyle)
    oStyle.PrinterPaperTray = sTray1

    REM Setzt die Druckoptionen
    Dim mPrintOpts(3) As New com.sun.star.beans.PropertyValue
    mPrintOpts(0).Name = "CopyCount"
    mPrintOpts(0).Value = 1
    mPrintOpts(1).Name = "Collate"
    mPrintOpts(1).Value = True
    mPrintOpts(2).Name = "Pages"
    mPrintOpts(2).Value = sPageNr
    mPrintOpts(3).Name = "Wait"
    mPrintOpts(3).Value = True

    REM Druckt
    oDoc.print(mPrintOpts())

    REM RESET OPTIONS
    REM Setzt die Hintergrunddruck-Option der Dokumenteinstellungen zurück.
    oSettings.PrintPageBackground = bBg2
    REM Setzt den Papierschacht in der Seitenvorlage zurück
```

```
   oStyle.PrinterPaperTray = sTray2
   REM Druckt 0 Seiten, um den Reset wirksam werden zu lassen
   Dim mPrintOpts2(0) As New com.sun.star.beans.PropertyValue
   mPrintOpts2(0).Name = "CopyCount"
   mPrintOpts2(0).Value = 0
   oDoc.print(mPrintOpts2())
End Sub

Sub CloseDocument(oDoc As Object)
' ------------------------------
' Dieses Makro schließt das aktuelle Dokument
'
' Autor Andrew Pitonyak (2010)
' Adaptiert von Vincent Van Houtte (2011)
' ------------------------------

   REM Prüft, ob das Dokument existiert
   If IsNull(oDoc) Then
      Exit Sub
   End If

   REM Speichert das Dokument, wenn es geändert wurde
   If (oDoc.isModified) Then
     If (oDoc.hasLocation And (Not oDoc.isReadOnly)) Then
       oDoc.store()
     Else
       oDoc.setModified(False)
     End If
   End If

   REM Schließt das Dokument
   oDoc.close(True)
End Sub

Sub ExportAsPdfAndSendEmail
' ------------------------------
' Dieses Makro konvertiert das aktive Dokument nach PDF und fügt die PDF-Datei
' einer neuen E-Mail hinzu. Empfängeradresse und Betreff werden automagisch
' aus Textfeldern (Eingabefeldern) innerhalb des Dokuments erzeugt.
'
' Dieses Makro setzt voraus, dass die Standard-E-Mail-Anwendung
' in Extras -> Optionen -> Internet -> E-Mail eingetragen ist.
'
' Dieses Makro nutzt SimpleCommandMail, das unter Windows nicht laufen wird.
' Versuchen Sie es stattdessen mit SimpleSystemMail.
'
' Autor Vincent Van Houtte (2010)
' ------------------------------
   Dim oDoc, MailClient, MailAgent, MailMessage
   Dim sDocURL As String, sPDFURL As String, sTo As String, sSubject As String

   REM Holt die Speicheradresse des Dokuments
   oDoc = ThisComponent
   If (Not oDoc.hasLocation()) Then
      oDoc.store()
   End If

   REM Fügt den Stempel des Sendezeitpunkts ein
   Dim sActionText As String
```

```
    sActionText = "ABGESENDET"
    InsertDTstamp(sActionText)

    REM Druckt die Seite
    PrintDocWithoutBgToTray1()

    REM Ersetzt .odt durch .pdf
    sDocURL = oDoc.getURL()
    sPDFURL = Left$(sDocURL,Len(sDocURL) - 4) + ".pdf"

    REM Speichert als PDF
    Dim args(0) As New com.sun.star.beans.PropertyValue
    args(0).Name = "FilterName"
    args(0).Value = "writer_pdf_Export"
    oDoc.storeToURL(sPDFURL, args())

    REM Entfernt den Datum/Uhrzeit-Stempel
    RemoveDTstamp()

    REM Holt für die Betreffzeile die Werte aus den Textfeldern innerhalb des Dokuments.
    Dim enuTF, aTextField
    Dim sDosName As String, sDosNum As String, sDosUref As String
    enuTF = oDoc.TextFields.createEnumeration
       Do While enuTF.hasMoreElements
         aTextField = enuTF.nextElement
         If aTextField.supportsService("com.sun.star.text.TextField.Input") Then
           Select Case aTextField.getPropertyValue("Hint")
             Case "DOS_NAAM":
               sDosName = aTextField.getPropertyValue("Content")
             Case "DOS_NUM":
               sDosNum = aTextField.getPropertyValue("Content")
             Case "REF_O":
               sDosNum = aTextField.getPropertyValue("Content")
             Case "UREF":
               sDosUref = aTextField.getPropertyValue("Content")
             Case "EMAIL_ADDR":
               sTo = aTextField.getPropertyValue("Content")
           End Select
         End If
       Loop
    sSubject = sDosName + " - " + sDosUref + " - " + sDosNum

    REM Versendet die PDF-Datei als E-Mail-Anhang
    MailAgent = CreateUnoService("com.sun.star.system.SimpleCommandMail")
    MailClient = MailAgent.querySimpleMailClient()
       MailMessage = MailClient.createSimpleMailMessage()
       MailMessage.setRecipient(sTo)
       MailMessage.setSubject(sSubject)
       MailMessage.setAttachement(Array(sPDFURL))
    MailClient.sendSimpleMailMessage(MailMessage, 0)

    REM Speichert und schließt das Dokument
    CloseDocument(oDoc)
End Sub
```

13.16. Services erzeugen

Ein Objekt, das das Interface XMultiServiceFactory unterstützt, darf Services erzeugen. Ein Objekt wird normalerweise von dem Objekt erzeugt, zu dem es auch gehören wird. Zum Beispiel wird eine Texttabelle von dem Textdokument erzeugt, das diese Texttabelle enthalten wird. Gleichermaßen wird das Objekt DocumentSettings vom betreffenden Dokument erzeugt. Die Funktion CreateUnoService erzeugt Services im Bereich der OOo-Anwendungsebene.

Das Objekt oSettings im Listing 341 unterstützt den Service com.sun.star.text.DocumentSettings. Das Objekt oSettings bezieht sich auf ThisComponent, denn es spiegelt die Einstellungen für ThisComponent – dem aktuellen Dokument – wider, nicht aber die für irgendein anderes Dokument.

Listing 341. *Erzeugt ein Objekt, das den Service DocumentSettings unterstützt.*

```
oSettings = ThisComponent.createInstance("com.sun.star.text.DocumentSettings")
```

Die Methode createInstance() gibt ein Objekt zurück, das den angeforderten Service nach Möglichkeit unterstützt. Wenn das nicht möglich ist, wird NULL zurückgegeben. Das zurückgegebene Objekt kann auch gleichzeitig andere Services unterstützen, s. Listing 342 und Bild 99.

Listing 342. *Inspizierung des Objekts für Einstellungen eines Textdokuments.*

```
Sub WriteDocumentSettings
  Dim oSettings 'Das zu erzeugende Settings-Objekt
  Dim s$        'Hilfsstring
  Dim i%        'Indexvariable
  Dim v         'Wird ein Array von Servicenamen enthalten
  REM Erzeugt ein Objekt, das den Service DocumentSettings unterstützt
  oSettings = ThisComponent.createInstance("com.sun.star.text.DocumentSettings")

  v = oSettings.getSupportedServiceNames()
  s = "**** Die einzelnen unterstützten Services ****" & Chr$(10) & Join(v, Chr$(10))
  s = s & Chr$(10) & Chr$(10) & "**** Getestete Services ****" & Chr$(10)

  REM Prüft, ob dieses erzeugte Objekt andere Services unterstützt.
  v = Array("com.sun.star.comp.Writer.DocumentSettings",_
            "com.sun.star.text.PrintSettings")

  For i = 0 To UBound(v)
    If oSettings.supportsService(v(i)) Then
      s = s & "Unterstützt den Service " & v(i) & Chr$(10)
    End If
  Next
  MsgBox s, 0, "Einige Services für " & oSettings.getImplementationName()

  REM Welchen Status hat die Eigenschaft PrintControls?
  Print oSettings.PrintControls

  REM Ich könnte den Status auf True oder False setzen
  'oSettings.PrintControls = True
End Sub
```

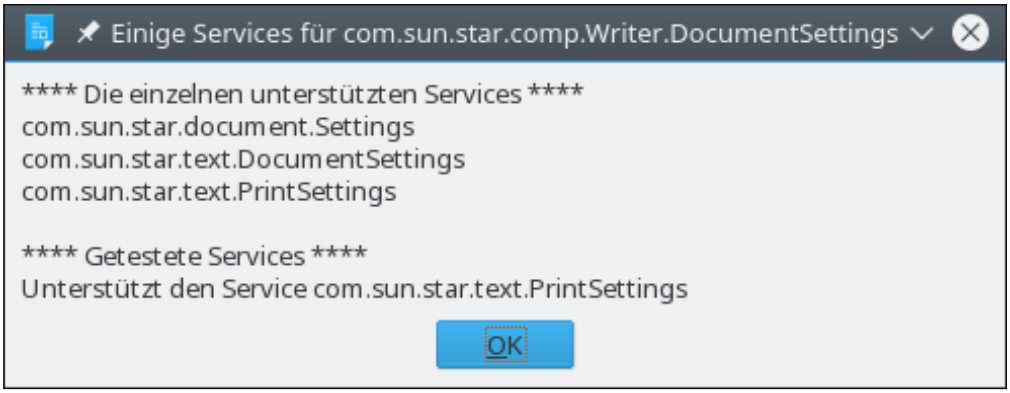


Bild 99. *Einige Textdokument-Einstellungen (LO 5.3).*

Eine genaue Untersuchung von Listing 342 und Bild 99 zeigt, dass das Objekt mehrere Services unterstützt. Die Ausgabe von AOO zeigt zusätzlich den unterstützten Service com.sun.star.text.PrintPreviewSettings, der allerdings als veraltet gekennzeichnet ist und von LO schon gänzlich deaktiviert ist. AOO weist auch den Service com.sun.star.comp.Writer.DocumentSettings als getestet aus.

Die Methode getSupportedServiceNames() listet die vom Objekt unterstützten Services auf.

Die Methode getImplementationName() gibt den Namen des Service zurück, der den erzeugten Service (com.sun.star.comp.Writer.DocumentSettings) eindeutig identifiziert. Dieser eindeutige Servicename ist nicht in der Liste der unterstützten Servicenamen.

Obwohl es im Listing 342 nicht gezeigt wird, können die beiden „Getesteten Services“ nicht über die Objektmethode createInstance() erzeugt werden.

13.17. Dokumenteinstellungen

In OOo gibt es zahlreiche Dokumentoptionen, erreichbar über den Dialog **Extras | Optionen**. Zum Beispiel kann man im Dialog „LibreOffice Writer | Drucken“ wählen, ob Grafiken gedruckt werden oder nicht. Normalerweise werden Eigenschaften in einem Dokument über die „get“- und „set“-Methoden geändert, oder durch den direkten Zugriff auf die Objekteigenschaften. Der Zugriff auf erweiterte Dokumenteinstellungen erfordert jedoch eine komplett andere Methode. Als erstes muss das Dokument ein Objekt erzeugen, das die Dokumenteinstellungen unterstützt. Jeder Dokumenttyp mit Ausnahme der Formeln ist in der Lage, mit der Methode createInstance() einen Service zu erzeugen. Tabelle 129 enthält eine Liste der Eigenschaften, die für alle Typen der Dokumenteinstellungen gelten. Es gibt noch mehr.

Tabelle 129. *Eigenschaften für jede Dokumentart im Service com.sun.star.document.Settings.*

Eigenschaft	Beschreibung
ForbiddenCharacters	Erlaubt den Zugriff auf andernfalls nicht erlaubte Zeichen.
LinkUpdateMode	Der Update-Modus für Hyperlinks beim Öffnen von Textdokumenten.
PrinterName	Der vom Dokument verwendete Drucker.
PrinterSetup	Plattform- und treiberabhängige Druckersetup-Daten.
IsKernAsianPunctuation	Wird Kerning auf asiatische Satzzeichen angewendet?
CharacterCompressionType	Für asiatische Zeichen verwendete Kompression (Zeichenabstand).
ApplyUserData	Sollen die im Dokument gespeicherten Benutzerdaten wieder geladen werden?
SaveVersionOnClose	Wird beim Schließen eines geänderten Dokuments eine neue Version erstellt?
UpdateFromTemplate	Soll ein Dokument aktualisiert werden, wenn sich die zugrunde liegende Dokumentvorlage ändert?
FieldAutoUpdate	Werden Felder in Textdokumenten automatisch aktualisiert?

Eigenschaft	Beschreibung
CurrentDatabaseDataSource	Name der Datenquelle, aus der die aktuellen Daten stammen.
CurrentDatabaseCommand	Name des angezeigten Objekts (oder der benutzte SQL-Befehl).
CurrentDatabaseCommandType	Bestimmt, wie die Eigenschaft DataTableName zu interpretieren ist.
DefaultTabStop	Standard-Tabulatorbreite.
IsPrintBooklet	Wird das Dokument als Broschüre gedruckt?
IsPrintBookletFront	Falls True, werden nur die Vorderseiten einer Broschüre gedruckt.
IsPrintBookletBack	Falls True, werden nur die Rückseiten einer Broschüre gedruckt.
PrintQuality	Die zu nutzende Druckqualität.
ColorTableURL	URL der Farbtabelle (SOC-Datei) mit einer Palette für Dialoge, die Farben nutzen.
PrinterIndependentLayout	Falls True, werden für das Layout keine Druckermetriken verwendet.

Jeweils eigene Services zu Dokumenteinstellungen gibt es für Writer-, Calc-, Draw- und Impress-Dokumente, s. Tabelle 130. Obwohl all diese Services den Service Settings (s. Tabelle 129) einbinden, sind alle Eigenschaften außer PrinterName und PrinterSetup optional.

Tabelle 130. *Die einzelnen Typen des Service DocumentSettings.*

Service DocumentSettings	Typ des Dokuments
com.sun.star.text.DocumentSettings	Writer
com.sun.star.sheet.DocumentSettings	Calc
com.sun.star.drawing.DocumentSettings	Draw
com.sun.star.presentation.DocumentSettings	Impress

Wie in Listing 342 und Bild 99 zu sehen ist, unterstützen die Dokumenteinstellungen des Writers den Service PrintSettings, s. Tabelle 131. Die Services für Dokumenteinstellungen in Draw und Impress enthalten jeweils eigene Einstellungen, die nur für die entsprechenden Dokumenttypen gelten.

Tabelle 131. *Eigenschaften im Service com.sun.star.text.PrintSettings.*

Eigenschaft	Beschreibung
PrintGraphics	Falls True, werden Grafikobjekte gedruckt.
PrintTables	Falls True, werden Texttabellen gedruckt.
PrintDrawings	Falls True, werden Zeichnungselemente gedruckt.
PrintLeftPages	Falls True, werden linke Seiten gedruckt.
PrintRightPages	Falls True, werden rechte Seiten gedruckt.
PrintControls	Falls True, werden Kontrollfelder gedruckt.
PrintReversed	Falls True, werden die Seiten in umgekehrter Reihenfolge gedruckt, beginnend mit der letzten Seite.
PrintPaperFromSetup	Falls True, wird der Papierschacht der Systemdruckeinstellung genutzt. Falls False, wird der in der Seitenvorlage festgelegte Papierschacht genutzt.
PrintFaxName	Name des Fax.
PrintAnnotationMode	Modus, wie Kommentare gedruckt werden. Verwendet die Enumerationswerte von com.sun.star.text.NotePrintMode: NOT, ONLY, DOC_END oder PAGE_END.
PrintProspect	Falls True, wird als Prospekt (Broschüre) gedruckt.
PrintPageBackground	Falls True, wird die Hintergrundfarbe und/oder das Hintergrundbild gedruckt.
PrintBlackFonts	Falls True, werden die Zeichen immer in Schwarz gedruckt.

13.18. Der coolste Trick, den ich kenne

Er funktioniert nur, wenn die OOo-Dialoge verwendet werden (**Extras | Optionen | LibreOffice | Allgemein**). Während Sie ein Dokument editieren:

1. Starten Sie den Öffnen-Dialog über **Datei | Öffnen**.
2. Geben Sie als Dateiname „vnd.sun.star.tdoc:/“ ein und klicken auf Öffnen.
3. Es werden aktuell geöffnete Dokumente zur Selektion angeboten. Wählen Sie eins aus und klicken auf Öffnen.

Sie sehen nun die internen Strukturen des Dokuments. Sie können sogar ein im Dokument eingebettetes Bild öffnen. Probieren Sie es am besten einmal mit einem Dokument mit Bildern aus.

13.19. Einen URL in anderen Sprachen konvertieren

Basic stellt die Funktionen ConvertToURL und ConvertFromURL zur Verfügung, andere Sprachen tun das nicht. Manche URI-Übersichten (URI = Uniform Resource Identifier = einheitlicher Bezeichner für Ressourcen) lassen wichtige Aspekte darüber unerwähnt, wie die beschriebenen URIs zu interpretieren sind. Beispielsweise ist für Datei-URLs nicht spezifiziert, wie die Bytesequenzen, aus denen die Pfadsegmente eines Datei-URL bestehen, auf einer bestimmten Plattform abgebildet werden: Die UNO-Umgebung setzt immer voraus, dass Pfadsegmente eines Datei-URL aus UTF-8-kodierten Strings bestehen (die dann in einer plattformspezifischen Weise auf Dateinamen abgebildet werden müssen), wohingegen andere Anwendungen häufig voraussetzen, dass Pfadsegmente von Datei-URLs direkt aus den Bytesequenzen der Dateinamen der Plattform bestehen.

Der ExternalUriReferenceTranslator bietet Methoden, zwischen solchen internen URIs (zum Beispiel den in der UNO-Umgebung verwendeten UTF-8-kodierten Datei-URLs) und externen URIs (zum Beispiel den an Bytesequenzen orientierten Datei-URLs anderer Anwendungen) zu konvertieren. Solche Konvertierungen wirken nur auf Datei-URLs.

```
x = CreateUnoService("com.sun.star.uri.ExternalUriReferenceTranslator")
Print x.translateToExternal("file:///c:/MyDoc.oot")
Print x.translateToInternal("file:/c:/MyDoc.oot")
```

13.20. Fazit

Die in diesem Kapitel vorgestellten Interfaces und Services bieten eine gute Einführung in die Möglichkeiten, die nicht direkt auf einen bestimmten OOo-Dokumenttyp bezogen sind. OOo hat zahlreiche weitere Funktionalitäten, die ich in diesem Kapitel hätte aufführen können, doch an dieser Stelle kann man nicht jeden Punkt erschöpfend behandeln. Betrachten Sie diese Themen und Methoden als Startpunkt zur Erkundung weiterer Potenziale in OOo.

14. Textdokumente

Der Inhalt von Writer-Dokumenten besteht im Wesentlichen aus Text, in Absätze gegliedert. Dieses Kapitel stellt Ihnen geeignete Methoden vor, den Inhalt eines Writer-Dokuments zu bearbeiten, zu durchsuchen, zu formatieren und zu ändern. Erst aber noch ein Rückblick, der auf die meisten Dokumenttypen passt.

In der Grundstruktur haben alle Dokumenttypen zwei Komponenten: die enthaltenen Daten und den Controller, der die Darstellung der Daten bestimmt. Textdokumente enthalten vor allem einfachen formatierten Text. Sie können zusätzlich aber noch anderen Inhalt einschließen, wie Tabellen, Rahmen, Grafiken, Textfelder, Textmarken, Fußnoten, Endnoten, Textbereiche, Verzeichniseinträge, aufgezeichnete Dokumentänderungen, Objekte für Vorlagen und Objekte für Nummerierungen. OOo verwendet dieselben Methoden und Interfaces zur Interaktion mit den meisten dieser Funktionalitäten. Folglich werden Sie, wenn Sie lernen, mit ein paar Inhaltstypen umzugehen, eine solide Basis gewinnen, mit allen zu arbeiten.

Tipp In OOo werden die in einem Dokument enthaltenen Daten „Modell“ genannt. Das grundlegende Modell-Interface ist com.sun.star.frame.XModel.

In OOo werden die in einem Dokument enthaltenen Daten „Modell“ genannt. Jedes Modell hat einen Controller, der für die visuelle Darstellung der Daten verantwortlich ist. Der Controller kennt die Position des visuellen Textcursors, die aktuelle Seite und was aktuell ausgewählt ist.

Tipp Wenn Sie versuchen herauszufinden, welcher Teil der OOo-API für eine bestimmte Aufgabenstellung zu nutzen ist, fragen Sie sich zuerst, ob es eine Frage der Darstellung oder der Daten ist. Beispielsweise ist das Absatzende Teil der Daten, aber eine neue Zeile wird normalerweise vom Controller bestimmt, wenn die Daten formatiert werden.

Jedes Textdokument unterstützt den Service com.sun.star.text.TextDocument. Wenn ich ein Makro schreibe, das benutzerfreundlich sein muss und ein Textdokument benötigt, prüfe ich, ob das Dokument den richtigen Typ hat. Dazu rufe ich die Objektmethode supportsService (s. Listing 343) auf.

Listing 343. *Textdokumente unterstützen den Service com.sun.star.text.TextDocument.*

```
REM Wenn es wirklich wichtig ist, sollten Sie den Dokumenttyp überprüfen,
REM um einen Laufzeitfehler zu vermeiden.
If Not ThisComponent.supportsService("com.sun.star.text.TextDocument") Then
  MsgBox "Das aktuelle Dokument ist kein Writer-Textdokument", 48, "Fehler"
  Exit Sub
End If
```

Ein Interface definiert eine Reihe von Methoden. Wenn ein Objekt ein Interface einbindet, bindet es gleichzeitig jede einzelne Methode ein, die von diesem Interface definiert ist. Ein Service definiert ein Objekt, indem es die eingebundenen Interfaces, die enthaltenen Eigenschaften und die exportierten anderen Services spezifiziert. Ein Service spezifiziert die eingebundenen Methoden auf indirektem Weg, nämlich durch die spezifizierten Interfaces. Die vom Service TextDocument unterstützten Interfaces bieten einen guten Überblick über die verfügbare Funktionalität, s. Tabelle 132.

Tabelle 132. *Von Textdokumenten unterstützte Interfaces.*

Service	Beschreibung
com.sun.star.text.XTextDocument	Das Hauptinterface für Textdokumente.
com.sun.star.text.XBookmarksSupplier	Zugriff auf die Textmarken.
com.sun.star.text.XChapterNumberingSupplier	Kapitelnummerierung.
com.sun.star.text.XDocumentIndexesSupplier	Zugriff auf die Kollektion der Verzeichnisse.
com.sun.star.text.XTextEmbeddedObjectsSupplier	Zugriff auf eingebettete Objekte.

Service	Beschreibung
com.sun.star.text.XEndnotesSupplier	Zugriff auf die Endnoten.
com.sun.star.text.XFootnotesSupplier	Zugriff auf die Fußnoten.
com.sun.star.text.XLineNumberingProperties	Zeilennummerierung.
com.sun.star.text.XPagePrintable	Druck mehrerer Seiten auf einer Druckseite.
com.sun.star.text.XReferenceMarksSupplier	Zugriff auf die Referenzen des Dokuments, die auf Textpositionen in einem Textdokument verweisen.
com.sun.star.text.XTextFieldsSupplier	Zugriff auf enthaltene Feldbefehle.
com.sun.star.text.XTextFramesSupplier	Zugriff auf enthaltene Textrahmen.
com.sun.star.text.XTextGraphicObjectsSupplier	Zugriff auf eingebettete und verlinkte Grafiken.
com.sun.star.text.XTextSectionsSupplier	Zugriff auf enthaltene Textbereiche.
com.sun.star.text.XTextTablesSupplier	Zugriff auf enthaltene Tabellen.
com.sun.star.style.XStyleFamiliesSupplier	Zugriff auf enthaltene Vorlagetypen.
com.sun.star.util.XNumberFormatsSupplier	Zugriff auf enthaltene Zahlenformate.
com.sun.star.util.XRefreshable	Aktualisiert Daten, die aus einer Datenquelle aktualisiert werden können.
com.sun.star.util.XReplaceable	Ersetzt Text, der mit einem Suchdeskriptor gefunden wurde.
com.sun.star.util.XSearchable	Durchsucht einen Textbereich nach einem bestimmten Stringmuster.
com.sun.star.beans.XPropertySet	Zugriff auf die benannten Dokumenteigenschaften.

14.1. Grundbausteine

Wenn Sie mit Textdokumenten arbeiten, werden Sie sehen, dass ein paar einfache Interfaces und Konzepte immer wiederkehren. Diese Grundbausteine beziehen sich aufeinander. Ihre jeweiligen Interfaces sind zirkulär definiert (sie referieren einander). Erfreulicherweise sind die Konzepte intuitiv und daher leicht zu begreifen, auch mit einer nur kurzen Einführung. Dieser Abschnitt legt mit einem kurzen Überblick über diese Grundbausteine die Basis für die weiter unten stehende detaillierte Besprechung.

14.1.1. Der primäre Textinhalt: das Interface XText

Den Textinhalt bildet ein Objekt, in dem das Interface XText eingebunden ist. Der Hauptzweck eines Textobjekts besteht darin, den Textinhalt bereitzustellen, Textcursors für die Bewegung durch den Text zu erzeugen, sowie Inhalte einzufügen und zu entfernen, s. Tabelle 133.

Tabelle 133. *Methoden im Interface com.sun.star.text.XText.*

Methode	Beschreibung
createTextCursor()	Gibt einen TextCursor-Service zum Durchlaufen des Textobjekts zurück.
createTextCursorByRange(XTextRange)	Gibt einen Textcursor zurück, der auf den bestimmten Textbereich (Range) begrenzt ist.
insertString(XTextRange, String, boolean)	Fügt eine Zeichenkette an der Stelle eines bestimmten Textrange in den Text ein. Jedes CR-Zeichen (ASCII 13) fügt einen neuen Absatz ein und jedes LF-Zeichen (ASCII 10) einen Zeilenumbruch. Wenn der boolesche Wert True ist, wird der Text des Range überschrieben, andernfalls wird die Zeichenkette hinter dem Ende des Range eingefügt.

Methode	Beschreibung
insertControlCharacter(XTextRange, Short, boolean)	Fügt ein Steuerzeichen (zum Beispiel Absatzende oder geschütztes Leerzeichen) in den Text ein. Der Wert vom Typ Short Integer ist in der Konstantengruppe com.sun.star.text.ControlCharacter definiert: • PARAGRAPH_BREAK = 0 – Beginn eines neuen Absatzes. • LINE_BREAK = 1 – Beginn einer neuen Zeile innerhalb eines Absatzes. • HARD_HYPHEN = 2 – Fügt einen Bindestrich ein, an dem nicht getrennt wird (geschützter Bindestrich). • SOFT_HYPHEN = 3 – Fügt ein Vorzugstrennzeichen ein für den Fall, dass das Wort am Zeilenende getrennt werden muss (weiches Trennzeichen). • HARD_SPACE = 4 – Fügt ein Leerzeichen ein, an dem zwei Wörter am Zeilenende nicht getrennt werden (geschütztes Leerzeichen). • APPEND_PARAGRAPH = 5 – Fügt einen neuen Absatz an. Wenn der boolesche Wert True ist, wird der Text des Range überschrieben, andernfalls wird das Kontrollzeichen an das Rangeende angefügt.
insertTextContent(XTextRange, XTextContent, boolean)	Fügt Inhalt (Textcontent) ein, der kein String ist, wie zum Beispiel eine Texttabelle, einen Textrahmen oder ein Textfeld. Im allgemeinen sollte der Textcontent vom Textobjekt erzeugt werden. Ist der boolesche Wert True, wird der Text im Textrange überschrieben, andernfalls wird der Textcontent an das Rangeende angefügt.
removeTextContent(XTextContent)	Löscht den bestimmten Textcontent aus dem Textobjekt.

Weil das Interface XText vom Interface XTextRange abstammt, unterstützen alle Objekte, die das Interface XText einbinden, auch die Objektmethoden, die vom Interface XTextRange definiert sind, s.Tabelle 134.

14.1.2. Textranges: das Interface XTextRange

Ein Textrange ist eines der wichtigsten Konzepte in einem Textdokument, weil so viele Interfaces vom Interface XTextRange abstammen. Der Hauptzweck eines Textrange liegt darin, eine Start- und eine Endposition im Text zu definieren, s. Tabelle 134. Die Start- und Endposition eines Textrange können durchaus identisch sein. Wenn das der Fall ist, beschreibt der Textrange eine Position im Text – zum Beispiel den Cursor im Textdokument, wenn kein Text ausgewählt ist. Wenn Start- und Endposition nicht identisch sind, repräsentieren sie eine Textauswahl.

Tabelle 134. *Methoden im Interface com.sun.star.text.XTextRange.*

Methode	Beschreibung
getText()	Gibt das XText-Interface zurück, das die Textposition enthält.
getStart()	Ein Textrange hat eine Start- und eine Endposition. Die Methode getStart() gibt einen Textrange zurück, der nur die Startposition dieses Textrange enthält.
getEnd()	Ein Textrange hat eine Start- und eine Endposition. Die Methode getEnd() gibt einen Textrange zurück, der nur die Endposition dieses Textrange enthält.
setString(String)	Ein Textrange hat eine Start- und eine Endposition. Die Methode setString() ersetzt den gesamten Text zwischen der Start- und der Endposition durch den angegebenen String. Alle Vorlagenzuweisungen werden gelöscht.
getString()	Gibt einen String zurück, der den Text in diesem Textrange enthält.

Jedes XTextRange-Objekt ist mit einem Textobjekt verbunden. Man kann sagen, dass in einem Objekt, das das Interface XText einbindet, ein XTextRange-Objekt steckt. Das Interface XText selbst

stammt vom Interface XTextRange ab. Mit der Objektmethode getText() erhalten Sie ein Textobjekt, das mit dem Textrange verbunden ist.

Tipp Zu vielen Aufgabenstellungen gehören der Zugriff auf eine Textmarke und ihre Ankerposition (ein Textrange) sowie das Einfügen eines Textinhalts an eben dieser Ankerposition. Die Methode setString() ist sicher die schnellste Art, reinen Text an der Ankerposition einzufügen, doch zum Einfügen von anderem Textinhalt braucht man ein Objekt, das das Interface XText unterstützt. Die Methode getText() gibt ein Objekt zurück, das Textinhalt mit Hilfe des Textrange einfügt.

Writer-Dokumente enthalten vorwiegend formatierten Text. Den Zugriff auf den formatierten Text erhält man entweder über die Objektmethode getText() oder in Basic direkt über die Dokumenteigenschaft Text. Meistens greife ich direkt über die Eigenschaft Text zu, weil es weniger Schreibaufwand ist.

```
ThisComponent.Text      'Das Textobjekt des aktuellen Dokuments
ThisComponent.getText() 'Das Textobjekt des aktuellen Dokuments
```

Das Textobjekt des Dokuments bindet das Interface XTextRange ein. Die einfachste Art, alle Textzeichen eines Textdokuments zu erhalten, ist der Aufruf der Objektmethode getString(), s. Tabelle 134. Diese Methode gibt einen einzelnen String zurück, der aus einer Textversion des Dokuments besteht. Jede Zelle einer Texttabelle wird als einzelner Absatz zurückgegeben, und alles ohne Formatierungen. Mit der Objektmethode setString() kann man den gesamten Text eines Dokuments in einem Rutsch setzen – mit setString() ist der bestehende Text komplett verloren! (S. Listing 344).

Listing 344. *Holt und setzt den gesamten Dokumenttext ohne Formatierung.*

```
MsgBox ThisComponent.Text.getString(), 0, "Der Textstring des Dokuments"
ThisComponent.Text.setString("Dies ist der neue Text")
```

Gewöhnlich wird nicht nur der reine Textstring benötigt. Daher holt und setzt man gewöhnlich den Dokumenttext mit anderen Methoden. Im allgemeinen kommt man bei großen und komplexen Dokumenten am besten mit getText() und verwandten Methoden zurecht, weil sie die modulare Behandlung komplexer Dokumente unterstützen.

14.1.3. Einfachen Text einfügen

Mit den bis jetzt präsentierten schlichten Informationen können Sie schon einfachen Textinhalt am Anfang und Ende eines Dokuments einfügen. Jede der Objektmethoden getStart() und getEnd() gibt einen Textrange zurück, der zum Einfügen von Text in ein Textobjekt genutzt werden kann (s. Tabelle 133 und Tabelle 134). Der Code im Listing 345 fügt einen einfachen Text am Anfang des Dokuments und einen neuen Absatz am Ende des Dokuments ein.

Listing 345. *Fügt einfachen Text am Anfang und am Ende des Dokuments ein.*

```
Sub InsertSimpleText
  Dim oText
  oText = ThisComponent.Text

  REM Fügt einfachen Text am Anfang ein
  oText.insertString(oText.getStart(), "Start des Textobjekts." & Chr$(13), False)
  REM Fügt am Ende einen neuen Absatz an
  oText.insertControlCharacter(oText.getEnd(),_
          com.sun.star.text.ControlCharacter.APPEND_PARAGRAPH, False)
End Sub
```

14.1.4. Textinhalt, der kein String ist: der Service TextContent

Der Hauptzweck des Service TextContent besteht darin, ein Objekt (den Textcontent) in dem umgebenden Text zu verankern. Im Prinzip gibt es zwei Arten von Textcontent: den, der Teil des umgebenden Textes ist (zum Beispiel ein Textfeld), und den, der eher wie ein schwebender Rahmen ist (zum Beispiel ein Bild oder eine Grafik).

Mit der Objektmethode getAnchor() finden Sie heraus, wo ein Textcontent im Text verankert ist. Diese Methode gibt einen Textrange mit der Definition der Ankerposition zurück. Für Rahmen-Textcontents muss der Text wissen, wie er um das Objekt herum fließen und wie das Objekt am Text verankert werden soll, s. Tabelle 135. Wie sich Textcontent verhält, der als Zeichen eingefügt wird, sollte leicht nachvollziehbar sein. Mit dem Inhalt geht es wie mit anderen Zeichen auch: er wird zwischen zwei anderen Zeichen im Text mitbewegt. Textcontent, der am Absatz verankert wird, wird jedoch nicht zwingend mit den Zeichen davor und dahinter mitbewegt. Es ist nur gesichert, dass das Objekt auf Dauer dem Absatz anhängt und dass es nicht in den Absatz eingefügt wird. Ich verankere gerne ein Objekt am Absatz, wenn ich will, dass der Text um das Objekt herum fließt, zum Beispiel eine Grafik rechts und der Text links.

Tabelle 135. *Eigenschaften im Service com.sun.star.text.TextContent.*

Eigenschaft	Beschreibung
AnchorType	Enumeration des Typs com.sun.star.text.TextContentAnchorType. Definiert, wie dieser Textcontent dem umgebenden Text beigefügt wird. • AT_PARAGRAPH – Der Anker sitzt an der linken oberen Position des Absatzes. Das Objekt bewegt sich mit dem Absatz. • AS_CHARACTER – Das Textcontent-Objekt wird als Zeichen verankert. Die Größe des Objekts beeinflusst die Höhe der Textzeile, und das Objekt bewegt sich wie ein Zeichen, wenn sich der umgebende Text bewegt. • AT_PAGE – Das Textcontent-Objekt wird an der Seite verankert. Das Objekt bewegt sich nicht, auch wenn sich der Text drumherum bewegt. • AT_FRAME – Das Textcontent-Objekt wird an einem Textrahmen verankert. • AT_CHARACTER – Das Textcontent-Objekt wird an einem Zeichen verankert.Das Objekt bewegt sich, wenn sich das Zeichen bewegt.
AnchorTypes	TextContentAnchorType-Array; mit den Verankerungstypen des Textcontents.
TextWrap	Enumeration des Typs com.sun.star.text.WrapTextMode. Definiert, wie der umgebende Text um das Textcontent-Objekt fließt. • NONE – Der Text fließt nicht um das Objekt herum. • THROUGHT – Der Textfluss ignoriert das Objekt. (Ja, es heißt THROUGHT.) Man kann sich das als THROUGH iT vorstellen in dem Sinne, dass der Text „durch es“ (das Objekt) fließt. • PARALLEL – Der Text fließt links und rechts des Objekts. • DYNAMIC – Die Textformatierung entscheidet über die beste Umfließmethode. • LEFT – Der Text fließt links des Objekts. • RIGHT – Der Text fließt rechts des Objekts.

Das Textobjekt kennt Methoden, Textcontent an bestimmten Positionen einzufügen, s. Tabelle 133. Generell muss der Typ TextContent vor dem Einfügen erst vom Dokument erzeugt werden, s. Listing 346.

Listing 346. *Fügt Textcontent (eine Texttabelle) an das Ende des aktuellen Dokuments an.*

```
Sub InsertSimpleTableAtEnd
  Dim oTable  'Neu erzeugte Tabelle zum Einfügen

  REM Das Dokument muss die Texttabelle erzeugen.
  oTable = ThisComponent.createInstance("com.sun.star.text.TextTable")
```

```
  oTable.initialize(3, 2) 'Drei Zeilen, zwei Spalten

  REM Nun wird die Texttabelle am Ende des Dokuments eingefügt.
  ThisComponent.Text.insertTextContent(_
                            ThisComponent.Text.getEnd(), oTable, False)
End Sub
```

Tipp Ganz allgemein muss der Textcontent-Typ erst vom Dokument erzeugt werden, bevor er eingefügt wird.

Mit der Objektmethode removeTextContent(XTextContent) (s. Tabelle 133) wird ein Textcontent gelöscht. Alternativ kann man auch an seine Stelle einen neuen Textinhalt setzen. Als einfaches Beispiel dient die Methode setString(), die den Textcontent mit einschließt, s. Listing 347.

Listing 347. *Löscht alle Textinhalte des gesamten Dokuments.*

```
ThisComponent.Text.setString("")  'Löscht ein gesamtes Dokument!
```

14.2. Absätze enumerieren

Writer-Dokumente enthälten hauptsächlich formatierten Text, in Absätze gegliedert. Writer-Methoden können sich auf Wörter, Sätze, Absätze und auf ein komplettes Textobjekt beziehen. Für formatierten Text sind Absätze die grundlegende Organisationsform, und Methoden zu Absätzen sind häufig die verlässlichsten, das heißt, sie enthalten weniger Bugs. Die Absätze kann man mit dem im Textobjekt des Dokuments definierten Interface XEnumerationAccess sequenziell auflisten. OOo behandelt Tabellen als einen besonderen Absatztyp, sie werden bei der Enumeration von Absätzen mit zurückgegeben, s. Listing 348.

Listing 348. *Zählt Absätze und Texttabellen.*

```
Sub EnumerateParagraphs
  Dim oEnum              'com.sun.star.container.XEnumerationAccess
  Dim oPar               'Irgendein Absatz
  Dim nPars As Integer   'Anzahl der Absätze
  Dim nTables As Integer 'Anzahl der Tabellen

  REM ThisComponent bezieht sich auf das aktuelle OOo-Dokument.
  REM Text ist eine Eigenschaft von ThisComponent als Textdokument.
  REM Die Objektmethode getText() gibt dasselbe zurück.
  REM createEnumeration() ist eine Objektmethode.
  oEnum = ThisComponent.Text.createEnumeration()
  Do While oEnum.hasMoreElements()
    oPar = oEnum.nextElement()

    REM Der zurückgegebene Absatz ist entweder ein normaler Absatz oder eine Tabelle
    If oPar.supportsService("com.sun.star.text.Paragraph") Then
      nPars = nPars + 1
    ElseIf oPar.supportsService("com.sun.star.text.TextTable") Then
      nTables = nTables + 1
    End If
  Loop
  MsgBox "Anzahl der Absätze : " & CStr(nPars) & Chr$(13) &_
         "Anzahl der Tabellen: " & CStr(nTables) & Chr$(13), 0,_
         "Absatztypen im Dokument"
End Sub
```

Tipp Visual Basic for Applications (VBA) unterstützt den Zugriff auf Absätze über einen Index, OOo nicht.

Bei der Enumeration der Absätze in einem Textdokument werden sowohl normale Absätze als auch Tabellen zurückgegeben. Mit der Methode supportsService() wird ermittelt, ob es ein Absatz oder eine Tabelle ist. Absatzobjekte unterstützen sowohl das Interface XTextRange als auch das Interface XTextContent. TextTable-Objekte hingegen unterstützen nur das Interface XTextContent.

Das Makro im Listing 348 enumeriert Absätze in dem Textobjekt der oberen Ebene. Viele Objekte enthalten ihre eigenen Textobjekte, zum Beispiel jede Tabellenzelle und jeder Rahmen.

14.2.1. Absatzeigenschaften

Absätze haben zahlreiche spezifische Eigenschaften, gekapselt in Services. Die primär auf den gesamten Absatz bezogenen Eigenschaften sind im Service ParagraphProperties gebündelt, s. Tabelle 136.

Tipp Der Service Paragraph ist nicht der einzige Service, der ParagraphProperties unterstützt. Auch andere Services, vor allem solche, die auch ein Textrange sind, unterstützen Absatzeigenschaften. Techniken, mit denen man die Absatzeigenschaften in Absätzen modifiziert, funktionieren auch für diese anderen Services.

Tabelle 136. *Eigenschaften im Service com.sun.star.style.ParagraphProperties.*

Eigenschaft	Beschreibung
ParaAdjust	Ausrichtung des Absatzes als Wert der Enumeration com.sun.star.style.ParagraphAdjust: • LEFT – Linksbündig. • RIGHT – Rechtsbündig. • CENTER – Zentriert. • BLOCK – Blocksatz mit Ausnahme der letzten Zeile. • STRETCH – Blocksatz inklusive der letzten Zeile.
ParaLastLineAdjust	Ausrichtung der letzten Zeile, falls ParaAdjust auf BLOCK gesetzt ist.
ParaLineSpacing	Zeilenabstand. Die Eigenschaft ist ein Struct des Typs com.sun.star.style.LineSpacing, das zwei Eigenschaften des Typs Short enthält. Die Eigenschaft Height ist für die Höhenangabe, und die Eigenschaft Mode bestimmt, wie die Höhe zu benutzen ist. Verwendet werden die in der Konstantengruppe com.sun.star.style.LineSpacingMode definierten Werte: • PROP = 0 – Die Höhe ist proportional. • MINIMUM = 1 – Die Höhe ist die Mindestzeilenhöhe. • LEADING = 2 – Die Höhe ist der Abstand zur Vorzeile. • FIX = 3 – Die Höhe ist fix.
ParaBackColor	Hintergrundfarbe des Absatzes als Long Integer.
ParaBackTransparent	Falls True, wird der Absatzhintergrund transparent.
ParaBackGraphicURL	URL der Hintergrundgrafik des Absatzes.
ParaBackGraphicFilter	Name des Grafikfilters für die Hintergrundgrafik des Absatzes.

Eigenschaft	Beschreibung
ParaBackGraphicLocation	Positionierung der Hintergrundgrafik als Wert der Enumeration com.sun.star.style.GraphicLocation: • NONE – Noch nicht zugewiesen. • LEFT_TOP – In der oberen linken Ecke. • MIDDLE_TOP – In der Mitte des oberen Randes. • RIGHT_TOP – In der oberen rechten Ecke. • LEFT_MIDDLE – In der Mitte des linken Randes. • MIDDLE_MIDDLE – In der Mitte des umgebenden Objekts. • RIGHT_MIDDLE – In der Mitte des rechten Randes. • LEFT_BOTTOM – In der unteren linken Ecke. • MIDDLE_BOTTOM – In der Mitte des unteren Randes. • RIGHT_BOTTOM – In der unteren rechten Ecke. • AREA – Skaliert über die gesamte umgebende Fläche. • TILED – Gekachelt über das umgebende Objekt.
ParaExpandSingleWord	Falls True, können einzelne Wörter ausgetrieben (gedehnt) werden.
ParaLeftMargin	Der linke Absatzeinzug in 1/100 mm als Long Integer.
ParaRightMargin	Der rechte Absatzeinzug in 1/100 mm als Long Integer.
ParaTopMargin	Der obere Absatzabstand in 1/100 mm als Long Integer. Der Abstand zwischen zwei Absätzen ist das Maximum aus dem unteren Abstand des vorhergehenden Absatzes und dem oberen Abstand des folgenden Absatzes.
ParaBottomMargin	Der untere Absatzabstand in 1/100 mm als Long Integer. Der Abstand zwischen zwei Absätzen ist das Maximum aus dem unteren Abstand des vorhergehenden Absatzes und dem oberen Abstand des folgenden Absatzes.
ParaLineNumberCount	Falls True, wird der Absatz in die Zeilennummerierung mit aufgenommen.
ParaLineNumberStartValue	Startwert der Zeilennummerierung als Long Integer.
PageDescName	String, der vor dem Absatz einen Seitenumbruch bewirkt. Die neue Seite nutzt diesen String als Namen der Seitenvorlage.
PageNumberOffset	Neue Seitennummer bei einem Seitenumbruch.
ParaRegisterModeActive	Falls True, wird der Modus Registerhaltigkeit aktiviert. Dabei hat jede Zeile dieselbe Höhe. Dieser Modus ist nur aktiv, wenn auch in der Seitenvorlage des Absatzes Registerhaltigkeit eingeschaltet ist.
ParaTabStops	Tabulatoren für diesen Absatz. Ein Array von Structs des Typs com.sun.star.style.TabStop. Das Struct enthält die folgenden Eigenschaften: • Position – Position relativ zur linken Umrandung. Long Integer • Alignment – Ausrichtung des Textranges vor dem Tabulator als Wert der Enumeration com.sun.star.style.TabAlign. Die gültigen Werte sind LEFT, RIGHT, CENTER, DECIMAL und DEFAULT. • DecimalChar – Das Dezimalzeichen. • FillChar – Füllzeichen für den Leerraum zwischen den Textranges.
ParaStyleName	Name der aktuellen Absatzvorlage.
DropCapFormat	Struct (com.sun.star.style.DropCapFormat), in dem die Bedingungen für die Anzeige einer Absatzinitialen geregelt sind. Enthält folgende Eigenschaften: • Lines – Anzahl der Zeilen für die Initiale. • Count – Anzahl der Zeichen für die Initiale. • Distance – Abstand der Initialen vom folgenden Text.

Eigenschaft	Beschreibung
DropCapWholeWord	Falls True, wird DropCapFormat auf das ganze erste Wort angewendet.
ParaKeepTogether	Falls True, wird nach diesem Absatz kein Seiten- oder Spaltenwechsel vorgenommen – zum Beispiel, damit eine Titelzeile nicht in der letzten Zeile einer Seite oder einer Spalte steht.
ParaSplit	Falls False, wird verhindert, dass mitten im Absatz ein Seiten- oder Spaltenwechsel vorgenommen wird.
NumberingLevel	Gliederungsebene des Absatzes.
NumberingRules	Die auf diesen Absatz anzuwendenden Nummerierungsregeln. Dieses Objekt bindet das Interface com.sun.star.container.XIndexReplace ein.
NumberingStartValue	Startwert der Nummerierung, falls ParaIsNumberingRestart True ist.
ParaIsNumberingRestart	Falls True, beginnt die Nummerierung mit dem aktuellen Absatz (s. NumberingStartValue).
NumberingStyleName	Name der verwendeten Nummerierungsvorlage.
ParaOrphans	Mindestanzahl der Zeilen eines Absatzes am Ende einer Seite, wenn der Absatz auf der nächsten Seite weitergeht (Schusterjungenregelung).
ParaWidows	Mindestanzahl der Zeilen eines Absatzes am Anfang einer Seite, wenn der Absatz auf der vorigen Seite begonnen wurde (Hurenkinderregelung).
ParaShadowFormat	Schattenformat des Absatzes als Struct com.sun.star.table.ShadowFormat: • Location – Position als Enumeration com.sun.star.table.ShadowLocation. Gültige Werte sind NONE, TOP_LEFT, TOP_RIGHT, BOTTOM_LEFT und BOTTOM_RIGHT. • ShadowWidth – Die Schattenbreite als Integer. • IsTransparent – Falls True, ist der Schatten transparent. • Color – Die Schattenfarbe als Long Integer.
LeftBorder	Die linke Umrandung als Struct com.sun.star.table.BorderLine: • Color – Die Linienfarbe. • InnerLineWidth – Die Breite der inneren einer Doppellinie (in 1/100 mm). Falls Null, wird eine einfache Linie gezogen. • OuterLineWidth – Die Breite einer einfachen Linie oder der äußeren einer Doppellinie (in 1/100 mm). Falls Null, wird keine Linie gezogen. • LineDistance – Der Abstand zwischen der inneren und der äußeren Linie einer Doppellinie (in 1/100 mm).
RightBorder	Die rechte Umrandung (s. LeftBorder).
TopBorder	Die obere Umrandung (s. LeftBorder).
BottomBorder	Die untere Umrandung (s. LeftBorder).
BorderDistance	Abstand zwischen Umrandung und Inhalt (in 1/100 mm).
LeftBorderDistance	Abstand zwischen linker Umrandung und Inhalt (in 1/100 mm).
RightBorderDistance	Abstand zwischen rechter Umrandung und Inhalt (in 1/100 mm).
TopBorderDistance	Abstand zwischen oberer Umrandung und Inhalt (in 1/100 mm).
BottomBorderDistance	Abstand zwischen unterer Umrandung und Inhalt (in 1/100 mm).

Eigenschaft	Beschreibung
BreakType	Der mit dem Absatz verwendete Umbruchtyp als Wert der Enumeration com.sun.star.style.BreakType: • NONE – Kein Seiten- oder Spaltenumbruch. • COLUMN_BEFORE – Spaltenumbruch vor dem aktuellen Absatz. Der aktuelle Absatz ist damit der erste in einer Spalte. • COLUMN_AFTER – Spaltenumbruch hinter dem aktuellen Absatz. Der aktuelle Absatz ist damit der letzte in einer Spalte. • COLUMN_BOTH – Spaltenumbruch vor und hinter dem aktuellen Absatz. Der aktuelle Absatz ist damit der einzige in einer Spalte. • PAGE_BEFORE – Seitenumbruch vor der aktuellen Seite. Der aktuelle Absatz ist damit der erste auf einer Seite. • PAGE_AFTER – Seitenumbruch hinter der aktuellen Seite. Der aktuelle Absatz ist damit der letzte auf einer Seite. • PAGE_BOTH – Seitenumbruch vor und hinter der aktuellen Seite. Der aktuelle Absatz ist damit der einzige auf einer Seite.
DropCapCharStyleName	Name der Zeichenvorlage für Absatzinitialen.
ParaFirstLineIndent	Einrückung der ersten Absatzzeile.
ParaIsAutoFirstLineIndent	Falls True, wird die erste Zeile automatisch eingerückt.
ParaIsHyphenation	Falls True, wird die automatische Silbentrennung eingeschaltet.
ParaHyphenationMaxHyphens	Höchstanzahl aufeinander folgender Trennstellen für jedes Wort im aktuellen Absatz.
ParaHyphenationMaxLeadingChars	Anzahl der Zeichen, die vor einer Trennstelle erhalten bleiben müssen.
ParaHyphenationMaxTrailingChars	Anzahl der Zeichen, die hinter einer Trennstelle erhalten bleiben müssen.
ParaVertAlignment	Die vertikale Ausrichtung des Absatzes als Wert der Enumeration com.sun.star.text.ParagraphVertAlign: • AUTOMATIC = 0 – Im automatischen Modus wird horizontaler Text an der Grundlinie ausgerichtet, genauso wie um 90 Grad gedrehter Text. Um 270 Grad gedrehter Text wird zentriert ausgerichtet. • BASELINE = 1 – Der Text wird an der Grundlinie ausgerichtet. • TOP = 2 – Der Text wird oben ausgerichtet. • CENTER = 3 – Der Text wird zentriert ausgerichtet. • BOTTOM = 4 – Der Text wird unten ausgerichtet.
ParaUserDefinedAttributes	Speichert XML-Attribute, die von den automatischen Vorlagen innerhalb der XML-Dateien gesichert und wiederhergestellt werden. Das Objekt bindet das Interface com.sun.star.container.XNameContainer ein.
NumberingIsNumber	Falls True, besteht die Absatznummerierung aus einer Zahl ohne Symbol. Diese Eigenschaft ist leer, wenn der Absatz nicht zu einer Nummerierungsfolge gehört.
ParaIsConnectBorder	Falls True, werden die Absatzumrandungen mit dem vorigen Absatz verschmolzen, wenn die Umrandungen identisch sind. Diese Eigenschaft kann leer sein.

Tipp Absatzeigenschaften werden gewöhnlich über Absatzvorlagen gesetzt – so sollte es jedenfalls sein.

Viele der Eigenschaften in der Tabelle 136 sind Structs. Änderungen darin bedürfen besonderer Beachtung, denn ein Struct wird nicht als Referenz, sondern als Wert kopiert. Nehmen wir einmal eine Struct-Eigenschaft: ParaLineSpacing. Obwohl der Code im Listing 349 korrekt aussieht, funktioniert er nicht. Dieser Fehler wird sehr häufig von Basic-Programmierern gemacht.

Listing 349. *Die direkte Modifizierung eines Structs in einer Variablen funktioniert* ***nicht****.*

```
oPar.ParaLineSpacing.Mode = com.sun.star.style.LineSpacing.LEADING
```

Der Code im Listing 349 funktioniert nicht, weil der Codeabschnitt „oPar.ParaLineSpacing“ eine Kopie des Structs gemacht hat. Der Modus wird zwar gesetzt, aber nur in der Kopie. Das Original bleibt unverändert. Der Code im Listing 350 zeigt den korrekten Weg, den Wert einer Struct-Eigenschaft zu ändern. Eine Kopie des Structs wird in der Variablen v abgelegt, die dann modifiziert wieder zurück kopiert wird.

Listing 350. *Die Modifizierung eines Structs funktioniert, wenn eine Kopie gemacht und diese wieder zurück kopiert wird.*

```
v = oPar.ParaLineSpacing
v.Mode = com.sun.star.style.LineSpacing.LEADING
oPar.ParaLineSpacing = v
```

Einen Seitenumbruch einfügen

Um einen Seitenumbruch einzufügen, setzen Sie die Eigenschaft PageDescName auf den Namen der nach dem Seitenumbruch zu verwendenden Seitenvorlage. Das kann dieselbe Vorlage wie die der aktuellen Seite sein. Allein der Umstand, dass die Eigenschaft PageDescName gesetzt ist – nicht, dass ein neuer Wert darin steht –, bewirkt den Seitenumbruch. Allerdings muss der Name der Seitenvorlage im Dokument existieren, sonst wird der Umbruch nicht eingefügt. Zusammen mit dem Seitenumbruch können Sie auch in der Eigenschaft PageNumberOffset eine neue Seitenzählung setzen, s. Listing 351.

Tipp Nur selten wird ein Seitenumbruch während der Enumeration der Absätze eingefügt. Viel häufiger nutzt man dazu einen Textcursor oder einen Textrange. Einen Seitenumbruch können Sie mit jedem Service einfügen, der die Absatzeigenschaften unterstützt.

Listing 351. *Fügt einen Seitenumbruch am Anfang des letzten Absatzes ein.*

```
Sub SetPageBreakAtEndFromEnumeration
  Dim oEnum    'com.sun.star.container.XEnumerationAccess
  Dim oParTest 'Irgendein Absatz
  Dim oPar     'Objekt des letzten Absatzes

  REM Sucht den letzten Absatz.
  oEnum = ThisComponent.Text.createEnumeration()
  Do While oEnum.hasMoreElements()
    oParTest = oEnum.nextElement()
    If oParTest.supportsService("com.sun.star.text.Paragraph") Then
      oPar = oParTest
    End If
  Loop

  REM Beachten Sie, dass dies den Namen der Seitenvorlage nicht ändert.
  oPar.PageDescName = oPar.PageStyleName

  REM Setzt die neue Seitenzahl auf 7.
  oPar.PageNumberOffset = 7
End Sub
```

Die Absatzvorlage zuweisen

Die Eigenschaft ParaStyleName kennzeichnet die Formatvorlage für diesen Absatz. Diese Eigenschaft kann direkt gesetzt werden.

Listing 352. *Enumeriert die Absatzvorlagen im aktuellen Dokument.*

```
Sub EnumerateParStyles()
  Dim oEnum    'com.sun.star.container.XEnumerationAccess
  Dim oCurPar  'Objekt des jeweils letzten Absatzes
  Dim s$       'Allgemeiner String
  Dim i%       'Zähler für die Absätze

  oEnum = ThisComponent.Text.createEnumeration()
  REM Die ersten 15 Absätze sollten als Demonstration reichen.
  Do While oEnum.hasMoreElements() And (i < 15)
    oCurPar = oEnum.nextElement()
    If oCurPar.supportsService("com.sun.star.text.Paragraph") Then
      s = s & oCurPar.ParaStyleName & Chr$(10)
    End If
    i = i + 1
  Loop
  MsgBox s
End Sub
```

14.2.2. Zeicheneigenschaften

In Absätzen gibt es eine Anzahl von Eigenschaften, die sich auf Zeichen beziehen. Genau wie die absatzspezifischen Eigenschaften sind auch diese optional und in Services gebündelt. Die primär zeichenspezifischen Eigenschaften findet man im Service CharacterProperties, s. Tabelle 137.

Tipp Viele der Eigenschaften werden durch einen Wert in einer Konstantengruppe repräsentiert. Konstantengruppen verwenden aussagekräftige Namen für konstante Werte. Die CharFontFamily zum Beispiel akzeptiert den Wert com.sun.star.awt.FontFamily.ROMAN für eine Roman-Schriftart mit Serifen. Man könnte auch den Wert 3 nehmen. In fast allen Fällen ist der erste Wert 0, der zweite 1 und so weiter. Code, der die aussagekräftigen Namen verwendet, ist leichter zu lesen und zu verstehen.

Tabelle 137. *Eigenschaften im Service com.sun.star.style.CharacterProperties.*

Eigenschaft	Beschreibung
CharFontName	Name der Schriftart in westlichem Text. Darf eine durch Komma getrennte Liste von Namen sein.
CharFontStyleName	Name des Schriftschnitts.
CharFontFamily	Schriftartfamilie als Wert der Konstantengruppe com.sun.star.awt.FontFamily: • DONTKNOW = 0 – Unbekannte Schriftartfamilie. • DECORATIVE = 1 – Familie von Schmuckschriften. • MODERN = 2 – Familie von Modern-Schriftarten. • ROMAN = 3 – Familie von Roman-Schriftarten mit Serifen. • SCRIPT = 4 – Familie von Schreibschriftarten. • SWISS = 5 – Familie von Roman-Schriftarten ohne Serifen. • SYSTEM = 6 – Familie von Systemschriftarten.
CharFontCharSet	Zeichensatz der Schriftart als Wert der Konstantengruppe com.sun.star.awt.CharSet. Die Namen der Werte sind selbsterklärend: DONTKNOW, ANSI, MAC, IBMPC_437 (= IBM-PC-Zeichensatznummer 437), IBMPC_850, IBMPC_860, IBMPC_86, IBMPC_863, IBMPC_865, SYSTEM und SYMBOL.
CharFontPitch	Zeichenbreite der Schriftart als Wert der Konstantengruppe com.sun.star.awt.FontPitch. Die Namen der Werte sind selbsterklärend: DONTKNOW, FIXED und VARIABLE.

Eigenschaft	Beschreibung
CharColor	Textfarbe als Long Integer.
CharEscapement	Prozentuale Angabe der Hoch-/Tiefstellung als Short Integer. Negative Werte bewirken Tiefstellung.
CharHeight	Zeichenhöhe in Punkt als Dezimalzahl.
CharUnderline	Unterstreichungstyp als Wert der Konstantengruppe com.sun.star.awt.FontUnderline: • NONE = 0 – Keine Unterstreichung. • SINGLE = 1 – Einfache Linie. • DOUBLE = 2 – Doppellinie. • DOTTED = 3 – Punktierte Linie. • DONTKNOW = 4 – Unbekannte Unterstreichung. • DASH = 5 – Gestrichelte Linie. • LONGDASH = 6 – Lang gestrichelte Linie. • DASHDOT = 7 – Linie als Strich-Punkt-Folge. • DASHDOTDOT = 8 – Linie als Strich-Punkt-Punkt-Folge. • SMALLWAVE = 9 – Kleine Welle. • WAVE = 10 – Welle. • DOUBLEWAVE = 11 – Doppelte Welle. • BOLD = 12 – Fette Linie. • BOLDDOTTED = 13 – Fette Punkte. • BOLDDASH = 14 – Fette Striche. • BOLDLONGDASH = 15 – Fette lange Striche. • BOLDDASHDOT = 16 – Fette Strich-Punkt-Folge. • BOLDDASHDOTDOT = 17 – Fette Strich-Punkt-Punkt-Folge. • BOLDWAVE = 18 – Fette Welle.
CharWeight	Schriftstärke als Wert der Konstantengruppe com.sun.star.awt.FontWeight: • DONTKNOW = 0.000 – Nicht spezifiziert / unbekannt. • THIN = 50.00 – 50% der Schriftstärke. • ULTRALIGHT = 60.00 – 60% der Schriftstärke. • LIGHT = 75.00 – 75% der Schriftstärke. • SEMILIGHT = 90.00 – 90% der Schriftstärke. • NORMAL = 100.00 – Normale Schriftstärke (100%). • SEMIBOLD = 110.00 – 110% der Schriftstärke. • BOLD = 150.00 – 150% der Schriftstärke. • ULTRABOLD = 175.00 – 175% der Schriftstärke. • BLACK = 200.00 – 200% der Schriftstärke.
CharPosture	Schriftneigung als Wert der Enumeration com.sun.star.awt.FontSlant: • NONE – Keine Neigung, normaler Text. • OBLIQUE – Schräg (nicht als Schriftschnitt entworfen). • ITALIC – Kursiv (als Schriftschnitt entworfen). • DONTKNOW – Unbekannt. • REVERSE_OBLIQUE – Umgekehrt schräg (nicht als Schriftschnitt entworfen). • REVERSE_ITALIC – Umgekehrt kursiv (als Schriftschnitt entworfen).
CharAutoKerning	Falls True, werden die Unterschneidungstabellen für die aktuelle Schriftart verwendet. Automatisches Unterschneiden passt die Abstände zwischen bestimmten Zeichenpaaren zur besseren Lesbarkeit an.

Eigenschaft	Beschreibung
CharBackColor	Zeichenhintergrundfarbe als Long Integer.
CharBackTransparent	Falls True, ist die Zeichenhintergrundfarbe transparent.
CharCaseMap	Schriftauszeichnung (Groß-/Kleinschreibung) als Wert der Konstantengruppe com.sun.star.style.CaseMap. Der aktuelle Text wird nicht geändert – nur die Darstellung. • NONE = 0 – Keine Änderung. Der übliche Wert. • UPPERCASE = 1 – Alle Zeichen in Großbuchstaben. • LOWERCASE = 2 – Alle Zeichen in Kleinbuchstaben. • TITLE = 3 – Der erste Buchstabe eines Wortes groß geschrieben. • SMALLCAPS = 4 – Alle Zeichen in Großbuchstaben, aber in einer kleineren Schriftart (Kapitälchen).
CharCrossedOut	Falls True, werden die Zeichen durchgestrichen.
CharFlash	Falls True, werden die Zeichen blinkend dargestellt.
CharStrikeout	Durchstreichungstyp als Wert der Konstantengruppe com.sun.star.awt.FontStrikeout: • NONE = 0 – Keine Durchstreichung. • SINGLE = 1 – Einfache Linie. • DOUBLE = 2 – Doppellinie. • DONTKNOW = 3 – Nicht spezifiziert. • BOLD = 4 – Fette Linie. • SLASH = 5 – Schrägstriche. • X = 6 – Buchstabe X.
CharWordMode	Falls True, werden weiße Zeichen (Leerzeichen und Tabulatoren) nicht mit durchgestrichen oder unterstrichen.
CharKerning	Wert der Unterschneidung als Short Integer.
CharLocale	Gebietsschema der Zeichen als Struct com.star.lang.Locale.
CharKeepTogether	Falls True, versucht OOo, die Zeichengruppe auf ein und derselben Zeile zu belassen. Ein erforderlicher Zeilensprung wird vor der Zeichengruppe vorgenommen.
CharNoLineBreak	Falls True, ignoriert OOo einen Zeilensprung in der Zeichengruppe. Ein erforderlicher Zeilensprung wird hinter der Zeichengruppe vorgenommen, so dass sie über die Zeilenumrandung hinausragen kann.
CharShadowed	Falls True, werden die Zeichen mit Schatteneffekt formatiert und dargestellt.
CharFontType	Format der Schriftart als Wert der Konstantengruppe com.sun.star.awt.FontType: • DONTKNOW = 0 – Unbekannt. • RASTER = 1 – Rastergrafik (Bitmap). • DEVICE = 2 – Ausgabegerätspezifisch, zum Beispiel Druckerschriftart. • SCALABLE = 3 – Skalierbar.
CharStyleName	Name der Zeichenvorlage als String.
CharContoured	Falls True, werden die Zeichen mit Kontur (3-D-Effekt) formatiert und dargestellt.
CharCombineIsOn	Falls True, wird der Text zweizeilig formatiert und dargestellt, so dass eine normale Absatzzeile aus zwei Textzeilen besteht. Der String in CharCombinePrefix wird in normaler Größe vorangesetzt, der Text in CharCombineSuffix wird in normaler Größe angehängt.
CharCombinePrefix	Präfix (meist eine Klammer), das mit CharCombineIsOn verwendet wird.
CharCombineSuffix	Suffix (meist eine Klammer), das mit CharCombineIsOn verwendet wird.

Eigenschaft	Beschreibung
CharRelief	Relief als Wert der Konstantengruppe com.sun.star.text.FontRelief: • NONE = 0 – Kein Relief, normaler Text. • EMBOSSED = 1 – Erhaben. • ENGRAVED = 2 – Vertieft.
CharEmphasize	Typ und Position des Hervorhebungskennzeichens in asiatischen Texten als Wert der Konstantengruppe com.sun.star.text.FontEmphasis. Das Zeichen wird über oder unter dem Text platziert (bzw. rechts oder links vom Text in vertikaler Darstellung). • NONE = 0 – Keine Hervorhebung. • DOT_ABOVE = 1 – Punkt über (rechts neben) dem Text. • CIRCLE_ABOVE = 2 – Kreisring über (rechts neben) dem Text. • DISK_ABOVE = 3 – Kreisscheibe über (rechts neben) dem Text. • ACCENT_ABOVE = 4 – Akzent über (rechts neben) dem Text. • DOT_BELOW = 11 – Punkt unter (links neben) dem Text. • CIRCLE_BELOW = 12 – Kreisring unter (links neben) dem Text. • DISK_BELOW = 13 – Kreisscheibe unter (links neben) dem Text. • ACCENT_BELOW = 14 – Akzent unter (links neben) dem Text.
RubyText	Text, der als Ruby verwendet wird. „Ruby bezeichnet ein Anmerkungssystem, bei dem der Text zusammen mit seiner Anmerkung in einer Zeile erscheint. Dies wird vor allem bei japanischen und chinesischen Texten zur Angabe der Aussprache genutzt, da die dort verwendeten chinesischen Schriftzeichen in vielen Fällen keinen Aufschluss über die tatsächliche Aussprache geben. Im Japanischen nennt sich diese Nutzung auch Furigana. Hierbei wird im Allgemeinen zusätzlich zu dem eigentlichen Text, der in chinesischen Schriftzeichen verfasst ist, eine phonetische Schrift verwendet. Im Japanischen sind dies die Kana, im Chinesischen hauptsächlich Zhuyin oder Pinyin.“ (Zitat aus Wikipedia deutsch, Stichwort „Ruby Annotation“).
RubyAdjust	Ausrichtung des Ruby-Texts als Wert der Enumeration com.sun.star.text.RubyAdjust: • LEFT – Linksbündig. • CENTER – Zentriert. • RIGHT – Rechtsbündig. • BLOCK – Blocksatz (gedehnt, bündig zu beiden Enden). • INDENT_BLOCK – Blocksatz mit schmalen Einrückungen auf beiden Seiten.
RubyCharStyleName	Name der Zeichenvorlage für den Ruby-Text.
RubyIsAbove	Falls True, wird der Ruby-Text über dem Text angezeigt (rechts bei vertikaler Darstellung).
CharRotation	Rotation des Zeichens in Grad als Short Integer. Nicht alle OOo-Varianten unterstützen alle Werte.
CharRotationIsFitToLine	Falls True, versucht OOo, den rotierten Text der umgebenden Zeilenhöhe anzupassen.
CharScaleWidth	Skalierung für hoch- oder tiefgestellte Zeichen als Prozentangabe (Short Integer).
HyperLinkURL	URL eines Hyperlinks als String.
HyperLinkTarget	Name des Zielfensters für einen Hyperlink als String.
HyperLinkName	Name eines Hyperlinks als String.
VisitedCharStyleName	Name der Zeichenvorlage für besuchte Hyperlinks als String.
UnvisitedCharStyleName	Name der Zeichenvorlage für nicht besuchte Hyperlinks als String.
CharEscapementHeight	Relative Höhe der hoch- oder tiefgestellten Zeichen als Prozentangabe (Short Integer). Negative Werte werden akzeptiert, sind aber nicht wirklich erwünscht.
CharNoHyphenation	Falls True, kann das Wort an diesem Zeichen nicht getrennt werden.
CharUnderlineColor	Farbe der Unterstreichung als Long Integer.

Eigenschaft	Beschreibung
CharUnderlineHasColor	Falls True, wird CharUnderlineColor für die Unterstreichung genutzt.
CharStyleNames	Array der Namen der im Text verwendeten Zeichenvorlagen. Die Reihenfolge ist nicht unbedingt relevant.

Tipp Wenn eine Eigenschaft den Wert DONTKNOW unterstützt, dann gilt die Eigenschaft als Hinweis, dass bestimmte Operationen effizienter ausgeführt werden oder dass ein passender Ersatz gesucht wird, wenn der gewünschte Wert nicht verfügbar ist. Wenn zum Beispiel eine bestimmte Schriftart nicht zur Verfügung steht, kann aus der CharFontFamily eine Schriftart des korrekten Typs gewählt werden.

Der Code im Listing 353 demonstriert die Änderung der Zeicheneigenschaften am Beispiel der FontRelief-Eigenschaft. Sie wird geändert und danach wieder zurückgesetzt.

Listing 353. *Setzt FontRelief und setzt den Wert wieder zurück.*

```
Sub ViewFontRelief
  Dim oEnum 'com.sun.star.container.XEnumerationAccess
  Dim oPar  'Irgenein Absatz
  Dim i%    'Allgemeine Zählvariable
  Dim s$

  oEnum = ThisComponent.Text.createEnumeration()
  Do While oEnum.hasMoreElements()
    oPar = oEnum.nextElement()

    REM Der zurückgegebene Absatz ist entweder ein echter Absatz oder eine Tabelle.
    If oPar.supportsService("com.sun.star.text.Paragraph") Then
      i = i + 1
      oPar.CharRelief = i Mod 3
    End If
  Loop

  MsgBox "Das Dokument verwendet nun als Zeichenrelief NONE, EMBOSSED und ENGRAVED"

  oEnum = ThisComponent.Text.createEnumeration()
  Do While oEnum.hasMoreElements()
    oPar = oEnum.nextElement()

    REM Der zurückgegebene Absatz ist entweder ein echter Absatz oder eine Tabelle.
    If oPar.supportsService("com.sun.star.text.Paragraph") Then
      i = i + 1
      oPar.CharRelief = com.sun.star.text.FontRelief.NONE
    End If
  Loop
End Sub
```

14.2.3. Absatzteile enumerieren

Absätze enthalten häufig unterschiedlich formatierten Text – zum Beispiel kann ein einzelnes Wort **fett** dargestellt sein, mitten in einem Satz normal formatierter Zeichen. Genauso wie man Absätze in einem Dokument enumerieren kann, so kann man auch die einzelnen Teile eines Absatzes enumerieren. Der Textinhalt jedes einzelnen enumerierten Teils ist gleich formatiert und vom selben Typ. Tabelle 138 zeigt die vom Service TextPortion direkt unterstützten Eigenschaften. Der Service TextPor-

tion exportiert den Service TextRange und damit auch die Absatzeigenschaften der Tabelle 136 und die Zeicheneigenschaften der Tabelle 137.

Tipp Ein Objekt, das Absatz- oder Zeicheneigenschaften unterstützt, bietet gewöhnlich auch einen Weg, einen Textrange festzulegen. Wenn eine bestimmte Eigenschaft innerhalb des Textrange unterschiedliche Werte hat, kann sie normalerweise nicht gesetzt werden. Beispielsweise kann ein Textrange aus mehr als einem Absatz bestehen. Wenn sich nicht alle enthaltenen Absätze auf dieselbe Absatzvorlage stützen, steht dem Textrange die Absatzvorlage-Eigenschaft nicht zur Verfügung. Wenn jedoch der Textrange auf Absätze begrenzt ist, die alle dieselbe Vorlage nutzen, wird diesem Textrange auch die Absatzvorlage-Eigenschaft verfügbar sein.

Tabelle 138. *Eigenschaften im Service com.sun.star.text.TextPortion.*

Eigenschaft	Beschreibung
TextPortionType	Der Typ des Absatzteils als String. Gültige Typnamen sind: • Text – String. • TextField – Textfeld. • TextContent – Als Zeichen oder an einem Zeichen verankerter Textcontent, der nicht wirklich Teil des Absatzes ist – zum Beispiel ein Textrahmen oder ein grafisches Objekt. Seit OOo 1.1.0 wird statt „TextContent“ der Typ „Frame“ zurückgegeben. Das OOo-Team führt diesen Bug unter der Nummer #24444. • Frame – ein Frame. Wird statt des Typs „TextContent“ zurückgegeben. • Footnote – Fußnote oder Endnote. • ControlCharacter – Steuerzeichen. • ReferenceMark – Querverweis. • DocumentIndexMark – Stichworteintrag. • Bookmark – Textmarke. • Redline – Rot markierter Teil, als Resultat der Änderungsaufzeichnung. • Ruby – Ruby-Attribut (in asiatischem Text).
ControlCharacter	Das Steuerzeichen als Short Integer, wenn der Absatzteil ein Steuerzeichen enthält.
Bookmark	Wenn der Text eine Textmarke enthält, ist dies eine Referenz auf die Textmarke. Die Eigenschaft bindet das Interface com.sun.star.text.XTextContent ein.
IsCollapsed	Falls True, ist der Absatzteil ein Punkt (d.h. ein Range mit identischer Start- und Endposition).
IsStart	Falls True, ist der Absatzteil ein Startteil, wenn nämlich zwei Absatzteile nötig sind, um ein Objekt zu umschließen. Ein DocumentIndexMark zum Beispiel ist zweigeteilt: jeweils ein Start- und ein Endteil umschließen den indexierten Text.

Das Makro im Listing 354 zeigt, wie der Textinhalt innerhalb eines Absatzes enumeriert wird. Es nummeriert die Absätze und gibt die Typen der jeweiligen Absatzteile aus. Die Absatzzählung ist errechnet, sie ist keine Absatzeigenschaft.

Listing 354. *Listet die Typen der Absatzteile auf.*

```
Sub EnumerateTextSections
  Dim oParEnum           'Absatzenumerator
  Dim oSecEnum           'Absatzteilenumerator
  Dim oPar               'Der aktuelle Absatz
  Dim oParSection        'Der aktuelle Absatzteil
  Dim nPars As Integer   'Absatzzählung
  Dim s$

  oParEnum = ThisComponent.Text.createEnumeration()
  Do While oParEnum.hasMoreElements()
    oPar = oParEnum.nextElement()
```

```
      If oPar.supportsService("com.sun.star.text.Paragraph") Then
        nPars = nPars + 1
        oSecEnum = oPar.createEnumeration()
        s = s & nPars & ":"
        Do While oSecEnum.hasMoreElements()
          oParSection = oSecEnum.nextElement()
          s = s & oParSection.TextPortionType & ":"
        Loop
        s = s & Chr$(10)
        If nPars Mod 10 = 0 Then
          MsgBox s, 0, "Absatzteile"
          s = ""
        End If
      End If
    Loop
    MsgBox s, 0, "Absatzteile"
End Sub
```

14.3. Bilder

Das folgende Makro fügt ein grafisches Textcontent-Objekt als Link in das aktuelle Dokument ein, und zwar an der Cursorposition. Man muss zwar die Bildgröße spezifizieren, nicht jedoch die Position.

Listing 355. *Fügt ein Bild als Link am Anfang des Dokuments ein.*

```
Sub InsertGraphicObject(oDoc, sURL$)
  Dim oCursor     'Textcursor
  Dim oGraph      'Bildobjekt
  Dim oText       'Dokumentinhalt

  oText = oDoc.getText()
  oCursor = oText.createTextCursor()
  oCursor.gotoStart(False)    'Textcursor an den Anfang des Dokuments
  oGraph = oDoc.createInstance("com.sun.star.text.GraphicObject")

  With oGraph
    .GraphicURL = sURL
    'Verankert als Zeichen
    .AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
    .Width = 6000
    .Height = 8000
  End With

  'Nun wird das Bild ins Dokument eingefügt.
  oText.insertTextContent(oCursor, oGraph, False)
End Sub
```

Sie können auch ein Zeichnungsobjekt einfügen, das dann nicht an der aktuellen Cursorposition, sondern in der Folie eingefügt wird. Dazu benötigen Sie com.sun.star.drawing.GraphicObjectShape und müssen dann die Position und die Größe bestimmen.

Manchmal sind Sie gezwungen, die Bildgröße zu schätzen, weil sie nicht verfügbar ist. Die folgende Methode geht davon aus, dass das Argument ein Service vom Typ com.sun.star.graphic.GraphicDescriptor ist, der wahlweise die Bildgröße in 1/100 mm und in Pixel bereitstellt. Die Methode gibt einen Wert in 1/100 mm zurück, was für die Darstellungsinterna benötigt wird.

Integer-Werte sind wegen sehr hoher Werte in den Zwischenrechnungen nicht angebracht.

Wenn die Größe in 1/100 mm vorliegt, wird sie verwendet. Danach wird die Pixelgröße geprüft. Ein Bild hat sowohl eine Pixelgröße als auch eine erwartete Pixeldichte (DPI, Dots per Inch). Die Pixelanzahl mag wohl vorhanden sein, aber nicht die DPI. Ich nehme einfach an, dass die Pixeldichte so ist wie die der Bildschirmanzeige. Mit anderen Worten, wenn die die erwartete Größe nicht vorliegt, sollte man annehmen, dass die Grafik für die Anzeige auf dem aktuellen Bildschirm gedacht ist.

Listing 356. *Schätzung der Bildgröße.*

```
Function RecommendGraphSize(oGraph)
  Dim oSize
  Dim lMaxW As Double  'Maximale Breite in 1/100 mm
  Dim lMaxH As Double  'Maximale Höhe in 1/100 mm

  'lMaxW = 6.75 * 2540 ' 6,75 Zoll
  'lMaxH = 9.5 * 2540  ' 9,5 Zoll
  lMaxW = 17000 ' 17 cm
  lMaxH = 24000 ' 24 cm

  If IsNull(oGraph) Or IsEmpty(oGraph) Then
    Exit Function
  End If
  oSize = oGraph.Size100thMM
  If oSize.Height = 0 Or oSize.Width = 0 Then
    '2540 ist ein Zoll, umgerechnet in 1/100 mm.
    'Ein Zoll hat 14440 Twips.
    'Zur Erinnerung: TwipsPerPixelX und  TwipsPerPixelY sind Basic-Funktionen.
    oSize.Height = oGraph.SizePixel.Height * 2540.0 * TwipsPerPixelY() / 1440
    oSize.Width = oGraph.SizePixel.Width * 2540.0 * TwipsPerPixelX() / 1440
  End If
  If oSize.Height = 0 Or oSize.Width = 0 Then
    'oSize.Height = 2540
    'oSize.Width = 2540
    Exit Function
  End If
  'Falls nötig, wird die Größe proportional reduziert.
  If oSize.Width > lMaxW Then
    oSize.Height = oSize.Height * lMaxW / oSize.Width
    oSize.Width = lMaxW
  End If
  If oSize.Height > lMaxH Then
    oSize.Width = oSize.Width * lMaxH / oSize.Height
    oSize.Height = lMaxH
  End If
  RecommendGraphSize = oSize
End Function
```

Das Einbetten eines Bildes geschieht folgendermaßen:

1. Eine Form wird erzeugt und der Folie hinzugefügt.
2. Mit Hilfe des Service GraphicProvider wird die Bildbeschreibung vom externen Speicher geholt, bevor das Bild geladen wird.
3. Die Bildbeschreibung bildet die Grundlage für eine Schätzung der Bildgröße. Es ist nur eine Schätzung, denn wir wissen nur, was in der Beschreibung steht, die es aber nicht wirklich wissen muss.
4. Die Form wird mit dem vom GraphicProvider bereitgestellten Bild verbunden. An diesem Punkt wird das Bild geladen und ist nun OOo bekannt.

5. Der URL der Form wird auf den URL eines neu erstellten grafischen Objekts kopiert. Damit referenzieren die Grafik und die Form dasselbe Bild.
6. Die Grafik wird als Zeichen verankert und an der Cursorposition ins Dokument eingefügt.
7. Die Form wird nicht mehr gebraucht. Sie wird entfernt.

Aus mir unerklärlichen Gründen werden alle Bilder in winziger Größe eingefügt (weniger als 1 cm). Ich verwende daher die Bildgrößenschätzung, um die Grafikgröße festzulegen.

Listing 357. *Bettet ein Bild in ein Dokument ein.*

```
' oDoc - Dokument, das das Bild enthalten soll.
' oCurs - Cursor, wo das Bild eingefügt werden soll.
' sURL - URL des einzufügenden Bildes.
' sParStyle - Absatzvorlage für die Einfügestelle.
Sub EmbedGraphic(oDoc, oCurs, sURL$, sParStyle$)
  Dim oShape
  Dim oGraph     'Das Grafikobjekt ist Textcontent.
  Dim oProvider  'Der Service GraphicProvider.
  Dim oText

  oShape = oDoc.createInstance("com.sun.star.drawing.GraphicObjectShape")
  oGraph = oDoc.createInstance("com.sun.star.text.GraphicObject")

  oDoc.getDrawPage().add(oShape)

  oProvider = CreateUnoService("com.sun.star.graphic.GraphicProvider")

  Dim oProps(0) As New com.sun.star.beans.PropertyValue
  oProps(0).Name  = "URL"
  oProps(0).Value = sURL

  REM Sichert die Originalgröße.
  Dim oSize100thMM       'Breite und Höhe in 1/100 mm
  Dim lHeight As Long
  Dim lWidth As Long
  oSize100thMM = RecommendGraphSize(oProvider.queryGraphicDescriptor(oProps))
  If Not IsNull(oSize100thMM) And Not IsEmpty(oSize100thMM) Then
    lHeight = oSize100thMM.Height
    lWidth = oSize100thMM.Width
  End If

  oShape.Graphic = oProvider.queryGraphic(oProps())
  oGraph.Graphicurl = oShape.Graphicurl
  oGraph.AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
  oText= oCurs.getText()
  oText.insertTextContent(oCurs, oGraph, False)
  oDoc.getDrawPage().remove(oShape)

  If lHeight > 0 And lWidth > 0 Then
    Dim oSize
    'Zuerst Struct als Wert kopieren, dann ändern und schließlich zurück kopieren.
    oSize = oGraph.Size
    oSize.Height = lHeight
    oSize.Width = lWidth
    oGraph.Size = oSize
  End If
```

```
  ' Setzt das Absatzformat, wenn es im Dokument existiert.
  Dim oStyles
  oStyles = oDoc.StyleFamilies.getByName("ParagraphStyles")
  If oStyles.hasByName(sParStyle) Then
    oCurs.ParaStyleName = sParStyle
  End If
End Sub
```

14.4. HTML einfügen und verlinkte Grafiken einbetten

Nachdem man eine Webpage in die Zwischenablage kopiert und von dort in ein Textdokument eingefügt hat, sind alle Bilder eingebettet. Grafiken in einem Dokument haben die Eigenschaft GraphicURL. Der URL für eine Grafik, die im Dokument steckt, beginnt mit „vnd.sun.star.GraphicObject:“. Mit den Mitteln der API werden Grafiken als Links eingefügt – sie werden ohne zusätzliche Arbeit nicht im Dokument eingebettet.

Das folgende Makro kombiniert eine Reihe von Techniken, um alle verlinkten Bilder im Dokument zu finden und sie zu eingebetteten Bildern zu konvertieren.

Listing 358. *Konvertiert alle verlinkten Bilder zu eingebetteten Bildern.*

```
Sub ConvertAllLinkedGraphics(Optional aDoc)
  Dim oDoc         ' Arbeitsdokument
  Dim oDP          ' Folie
  Dim i%           ' Indexzähler
  Dim oGraph       ' Grafikobjekt in der Folie
  Dim iLinked%     ' Anzahl verlinkter Bilder
  Dim iEmbedded%   ' Anzahl eingebetteter Bilder
  Dim iConverted%  ' Anzahl verlinkter Bilder, die eingebettet wurden
  Dim s1$          ' Name des Service GraphicObjectShape
  Dim s2$          ' Name des Service TextGraphicObject

  s1 = "com.sun.star.drawing.GraphicObjectShape"
  s2 = "com.sun.star.text.TextGraphicObject"

  If IsMissing(aDoc) Or IsNull(aDoc) Or IsEmpty(aDoc) Then
    oDoc = ThisComponent
  Else
    oDoc = aDoc
  End If

  REM Listet die Bilder in der Folie über ihren Indexwert.
  oDP = oDoc.getDrawPage()
  For i = 0 To oDP.getCount() - 1
    oGraph = oDP.getByIndex(i)
    If oGraph.supportsService(s1) Or oGraph.supportsService(s2) Then
      If InStr(oGraph.GraphicURL, "vnd.sun") <> 0 Then
        iEmbedded = iEmbedded + 1
      Else
        iLinked = iLinked + 1
        If EmbedLinkedGraphic(oGraph, oDoc) Then  'Versuch der Einbettung
          iConverted = iConverted + 1
        End If
      End If
    End If
  Next
  Print iLinked & " verlinkte und " & iEmbedded & _
```

```
        " eingebettete Bilder gefunden und " & iConverted & " konvertiert."
End Sub

Function EmbedLinkedGraphic(oGraph, oDoc) As Boolean
  REM Autor: Andrew Pitonyak
  Dim sGraphURL$  ' Externer URL des Bildes.
  Dim oGraph_2    ' Neu erzeugtes Bild.
  Dim oCurs       ' Cursor, wo das Bild eingefügt wird.
  Dim oText       ' Textobjekt, das das Bild enthält.
  Dim oAnchor     ' Ankerpunkt des Bildes.
  Dim s1$         ' Name des Service GraphicObjectShape
  Dim s2$         ' Name des Service TextGraphicObject

  EmbedLinkedGraphic = False
  If InStr(oGraph.GraphicURL, "vnd.sun") <> 0 Then
    REM Ignoriert ein Bild, das schon eingebettet ist.
    Exit Function
  End If
  s1 = "com.sun.star.drawing.GraphicObjectShape"
  s2 = "com.sun.star.text.TextGraphicObject"
  If oGraph.supportsService(s1) Then
    REM Konvertiert ein GraphicObjectShape.
    'Kopiert die Verankerung.
    oAnchor = oGraph.getAnchor()
    oText = oAnchor.getText()
    'Erzeugt ein GraphicObjectShape und kopiert die Eigenschaften der verlinkten
    'Grafik auf die neue Grafik.
    oGraph_2 = ThisComponent.createInstance(s1)
    oGraph_2.GraphicObjectFillBitmap = oGraph.GraphicObjectFillBitmap
    oGraph_2.Size = oGraph.Size
    oGraph_2.Position = oGraph.Position
    'Fügt die neue Grafik an der Ankerposition ein und löscht die verlinkte Grafik.
    oText.insertTextContent(oAnchor, oGraph_2, False)
    oText.removeTextContent(oGraph)
    EmbedLinkedGraphic = True
  ElseIf oGraph.supportsService(s2) Then
    REM Konvertiert ein TextGraphicObject.
    Dim oBitmaps    'Liste aller TextGraphicObjects
    Dim sNewURL$    'Neuer URL
    Dim sName$      'Visuell lesbarer Name zur Anzeige im GUI

    sName$ = oGraph.LinkDisplayName
    oBitmaps = oDoc.createInstance("com.sun.star.drawing.BitmapTable")
    If oBitmaps.hasByName(sName) Then
      Print "Der Anzeigename des Links " & sName & " ist schon vorhanden."
      Exit Function
    End If

    'Fügt die Grafik ein und verlinkt sie intern.
    oBitmaps.insertByName(sName, oGraph.GraphicURL)
    sNewURL$ = oBitmaps.getByName(sName)
    oGraph.GraphicURL = sNewURL
    EmbedLinkedGraphic = True
  End If
End Function
```

14.5. Cursors

Die Möglichkeit, den gesamten Textinhalt zu enumerieren, wird man hauptsächlich für solche Aufgaben wie den Dokumentexport heranziehen, denn dabei müssen alle Inhalte in der Reihenfolge ihres Vorkommens angefasst werden. Weit häufiger dienen Cursors zur Manipulation eines Dokuments. Ein Textcursor ist ein Textrange, der innerhalb eines Textobjekts bewegt werden kann. Ein Textcursor kann also nicht nur auf einen einzelnen Punkt im Text bezogen sein, sondern auch auf eine Textspanne. Mit den Cursor-Bewegungsmethoden kann man einen Cursor neu positionieren und die Spanne des ausgewählten Textes erweitern. In OOo gibt es den Textcursor in zwei Arten: den sichtbaren Cursor (s. Tabelle 139) und den unsichtbaren Cursor (s. Tabelle 141).

14.5.1. Viewcursors

Wie der englische Name schon zeigt, ist der Viewcursor der sichtbare Cursor. In einem einzelnen Dokumentfenster kann man nur eine Darstellung gleichzeitig sehen. Analog dazu kann man auch nur einen Viewcursor gleichzeitig haben. Ein Viewcursor unterstützt Befehle, die sich direkt auf die Ansicht auswirken. Um den Cursor um je eine Zeile oder eine Bildschirmseite weiter zu bewegen, brauchen Sie einen Viewcursor. Der Viewcursor weiß, wie der Text dargestellt wird, s. Tabelle 140.

Tabelle 139. *Die Interfaces der verschiedenen Viewcursors.*

Cursor	Beschreibung
com.sun.star.view.XViewCursor	Einfacher Cursor mit grundlegenden Bewegungsmethoden, die sowohl im Text als auch in Tabellen funktionieren.
com.sun.star.text.XTextViewCursor	Von XTextCursor abgeleitet. Cursor in der Ansicht eines Textdokuments. Unterstützt nur sehr einfache Bewegungen.
com.sun.star.view.XLineCursor	Definiert zeilenbezogene Methoden. Ist nicht von einem Textrange abgeleitet.
com.sun.star.text.XPageCursor	Definiert seitenbezogene Methoden. Ist nicht von einem Textrange abgeleitet.
com.sun.star.view.XScreenCursor	Definiert Methoden, jeweils eine Bildschirmseite auf und ab zu scrollen.

Die meisten Cursor-Bewegungsmethoden erwarten ein boolesches Argument, das festlegt, ob der Textrange des Cursors erweitert wird oder nicht. Anders gesagt, wenn der boolesche Ausdruck True ist, wird eine Auswahl erzeugt, mit False wird der Cursor nur verschoben, ohne dass Text ausgewählt wird. In der Tabelle 140 wird die Beschreibung des booleschen Arguments bei den Bewegungsmethoden als nun bekannt vorausgesetzt. Eine weitere Gemeinsamkeit der Bewegungsmethoden ist die Zurückgabe eines booleschen Werts. True heißt, dass die Bewegung durchgeführt werden konnte, False heißt, dass es einen Fehler gab. Eine Bewegung schlägt fehl, wenn sie nicht vollständig beendet werden kann. Zum Beispiel kann man den Cursor nicht nach unten bewegen, wenn er am Ende des Dokuments steht. Der Screencursor wird vom aktuellen Controller des Dokuments bereitgestellt, s. Listing 359.

Tabelle 140. *Mit einem Viewcursor verbundene Methoden.*

Interface	Methode	Beschreibung
XViewCursor	goDown(n, Boolean)	Bewegt den Cursor n Zeilen nach unten.
XViewCursor	goUp(n, Boolean)	Bewegt den Cursor n Zeilen nach oben.
XViewCursor	goLeft(n, Boolean)	Bewegt den Cursor n Zeichen nach links.
XViewCursor	goRight(n, Boolean)	Bewegt den Cursor n Zeichen nach rechts.
XTextViewCursor	isVisible()	True, wenn der Cursor sichtbar ist.
XTextViewCursor	setVisible(Boolean)	Macht den Cursor sichtbar oder unsichtbar.
XTextViewCursor	getPosition()	Gibt ein Struct com.sun.star.awt.Point zurück mit den Koordinaten der Cursorposition bezogen auf die obere linke Ecke der ersten Seite des Dokuments.
XLineCursor	isAtStartOfLine()	True, wenn der Cursor am Zeilenanfang steht.

Interface	Methode	Beschreibung
XLineCursor	isAtEndOfLine()	True, wenn der Cursor am Zeilenende steht.
XLineCursor	gotoEndOfLine(Boolean)	Bewegt den Cursor an das Ende der aktuellen Zeile.
XLineCursor	gotoStartOfLine(Boolean)	Bewegt den Cursor an den Anfang der aktuellen Zeile.
XPageCursor	jumpToFirstPage()	Bewegt den Cursor auf die erste Seite.
XPageCursor	jumpToLastPage()	Bewegt den Cursor auf die letzte Seite.
XPageCursor	jumpToPage(n)	Bewegt den Cursor auf die Seite n.
XPageCursor	getPage()	Gibt die aktuelle Seitenzahl als Short Integer zurück.
XPageCursor	jumpToNextPage()	Bewegt den Cursor auf die folgende Seite.
XPageCursor	jumpToPreviousPage()	Bewegt den Cursor auf die vorherige Seite.
XPageCursor	jumpToEndOfPage()	Bewegt den Cursor an das Ende der aktuellen Seite.
XPageCursor	jumpToStartOfPage()	Bewegt den Cursor an den Anfang der aktuellen Seite.
XScreenCursor	screenDown()	Scrollt die Ansicht eine sichtbare Seite nach unten.
XScreenCursor	screenUp()	Scrollt die Ansicht eine sichtbare Seite nach oben.

Listing 359. *Scrollt eine Bildschirmseite nach unten.*

```
Sub ScrollDownOneScreen
  REM Der Viewcursor des aktuellen Controllers
  ThisComponent.CurrentController.getViewCursor().screenDown()
End Sub
```

Noch häufiger wird der Viewcursor dafür verwendet, irgendwelche Sonderzeichen an der aktuellen Cursorposition einzufügen. Das Makro im Listing 360 fügt das Zeichen mit dem Unicodewert 257 (ein „a“ mit einem Balken) an der aktuellen Cursorposition ein. Ein solches Makro wird gewöhnlich mit einer Tastenkombination verknüpft, um Sonderzeichen einzufügen, die nicht auf der Tastatur sind. Das Makro im Listing 360 ist kurz und einfach, aber sehr nützlich.

Listing 360. *Fügt das Zeichen mit dem Unicodewert 257 an der aktuellen Cursorposition ein.*

```
Sub InsertControlCharacterAtCurrentCursor
  Dim oViewCursor
  oViewCursor = ThisComponent.CurrentController.getViewCursor()
  oViewCursor.getText.insertString(oViewCursor.getStart(), Chr$(257), False)
End Sub
```

14.5.2. Textcursors (im Gegensatz zu Viewcursors)

Der Viewcursor weiß, wie die Daten angezeigt werden, weiß aber nichts über die Daten selbst. Textcursors wissen im Gegensatz dazu eine Menge über die Daten, aber nur sehr wenig über ihre Darstellung. Viewcursors kennen zum Beispiel keine Wörter oder Absätze, und Textcursors kennen keine Zeilen, Bildschirme oder Seiten, s. Tabelle 141.

Tabelle 141. *Alle Textcursor-Interfaces binden das Interface XTextCursor ein.*

Cursor	Beschreibung
com.sun.star.text.XTextCursor	Der grundlegende Textcursor. Definiert einfache Bewegungsmethoden.
com.sun.star.text.XWordCursor	Bietet Bewegungs- und Testmethoden in Bezug auf Wörter.
com.sun.star.text.XSentenceCursor	Bietet Bewegungs- und Testmethoden in Bezug auf Sätze.
com.sun.star.text.XParagraphCursor	Bietet Bewegungs- und Testmethoden in Bezug auf Absätze.
com.sun.star.text.XTextViewCursor	Abgeleitet von XTextCursor. Beschreibt einen Cursor in der Ansicht eines Textdokuments.

Tipp Textcursors und Viewcursors überschneiden sich ein wenig. XTextViewCursor ist von XTextCursor abgeleitet und unterstützt daher die XTextCursor-Methoden, allerdings nicht die Funktionalitäten, die sich auf die zugrunde liegenden Daten beziehen, wie wort- oder absatzbezogene Methoden, s. Tabelle 142.

Der Wortcursor, der Satzcursor und der Absatzcursor, alle definieren prinzipiell identische Objektmethoden, s. Tabelle 142. Das Interface XTextViewCursor wird in der Tabelle 140 vorgestellt und ist daher in der Tabelle 142 nicht enthalten.

Tabelle 142. *Auf Textcursors bezogene Objektmethoden.*

Interface	Methode	Beschreibung
XTextCursor	collapseToStart()	Setzt die Endposition auf die Startposition.
XTextCursor	collapseToEnd()	Setzt die Startposition auf die Endposition.
XTextCursor	isCollapsed()	True, wenn Start- und Endposition gleich sind.
XTextCursor	goLeft(n, Boolean)	Bewegt den Cursor um n Zeichen nach links.
XTextCursor	goRight(n, Boolean)	Bewegt den Cursor um n Zeichen nach rechts.
XTextCursor	gotoStart(Boolean)	Bewegt den Cursor an den Textanfang.
XTextCursor	gotoEnd(Boolean)	Bewegt den Cursor an das Textende.
XTextCursor	gotoRange(XTextRange, Boolean)	Bewegt oder expandiert den Cursor zum Textrange.
XWordCursor	isStartOfWord()	True, wenn der Cursor am Wortanfang steht.
XWordCursor	isEndOfWord()	True, wenn der Cursor am Wortende steht.
XWordCursor	gotoNextWord(Boolean)	Bewegt den Cursor zum Anfang des nächsten Worts.
XWordCursor	gotoPreviousWord(Boolean)	Bewegt den Cursor zum Ende des vorherigen Worts.
XWordCursor	gotoEndOfWord(Boolean)	Bewegt den Cursor zum Ende des aktuellen Worts.
XWordCursor	gotoStartOfWord(Boolean)	Bewegt den Cursor zum Anfang des aktuellen Worts.
XSentenceCursor	isStartOfSentence()	True, wenn der Cursor am Satzanfang steht.
XSentenceCursor	isEndOfSentence()	True, wenn der Cursor am Satzende steht.
XSentenceCursor	gotoNextSentence(Boolean)	Bewegt den Cursor zum Anfang des nächsten Satzes.
XSentenceCursor	gotoPreviousSentence(Boolean)	Bewegt den Cursor zum Anfang des vorherigen Satzes.
XSentenceCursor	gotoEndOfSentence(Boolean)	Bewegt den Cursor zum Ende des aktuellen Satzes.
XSentenceCursor	gotoStartOfSentence(Boolean)	Bewegt den Cursor zum Anfang des aktuellen Satzes.
XParagraphCursor	isStartOfParagraph()	True, wenn der Cursor am Absatzanfang steht.
XParagraphCursor	isEndOfParagraph()	True, wenn der Cursor am Absatzende steht.
XParagraphCursor	gotoNextParagraph(Boolean)	Bewegt den Cursor zum Anfang des nächsten Absatzes.
XParagraphCursor	gotoPreviousParagraph(Boolean)	Bewegt den Cursor zum Anfang des vorherigen Absatzes.
XParagraphCursor	gotoEndOfParagraph(Boolean)	Bewegt den Cursor zum Ende des aktuellen Absatzes.
XParagraphCursor	gotoStartOfParagraph(Boolean)	Bewegt den Cursor zum Anfang des aktuellen Absatzes.

14.5.3. Mit einem Cursor den Text durchlaufen

Es ist zwar nicht wirklich schwierig, mit einem Cursor den Text zu durchlaufen, doch ich tat mich lange Zeit damit schwer, bis ich schließlich erkannte, dass ich einem grundlegenden, aber einfachen Missverständnis aufgesessen war. Listing 361 zeigt eine gebräuchliche, aber subtil inkorrekte Methode, mit einem Cursor durch den Text zu wandern. Das Makro versucht, den Cursor von einem Absatz zum nächsten zu bewegen und dabei jeweils einen Absatz auszuwählen. Selbstverständlich wird irgend etwas mit dem Absatz getan, etwa eine andere Absatzvorlage gesetzt.

Listing 361. *Beispiel für falschen Gebrauch des Cursors: Dieser Code verpasst den letzten Absatz des Dokuments.*

```
Dim oCursor
REM Erzeugt einen Textcursor
oCursor = ThisComponent.Text.createTextCursor()
REM Start am Anfang des Dokuments.
REM Das ist dasselbe wie der Anfang des ersten Absatzes.
oCursor.gotoStart(False)
REM Und jetzt läuft es falsch!
REM Der Cursor erstreckt sich nun vom Anfang des ersten Absatzes
REM zum Anfang des zweiten Absatzes.
Do While oCursor.gotoNextParagraph(True)
  REM Der Absatz wird hier bearbeitet!
  REM Nun wird die Auswahl verworfen,
  REM und der Cursor steht am Anfang des nächsten Absatzes.
  oCursor.goRight(0, False)
Loop
```

Tipp Ich habe solch falschen Code wie im Listing 361 produziert, bevor ich Cursors wirklich verstand.

Das Problem mit Listing 361 ist, dass die Methode gotoNextParagraph(True) die Cursorauswahl vom Beginn eines Absatzes zum Beginn des nächsten Absatzes bewirkt. Der erste Fehler dabei ist, dass zwei Absätze ausgewählt sind. Wenn die beiden verschiedenen Absätze nicht dieselbe Absatzvorlage verwenden, wird die Eigenschaft ParaStyleName als Wert einen leeren String zurückgeben. Der zweite Fehler tritt auf, wenn der Cursor (wie im Listing 361 gezeigt) am Anfang des letzten Absatzes steht. Dann kann er nicht zum nächsten Absatz bewegt werden, denn den gibt es gar nicht. Die Anweisung „gotoNextParagraph(True)“ gibt False zurück, und der letzte Absatz wird nicht bearbeitet. Im Listing 362 wird eine korrekte Methode gezeigt, alle Absätze mit Hilfe eines Cursors zu durchlaufen.

Listing 362. *Der korrekte Weg, mit einem Cursor umzugehen.*

```
Dim oCursor
REM Erzeugt einen Textcursor
oCursor = ThisComponent.Text.createTextCursor()
REM Start am Anfang des Dokuments.
REM Das ist dasselbe wie der Anfang des ersten Absatzes.
oCursor.gotoStart(False)
Do
  REM Der Cursor steht schon am Anfang des aktuellen Absatzes,
  REM so dass nun der gesamte Absatz ausgewählt wird.
  oCursor.gotoEndOfParagraph(True)
  REM Der Absatz wird hier bearbeitet!
  REM Die Schleife bewegt den Cursor zum nächsten Absatz
  REM und verwirft gleichzeitig die Textauswahl.
Loop While oCursor.gotoNextParagraph(False)
```

Es geht darum, den Cursor über dem aktuellen Absatz auszurichten und dann über so ausgerichtete Absätze zu iterieren, statt sich schon auf den nächsten Absatz auszubreiten, wenn man noch den aktuellen bearbeitet.

Tipp Man kann mit Hilfe eines Cursors oder durch Enumeration ein Dokument durchlaufen, und so auf alle Absätze zugreifen. Mit einem Cursor ist es fünfmal schneller als mit einem Enumerator.

Es ist sehr einfach, einen Iterator zu schreiben, der den Text durchläuft und dabei Wörter, Sätze und Absätze zählt, s. Listing 363. Da in der LO-Version 6.4.5.2 – vielleicht auch in anderen Versionen – der Satz-Cursor fehlerhaft ist, ist die Zählung der Sätze im folgenden Makro abgeschaltet. Dennoch braucht es geraume Zeit bis zur Informationsmeldung.

Listing 363. *Zählt Absätze, Sätze und Wörter.*

```
Sub CountWordSentPar
  Dim oCursor             'Textcursor
  Dim nPars As Long       'Absatzzähler
  Dim nSentences As Long  'Satzzähler
  Dim nWords As Long      'Wortzähler

  REM Erzeugt einen Textcursor.
  oCursor = ThisComponent.Text.createTextCursor()

  'Zählt die Absätze vom Anfang des Dokuments an.
  oCursor.gotoStart(False)
  Do
    nPars = nPars + 1
  Loop While oCursor.gotoNextParagraph(False)

  'Zählt die Sätze vom Anfang des Dokuments an.
  'In LO 6.4.5.2, vielleicht auch in anderen Versionen gibt gotoNextSentence()
  'immer True zurück, auch wenn das Dokumentende schon überschritten ist,
  'mit dem Resultat einer Endlosschleife.

  'oCursor.gotoStart(False)
  'Do
  '  nSentences = nSentences + 1
  'Loop While oCursor.gotoNextSentence(False)

  'Zählt die Wörter vom Anfang des Dokuments an.
  oCursor.gotoStart(False)
  Do
    nWords = nWords + 1
  Loop While oCursor.gotoNextWord(False)

  MsgBox "Absätze: " & nPars & Chr$(10) &_
     "Sätze: " & nSentences & Chr$(10) &_
     "Wörter: " & nWords & Chr$(10), 0, "Dokumentstatistik"
End Sub
```

Den Viewcursor mit dem Textcursor synchronisieren

Ein Absatz wird gemäß den Seitenrandeinstellungen in Zeilen umbrochen. Eine Änderung der Seitenränder bringt veränderte Zeilenumbrüche. Und wenn man Text aus dem Writer kopiert, so enthält er keine Seitenumbrüche. Ein Beispiel aus dem wirklichen Leben (Listing 364) fügt richtige Zeilenumbrüche dort in den Text ein, wo die Zeilen auf dem Bildschirm umbrochen werden.

Man benötigt sowohl einen Viewcursor als auch einen Textcursor, um den Text zu durchlaufen. Der Viewcursor weiß, wo eine Zeile endet, weiß aber nicht, wo der Absatz anfängt oder endet. Ein Textcursor kennt die Absätze, der Viewcursor aber nicht.

Listing 364 zeigt, wie die Position des Viewcursors auf dem Bildschirm geändert wird.

Listing 364. *Fügt Zeilenumbrüche in einen Absatz ein.*

```
Sub LineBreaksInParagraph
  Dim oText       'Kürzer als immer ThisComponent.Text
  Dim oViewCursor 'Kürzer als immer ThisComponent.CurrentController.getViewCursor()
```

```
    Dim oTextCursor 'Neu erzeugter Textcursor
    Dim oSaveCursor 'Um den Viewcursor wiederherzustellen.

    oText = ThisComponent.Text

    REM Sie benötigen einen Viewcursor,
    REM weil nur der weiß, wo eine Zeile endet.
    oViewCursor = ThisComponent.CurrentController.getViewCursor()

    REM Sie benötigen einen Textcursor, um das Absatzende zu finden.
    REM Leider leider ist der Viewcursor kein Absatzcursor.
    oTextCursor = oText.createTextCursorByRange(oViewCursor)

    REM Das müssen Sie nur tun, wenn Sie den Viewcursor wiederherstellen wollen.
    oSaveCursor = oText.createTextCursorByRange(oViewCursor)

    REM Cursor auf den Anfang des aktuellen Absatzes bewegen,
    REM so dass der gesamte Absatz bearbeitet werden kann.
    If Not oTextCursor.isStartOfParagraph() Then
      oTextCursor.gotoStartOfParagraph(False)
      oViewCursor.gotoRange(oTextCursor, False)
    End If
    REM Nun gehen wir Zeile für Zeile vor.
    Do While True
      REM Nur der Viewcursor kennt das Zeilenende, denn das ist eine Frage
      REM der Dokumentformatierung und hat nichts mit Satzzeichen zu tun.
      oViewCursor.gotoEndOfLine(False)

      REM Der Textcursor geht mit dem Viewcursor an das Ende der aktuellen Zeile,
      REM um zu prüfen, ob er am Ende des Absatzes steht.
      oTextCursor.gotoRange(oViewCursor, False)

      REM Ein Zeilenumbruch wird nur eingefügt, wenn das Absatzende NICHT erreicht ist.
      If oTextCursor.isEndOfParagraph() Then Exit Do

      REM Fügt an der aktuellen Viewcursorposition einen Zeilenumbruch ein.
      oText.insertControlCharacter(oViewCursor,_
         com.sun.star.text.ControlCharacter.LINE_BREAK, False)
    Loop
    REM Wenn Sie nur den Cursor vom Ende des aktuellen Absatzes wegbewegen wollen,
    REM dann wird das reichen.
    oViewCursor.goRight(1, False)
    REM Ich würde aber lieber die Viewcursorposition wiederherstellen.
    REM oViewCursor.gotoRange(oSaveCursor, False)
End Sub
```

14.5.4. Mit Hilfe eines Cursors auf Inhalt zugreifen

Ich habe es mir zur Gewohnheit gemacht, die von OOo zurückgegebenen Objekte zu inspizieren. Bei der Inspizierung eines Viewcursors entdeckte ich einige nützliche, aber nicht dokumentierte Eigenschaften. Die Entwickler bei Sun versicherten mir, dass diese Eigenschaften in die Dokumentation einfließen werden. Ich bemerkte unter anderem folgende Eigenschaften: Cell, DocumentIndex, DocumentIndexMark, Endnote, Footnote, ReferenceMark, Text, TextField, TextFrame, TextSection und TextTable. Wenn der Cursor in einer Texttabelle steht, ist die Cursoreigenschaft TextTable nicht

leer. Wenn der Cursor auf einem Textfeld steht, ist die Cursoreigenschaft TextField nicht leer. Diese speziellen Eigenschaften sind leer, wenn sie irrelevant sind.

Ein Textfeld (s. Abschnitt Fehler: Verweis nicht gefunden14.10 Textfelder) als Typ TextField ist Textinhalt, der sich zwar in den umgebenden Text nahtlos einpasst, aber von einer anderen Quelle stammt. Die Quelle kann ein Feldbefehl sein, beispielsweise das Datum oder die aktuelle Seitenzahl. Es kann auch Text aus einem Querverweis oder aus einem Datenbankfeld sein. Ein solches Textfeld darf nicht verwechselt werden mit dem Textrahmen, der bei LO über das Menü **Einfügen > Textfeld** eingefügt werden kann.

Zum ersten Mal habe ich diese undokumentierten Eigenschaften genutzt, als ich darum gebeten wurde herauszufinden, ob der Viewcursor in einer Texttabelle steht, und wenn es so ist, in welcher Zelle (s. Listing 365).

Listing 365. *Test der Eigenschaften des Viewcursors.*

```
If Not IsEmpty(oViewCursor.TextTable) Then
```

Ein Cursor bietet die Möglichkeit, schnell Objekte zu finden, die sich in der Nähe von etwas anderem befinden. Zum Beispiel sollte ich kürzlich ein Textfeld im aktuellen Absatz finden – der aktuelle Absatz ist der, im der Viewcursor steht. Der Zugriff auf den Viewcursor ist einfach, und mit Hilfe eines Absatzcursors kann man sich im aktuellen Absatz bewegen.

In meinem ersten Versuch rief ich die Objektmethode createContentEnumeration(„com.sun.star.text.TextContent“) des Cursors auf. So erhält man eine Enumeration des Textcontent, das heißt solcher Objekte wie eingefügte Schaltflächen. Ich hatte fälschlicherweise angenommen, dass auf diese Weise auch Textfelder mit enumeriert werden. Mit meinem zweiten Versuch gelang es mir, ein Textfeld zu finden, nämlich mit der Objektmethode createEnumeration(). Diese Methode gibt eine Enumeration der Absätze unter dem Cursor zurück. Die Enumeration des Absatzinhalts bietet dann den Zugang zu dem Textfeld. Mein letzter, auch erfolgreicher Versuch setzt den Cursor an den Absatzanfang und bewegt ihn dann schrittweise durch den Absatz und testet die TextField-Eigenschaft an jeder Stelle. Das Makro im Listing 366 zeigt all diese Methoden, die ich zum Auffinden eines Textfeldes im aktuellen Absatz ausprobiert habe.

Listing 366. *Prüft, ob im aktuellen Absatz ein Textfeld ist.*

```
Sub TextFieldInCurrentParagraph
  Dim oEnum          'Cursor-Enumerator
  Dim oSection       'Aktelle Auswahl
  Dim oViewCursor    'Akteller Viewcursor
  Dim oTextCursor    'Neu erstellter Textcursor
  Dim oText          'Textobjekt im aktuellen Dokument
  Dim s$
  Dim sTextContent$ 'Servicename für Textcontent
  sTextContent = "com.sun.star.text.TextContent"

  oText = ThisComponent.Text
  oViewCursor = ThisComponent.CurrentController.getViewCursor()

  REM Der Viewcursor wird geholt, dann wird der Absatz ausgewählt,
  REM in dem der Viewcursor steht.
  oTextCursor = oText.createTextCursorByRange(oViewCursor)

  REM Bewegt den Textcursor an den Absatzanfang als Einzelpunkt.
  oTextCursor.gotoStartOfParagraph(False)
  REM Bewegt den Cursor an das Absatzende und erweitert dabei
  REM die Auswahl, so dass der gesamte Absatz ausgewählt ist.
  oTextCursor.gotoEndOfParagraph(True)

  REM Ich will den Textcontent enumerieren, der sich unter dem Cursor befindet.
```

```
    REM So finde ich zwar eingefügte Zeichenobjekte wie Schaltflächen,
    REM aber keine Textfelder!
    oEnum = oTextCursor.createContentEnumeration(sTextContent)
    Do While oEnum.hasMoreElements()
      oSection = oEnum.nextElement()
      Print "Enumeration des TextContent: " & oSection.ImplementationName
    Loop

    REM Folgendes enumeriert die Absätze unter dem Textcursor.
    oEnum = oTextCursor.createEnumeration()
    Dim v
    Do While oEnum.hasMoreElements()
      v = oEnum.nextElement()
      Dim oSubSection
      Dim oSecEnum

      REM Wir können die Absatzteile enumerieren, um
      REM Textfelder und anderen Absatzinhalt zu finden.
      oSecEnum = v.createEnumeration()
      s = "Enumeration des Typs: " & v.ImplementationName
      Do While oSecEnum.hasMoreElements()
          oSubSection = oSecEnum.nextElement()
          s = s & Chr$(10) & oSubSection.TextPortionType
          If oSubSection.TextPortionType = "TextField" Then
            s = s & " <== Typ " & oSubSection.TextField.ImplementationName
          End If
      Loop
      MsgBox s, 0, "Enumeration eines einzelnen Absatzes"
    Loop

    REM Noch ein anderer Weg, ein Textfeld zu finden.
    REM Man startet am Absatzanfang und bewegt den Cursor schrittweise weiter
    REM und testet die TextField-Eigenschaft.
    oTextCursor.gotoStartOfParagraph(False)
    Do While oTextCursor.goRight(1, False) And _
         Not oTextCursor.isEndOfParagraph()
      If Not IsEmpty(oTextCursor.TextField) Then
        Print "Die TextField-Eigenschaft ist nicht leer, das Textfeld ist benutzbar."
      End If
    Loop
End Sub
```

Achtung Es mag gar nicht intuitiv klingen, aber es ist nicht nur möglich, sondern kommt sogar häufig vor, dass in einem Textrange das Ende vor dem Anfang kommt. Die Reihenfolge der Start- und Endpositionen spielt hauptsächlich dann eine Rolle, wenn der Nutzer die Auswahl vorgenommen hat, aber auch, wenn der Textcursor nach einer Bewegung erweitert wird.

Wie die Objektmethoden getStart() und getEnd() implizieren, ist es möglich, dass ein Textrange einen einzelnen Punkt repräsentiert. Es ist auch möglich, dass die Startposition hinter der Endposition liegt. Im Umgang mit ausgewähltem Text sind unerwartete Start- und Endpositionen häufig ein Problem. Wenn Sie mit der Maus oder der Tastatur Text auswählen, ist der Ausgangspunkt der Auswahl im allgemeinen die Startposition. Wenn man den Punkt des Auswahlendes in Richtung Dokumentanfang schiebt, wird die Endposition des Textrange vor der Startposition liegen. Dasselbe Verhalten kann auftreten, wenn man einen Cursor manuell bewegt oder erweitert. Dieses Verhalten ist nicht dokumentiert, wurde aber seit OOo 1.1.0 beobachtet, und wie jedes nicht dokumentierte Verhalten mag

es sich irgendwann einmal ändern. Das Textobjekt kann zwei Textranges vergleichen (s. Tabelle 143), aber das Textobjekt muss beide Textranges auch enthalten – ein Textrange hat die Objektmethode getText(), mit der das Textobjekt zurückgegeben wird, das diesen Textrange enthält.

Tabelle 143. *Methoden im Interface com.sun.star.text.XTextRangeCompare.*

Methode	Beschreibung
compareRegionStarts(XTextRange, XTextRange)	• 1 = der erste Range startet vor dem zweiten. • 0 = der erste Range startet an derselben Stelle wie der zweite. • -1 = der erste Range startet nach dem zweiten.
compareRegionEnds(XTextRange, XTextRange)	• 1 = der erste Range endet vor dem zweiten. • 0 = der erste Range endet an derselben Stelle wie der zweite. • -1 = der erste Range endet nach dem zweiten.

14.6. Textauswahl

Eine Textauswahl ist Text, der vom Benutzer ausgewählt wurde, entweder über die Tastatur oder mit der Maus. Eine Textauswahl wird durch nichts anderes als einen Textrange repräsentiert. Wenn Sie eine Textauswahl gefunden haben, können Sie den reinen Text mit getString() holen oder mit setString() setzen. Ein String ist zwar auf eine Größe von 64 KB limitiert, eine Auswahl aber nicht. In manchen Fällen können Sie daher die Methoden getString() und setString() nicht verwenden. Es ist dann wahrscheinlich besser, mit einem Cursor den ausgewählten Text zu durchlaufen und Textinhalt über das Textobjekt des Cursors einzufügen. Im Umgang mit ausgewähltem Text sehen die meisten Aufgabenstellungen auf einer abstrakten Ebene gleich aus:

```
If (nichts ist ausgewählt) Then
  arbeite mit dem gesamten Dokument
Else
  For Each Auswahl
    arbeite mit dem ausgewähltem Bereich
```

Der schwierige Teil, der sich von Fall zu Fall ändert, ist das Schreiben eines Makros, das über eine Auswahl oder zwischen zwei Textranges iteriert.

14.6.1. Ist Text ausgewählt?

Textdokumente unterstützen das Interface XTextSectionsSupplier (s. Tabelle 132), das nur die eine Methode getCurrentSelection() definiert. Wenn es keinen aktuellen Controller gibt (was bedeutet, dass Sie als fortgeschrittener Benutzer OOo als Server ohne Benutzerschnittstelle verwenden und sowieso nicht nach ausgewähltem Text suchen), wird getCurrentSelection() NULL statt eines ausgewählten Textes zurückgeben.

Ist die Auswahlzählung null, ist nichts ausgewählt. Ich habe zwar noch nie eine Auswahlzählung gesehen, die null ist, aber es schadet nichts, den Test zu machen. Wenn kein Text ausgewählt ist, gibt es dennoch eine Auswahl der Länge null – Start- und Endposition sind dieselben. Sie können prüfen, ob die ausgewählte Stringlänge gleich null ist:

```
If Len(oSel.getString()) = 0 Then nichts ist ausgewählt
```

Sie können aber auch einen Textcursor aus dem ausgewählten Range erzeugen und dann prüfen, ob Start- und Endposition identisch sind.

```
oCursor = oDoc.Text.createTextCursorByRange(oSel)
If oCursor.isCollapsed() Then nichts ist ausgewählt
```

Die Makrofunktion im Listing 367 führt die ganze Überprüfung durch und gibt True zurück, wenn es eine Auswahl gibt, und False, wenn nichts ausgewählt wurde.

Listing 367. *Stellt fest, ob es eine Auswahl gibt.*

```
Function IsAnythingSelected(oDoc As Object) As Boolean
  Dim oSelections 'Enthält alle Selektionen
  Dim oSel        'Enthält eine bestimmte Selektion
  Dim oCursor     'Textcursor für den Test eines punktförmigen Range
  REM Grundannahme, dass es keine Selektion gibt.
  IsAnythingSelected = False
  If IsNull(oDoc) Then Exit Function
  'Der aktuelle Controller enthält die aktuelle Selektion.
  'Wenn es keinen Controller gibt, wird NULL zurückgegeben.
  oSelections = oDoc.getCurrentSelection()
  If IsNull(oSelections) Then Exit Function
  If oSelections.getCount() = 0 Then Exit Function
  If oSelections.getCount() > 1 Then
    REM Es gibt mehr als eine Selektion, also wird True zurückgegeben.
    IsAnythingSelected = True
  Else
    REM Es gibt nur eine Selektion, also wird sie geholt.
    oSel = oSelections.getByIndex(0)

    REM Erzeugt einen Textcursor, der den Range umfasst,
    REM um zu sehen, ob es ein Einzelpunkt ist.
    oCursor = oDoc.Text.createTextCursorByRange(oSel)
    If Not oCursor.isCollapsed() Then IsAnythingSelected = True

    REM Sie können auch die Start- und Endposition der Selektion
    REM vergleichen, um zu sehen, ob sie identisch sind.
    REM If oDoc.Text.compareRegionStarts(oSel.getStart(),_
    REM      oSel.getEnd()) <> 0 Then
    REM  IsAnythingSelected = True
    REM End If
  End If
End Function
```

Auf eine Auswahl zuzugreifen ist kompliziert, weil es möglich ist, mehrere unzusammenhängende Selektionen zu erstellen. Manche Selektionen sind leer, manche nicht. Wenn Sie Code zur Behandlung von Textselektionen schreiben, sollten all diese Fälle beachtet werden, weil sie häufig vorkommen. Das Beispiel im Listing 368 iteriert über alle ausgewählten Bereiche und gibt sie in einer Meldung aus. Zweierlei ist bemerkenswert. Bei einer Mehrfachauswahl ist die Auswahl mit dem Indexwert 0 immer ein Leerstring. Außerdem werden die Selektionen nach der Reihenfolge ihrer Auswahl und nicht nach der Aufeinanderfolge im Text gelistet.

Listing 368. *Gibt ausgewählten Text aus.*

```
Sub PrintMultipleTextSelection
  Dim oSelections             'Enthält alle Selektionen
  Dim oSel                    'Enthält eine bestimmte Selektion
  Dim iSelCount%              'Anzahl der Mehrfachselektionen
  Dim sMessage$               'Messagetext
  Dim lWhichSelection As Long 'Welche Selektion auszugeben ist

  If Not IsAnythingSelected(ThisComponent) Then
    Print "Es ist nichts ausgewählt"
  Else
    oSelections = ThisComponent.CurrentSelection
    iSelCount = oSelections.Count
```

```
    If iSelCount = 1 Then
      'Einfache Auswahl
      sMessage = "Einfache Auswahl: " & Chr(10) & _
                 oSelections.getByIndex(0).String
      MsgBox sMessage, 0, "Auswahl"
    Else
      'Bei einer Mehrfachauswahl ist die Auswahl mit dem Index 0
      'immer ein Dummywert mit einem Leerstring.
      sMessage = "Mehrfachauswahl:"
      For lWhichSelection = 1 To iSelCount - 1
        oSel = oSelections.getByIndex(lWhichSelection)
        sMessage = sMessage & Chr(10) & lWhichSelection & ": " & oSel.String
      Next
      MsgBox sMessage, 0, "Auswahl"
    End If
  End If
End Sub
```

14.6.2. Textauswahl: Welches Ende ist wo?

Selektionen sind Textranges mit sowohl einer Start- als auch einer Endposition. Welches Ende des Textes aber wo ist, wird von der Auswahlmethode bestimmt. Positionieren Sie zum Beispiel einmal den Cursor in der Mitte einer Zeile und selektieren Text dadurch, dass Sie den Cursor entweder nach rechts oder nach links verschieben. In beiden Fällen bleibt die Startposition dieselbe. Aber in einem der beiden Fälle wird die Startposition hinter der Endposition sein. Das Textobjekt bietet Methoden, die Start- und Endpositionen von Textranges zu vergleichen (s. Tabelle 143). Im Listing 369 verwende ich beide Methoden, um die linke und rechte Cursorposition der Textauswahl zu finden.

Listing 369. *Ermittelt die linke und rechte Cursorposition.*

```
'oSel ist eine Textauswahl oder ein Cursorrange.
'oText ist das Textobjekt
Function GetLeftMostCursor(oSel, oText)
  Dim oRange      'Textbereich
  Dim oCursor

  If oText.compareRegionStarts(oSel.getEnd(), oSel) >= 0 Then
    'Das Ende der Auswahl steht vor oder an derselben Stelle wie der Start.
    'Links ist also das Auswahlende.
    oRange = oSel.getEnd()
  Else
    '"Normale" Auswahl: Links ist also der Auswahlstart.
    oRange = oSel.getStart()
  End If
  oCursor = oText.createTextCursorByRange(oRange)
  oCursor.goRight(0, False)  'Gibt dem Cursor eine Laufrichtung.
  GetLeftMostCursor = oCursor
End Function

'oSel ist eine Textauswahl oder ein Cursorrange.
'oText ist das Textobjekt
Function GetRightMostCursor(oSel, oText)
  Dim oRange
  Dim oCursor

  If oText.compareRegionEnds(oSel.getStart(), oSel) >= 0 Then
    '"Normale" Auswahl. Der Start steht vor oder an derselben Stelle wie das Ende.
    'Rechts ist also das Auswahlende.
    oRange = oSel.getEnd()
```

```
    Else
      'Auswahl nach links. Rechts ist also der Auswahlstart.
      oRange = oSel.getStart()
    End If
    oCursor = oText.createTextCursorByRange(oRange)
    oCursor.goLeft(0, False)  'Gibt dem Cursor eine Laufrichtung.
    GetRightMostCursor = oCursor
  End Function
```

Beim Durchlaufen eines Dokuments mit einem Textcursor habe ich festgestellt, dass der Cursor sich die Laufrichtung merkt. Der von den Makros im Listing 369 jeweils zurückgegebene Cursor wird um null Zeichen nach links oder nach rechts bewegt. Dadurch erhält er eine Orientierung in Richtung der Textauswahl. Das ist auch ein Thema, wenn man einen Cursor nach rechts bewegt und ihn dann nach links umdreht. Ich bewege immer zuerst den Cursor um null Zeichen in die gewünschte Richtung, bevor ich ihn tatsächlich bewege. Dann kann mein Makro diesen Cursor nutzen, um den ausgewählten Text vom Start (nach rechts) oder vom Ende (nach links) zu durchlaufen.

14.6.3. Die Textauswahl-Rahmenstruktur

Im Umgang mit ausgewähltem Text nutze ich eine Rahmenstruktur, die ein zweidimensionales Array von Start- und Endcursors zur Iteration zurückgibt. Durch diese Rahmenstruktur kann ich den Code zur Iteration über den ausgewählten Text oder das gesamte Dokument minimieren. Wenn kein Text ausgewählt ist, fragt die Rahmenstruktur, ob das Makro das gesamte Dokument verwenden soll. Bei der Antwort Ja wird ein Cursor am Anfang und Ende des Dokuments erzeugt. Wenn Text ausgewählt ist, wird nacheinander für jede Selektion ein Cursor mit der entsprechenden Start- und Endposition erstellt, s. Listing 370.

Listing 370. *Erzeugt Cursors mit ausgewählten Bereichen.*

```
'sPrompt : Der Text der Frage, ob der gesamte Text durchlaufen werden soll.
'oCursors() : Erhält die Rückgabe-Cursors
'Gibt True zurück, wenn der gesamte Text durchlaufen werden soll, False, wenn nicht.
Function CreateSelectedTextIterator(oDoc, sPrompt$, oCursors()) As Boolean
  Dim oSelections             'Enthält alle Selektionen
  Dim oSel                    'Enthält eine bestimmte Selektion
  Dim oText                   'Textobjekt des Dokuments
  Dim lSelCount As Long       'Anzahl der Selektionen
  Dim lWhichSelection As Long 'Aktuelle Selektion
  Dim oLCursor, oRCursor      'Temporäre Cursors (links, rechts)
  CreateSelectedTextIterator = True
  oText = oDoc.Text
  If Not IsAnythingSelected(oDoc) Then
    Dim i%
    i% = MsgBox("Kein Text ausgewählt!" + Chr$(13) + sPrompt, _
      1 Or 32 Or 256, "Warnung")
    If i% = 1 Then
      'Schaltfläche "OK" gedrückt.
      oLCursor = oText.createTextCursorByRange(oText.getStart())
      oRCursor = oText.createTextCursorByRange(oText.getEnd())
      oCursors = DimArray(0, 1)  'Zweidimensionales Array mit einer Zeile
      oCursors(0, 0) = oLCursor
      oCursors(0, 1) = oRCursor
    Else
      'Schaltfläche "Abbrechen" gedrückt.
      oCursors = DimArray()      'Leeres Array als Rückgabe
      CreateSelectedTextIterator = False
```

```
    End If
  Else
    'Es wurde etwas ausgewählt.
    oSelections = oDoc.getCurrentSelection()
    lSelCount = oSelections.getCount()
    'Array zur Rückgabe mit so vielen Zeilen wie Selektionen.
    oCursors = DimArray(lSelCount - 1, 1)
    For lWhichSelection = 0 To lSelCount - 1
      'Für jede Auswahl je einen linken und einen rechten Cursor.
      oSel = oSelections.getByIndex(lWhichSelection)
      oLCursor = GetLeftMostCursor(oSel, oText)
      oRCursor = GetRightMostCursor(oSel, oText)
      oCursors(lWhichSelection, 0) = oLCursor
      oCursors(lWhichSelection, 1) = oRCursor
    Next
  End If
End Function
```

Tipp Das Argument oCursors() ist ein Array, das durch das Makro im Listing 370 gesetzt wird.

Das Makro im Listing 371 nutzt die Textauswahl-Rahmenstruktur, um die Unicodewerte der Textauswahl auszugeben.

Listing 371. *Gibt den Unicode der Textauswahl aus.*

```
Sub PrintUnicodeExamples
  Dim oCursors(), i%
  If Not CreateSelectedTextIterator(ThisComponent, _
    "Unicode für das gesamte Dokument ausgeben?", oCursors()) Then Exit Sub
  For i% = LBound(oCursors()) To UBound(oCursors())
    PrintUnicode_worker(oCursors(i%, 0), oCursors(i%, 1), ThisComponent.Text)
  Next i%
End Sub

Sub PrintUnicode_worker(oLCursor, oRCursor, oText)
  Dim s As String   'Ausgabestring
  Dim ss As String  'Temporärer String

  If IsNull(oLCursor) Or IsNull(oRCursor) Or IsNull(oText) Then Exit Sub
  If oText.compareRegionEnds(oLCursor, oRCursor) <= 0 Then Exit Sub

  REM Richtet den Cursor ohne Textselektion richtig aus.
  oLCursor.goRight(0, False)
  Do While oLCursor.goRight(1, True)_
     And oText.compareRegionEnds(oLCursor, oRCursor) >= 0
    ss = oLCursor.getString()
    REM Der String kann leer sein.
    If Len(ss) > 0 Then
      s = s & oLCursor.getString() & "=" & Asc(oLCursor.getString()) & " "
    End If
    oLCursor.goRight(0, False)
  Loop
  MsgBox s, 0, "Unicodewerte"
End Sub
```

14.6.4. Leerzeichen und Leerzeilen entfernen: ein größeres Beispiel

Häufig wird nach einem Makro gefragt, das überzählige Leerzeichen entfernt. Um alle leeren Absätze zu entfernen, nimmt man am besten die Option „Leere Absätze entfernen“ (**Extras | Autokorrek-**

tur-Optionen | Optionen). Zum Entfernen ausgewählter Absätze oder Folgen von Leerzeichen brauchen Sie ein Makro.

Dieser Abschnitt präsentiert eine Makrogruppe, die alle Folgen von weißen Zeichen durch ein einzelnes weißes Zeichen ersetzt. Sie können dieses Makro leicht modifizieren, um verschiedene Arten von weißen Zeichen zu entfernen. Die unterschiedlichen Arten von Leerraum sind nach ihrer Wichtigkeit geordnet. Wenn also auf ein einfaches Leerzeichen ein neuer Absatz folgt, bleibt der Absatz erhalten und das Leerzeichen wird gelöscht. Im Endeffekt werden alle weißen Zeichen vom Anfang und Ende einer Zeile entfernt.

Was sind weiße Zeichen?

Der Begriff „weißes Zeichen“ bezeichnet alle Zeichen, die als leerer Raum dargestellt werden. Dazu gehören Tabulatoren (ASCII 9), normale Leerzeichen (ASCII 32), geschützte Leerzeichen (ASCII 160), Absatzzeichen (ASCII 13) und Zeilenumbrüche (ASCII 10). Wenn man die Definition der weißen Zeichen in einer Funktion kapselt (s. Listing 372), kann man diese Definition auf einfachste Weise ändern, um bestimmte Zeichen auszuschließen.

Listing 372. *Definition der weißen Zeichen.*

```
Function IsWhiteSpace(iChar As Integer) As Boolean
  Select Case iChar
    Case 9, 10, 13, 32, 160
      IsWhiteSpace = True
    Case Else
      IsWhiteSpace = False
  End Select
End Function
```

Sollte der Zeitfaktor eine wichtige Rolle spielen, so ist der Zugriff schneller, wenn man ein Array mit dem Unicodewert als Index erstellt, mit den Werten True oder False als Indikatoren, ob ein Zeichen weiß ist oder nicht. Es ist nur etwas umständlicher, weil das Array vor dem Aufruf zu erstellen ist.

```
Dim isWhiteArray(0 To 161) As Boolean
  For i = LBound(isWhiteArray) To UBound(isWhiteArray)
    isWhiteArray(i) = False
  Next
  isWhiteArray(9) = True
  isWhiteArray(10) = True
  isWhiteArray(13) = True
  isWhiteArray(32) = True
  isWhiteArray(160) = True
```

Die Funktion zum Prüfen eines Zeichens sieht so aus:

```
Function IsWhiteSpace2(iChar As Integer) As Boolean
  If iChar > UBound(isWhiteArray) Then
    IsWhiteSpace2 = False
  Else
    IsWhiteSpace2 = isWhiteArray(iChar)
  End If
End Function
```

Rangfolge der Zeichen für die Löschentscheidung

In einer Folge von weißen Zeichen wird jedes Zeichen mit dem vorherigen verglichen. Sind beide Zeichen weiß, wird das weniger wichtige Zeichen gelöscht. Wenn zum Beispiel ein Leerzeichen und ein neuer Absatz nebeneinander stehen, wird das Leerzeichen gelöscht. Der Funktion RankChar() (s.

Listing 373) werden zwei Zeichen übergeben: das vorhergehende und das aktuelle Zeichen. Der zurückgegebene Integer-Wert zeigt an, ob ein Zeichen oder welches Zeichen gelöscht werden soll.

Listing 373. *Wertet Zeichen für die Löschung aus.*

```
'-1 heißt, dass das vorhergehende Zeichen zu löschen ist.
' 0 heißt, dass dieses Zeichen zu ignorieren ist.
' 1 heißt, dass dieses Zeichen zu löschen ist.
' Das Eingabezeichen 0 bedeutet den Anfang einer Zeile.
' Rangfolge von oben nach unten: 0, 13, 10, 9, 160, 32
Function RankChar(iPrevChar, iCurChar) As Integer
  If Not IsWhiteSpace(iCurChar) Then  'Kein weißes Zeichen. Ignorieren.
    RankChar = 0
  ElseIf iPrevChar = 0 Then           'Zeilenanfang, das aktuelle Zeichen ist weiß.
    RankChar = 1                      '      Das aktuelle Zeichen löschen.
  ElseIf Not IsWhiteSpace(iPrevChar) Then
                          'Das aktuelle Zeichen ist weiß, aber nicht das vorhergehende.
    RankChar = 0                      '      Ignorieren.

  REM An diesem Punkt sind beide Zeichen weiß.
  ElseIf iPrevChar = 13 Then          'Das vorhergehende Zeichen hat den höchsten Rang.
    RankChar = 1                      '      Das aktuelle Zeichen löschen.
  ElseIf iCurChar = 13 Then           'Das aktuelle Zeichen hat den höchsten Rang.
    RankChar = -1                     '      Das vorhergehende Zeichen löschen.

  REM Keins der Zeichen ist ein neuer Absatz, das höchstrangige Zeichen.
  ElseIf iPrevChar = 10 Then          'Das vorhergehende Zeichen ist ein Zeilenumbruch.
    RankChar = 1                      '      Das aktuelle Zeichen löschen.
  ElseIf iCurChar = 10 Then           'Das aktuelle Zeichen ist ein Zeilenumbruch.
    RankChar = -1                     '      Das vorhergehende Zeichen löschen.

  REM An diesem Punkt ist ein Tabulator das Zeichen mit dem höchstmöglichen Rang.
  ElseIf iPrevChar = 9 Then           'Das vorhergehende Zeichen ist ein Tabulator.
    RankChar = 1                      '      Das aktuelle Zeichen löschen.
  ElseIf iCurChar = 9 Then            'Das aktuelle Zeichen ist ein Tabulator.
    RankChar = -1                     '      Das vorhergehende Zeichen löschen.
  ElseIf iPrevChar = 160 Then
                           'Das vorhergehende Zeichen ist ein geschütztes Leerzeichen.
    RankChar = 1                      '      Das aktuelle Zeichen löschen.
  ElseIf iCurChar = 160 Then
                           'Das aktuelle Zeichen ist ein geschütztes Leerzeichen.
    RankChar = -1                     '      Das vorhergehende Zeichen löschen.
  ElseIf iPrevChar = 32 Then
                           'Das vorhergehende Zeichen ist ein normales Leerzeichen.
    RankChar = 1                      '      Das aktuelle Zeichen löschen.

  REM Man sollte wahrscheinlich nie hierher kommen... Beide Zeichen sind weiß
  REM und das vorhergehende Zeichen ist ein unbekanntes weißes Zeichen.
  ElseIf iCurChar = 32 Then
                           'Das aktuelle Zeichen ist ein normales Leerzeichen.
    RankChar = -1                     '      Das vorhergehende Zeichen löschen.
  Else                                'Dazu sollte es wohl nie kommen.
    RankChar = 0                      '      Also einfach ignorieren!
  End If
End Function
```

Wie man die Standard-Rahmenstruktur nutzt

Mit Hilfe der Standard-Textauswahl-Rahmenstruktur werden überzählige weiße Zeichen entfernt. Die Startroutine ist so einfach, dass sie kaum der Erwähnung wert ist.

Listing 374. *Entfernt weiße Zeichen.*

```
Sub RemoveEmptySpace
  Dim oCursors(), i%
  If Not CreateSelectedTextIterator(ThisComponent, _
    "ALLE weißen Zeichen des GESAMTEN Dokuments entfernen?", oCursors()) Then Exit Sub
  For i% = LBound(oCursors()) To UBound(oCursors())
     RemoveEmptySpaceWorker(oCursors(i%, 0), oCursors(i%, 1), ThisComponent.Text)
  Next i%
End Sub
```

Das Arbeitsmakro

Das Makro im Listing 375 ist der interessanteste Teil der Aufgabe. Es entscheidet, was gelöscht wird und was stehenbleibt. Zu beachten sind folgende Punkte:

- Weil ein Textcursor benutzt wird, wird die Formatierung nicht geändert.
- Ein Textrange (Cursor) kann einen Textcontent enthalten, der einen String der Länge Null zurückgibt. Das betrifft zum Beispiel in das Dokument eingefügte Schaltflächen und Grafiken. Der Umgang mit Ausnahmen macht das Makro komplizierter. Vieles wäre sehr einfach, wenn man solche Ausnahmen wie zum Beispiel eingefügte Grafiken ignorierte. Wenn Sie ganz genau wissen, dass Ihr Makro nur einfache kontrollierte Daten enthält, könnten Sie auf robustes Verhalten verzichten, um die Komplexität zu reduzieren. Listing 375 bietet die sichere Variante mit Behandlung von Ausnahmen.
- Wenn die Textauswahl mit einem weißen Zeichen startet oder endet, wird es entfernt, auch wenn es sich nicht um den Start oder das Ende des Dokuments handelt.

Listing 375. *Entfernung weißer Zeichen im Einsatz.*

```
Sub RemoveEmptySpaceWorker(oLCursor, oRCursor, oText)
  Dim s As String           'Temporärer String
  Dim i As Integer          'Temporärer Integer zum Vergleich von Textranges
  Dim iLastChar As Integer  'Unicode des vorhergehenden Zeichens
  Dim iThisChar As Integer  'Unicode des aktuellen Zeichens
  Dim iRank As Integer      'Integer für die Löschentscheidung

  REM Wenn etwas NULL ist, wird nichts getan.
  If IsNull(oLCursor) Or IsNull(oRCursor) Or IsNull(oText) Then Exit Sub

  REM Ranges, die einen Punkt repräsentieren, werden ignoriert.
  If oText.compareRegionEnds(oLCursor, oRCursor) <= 0 Then Exit Sub

  REM Zeilenanfang als Standardwert für das erste und letzte Zeichen.
  iLastChar = 0
  iThisChar = 0

  REM Start mit dem Cursor ganz links in Richtung Dokumentende.
  REM Es wird nichts ausgewählt.
  oLCursor.goRight(0, False)

  REM Am Dokumentende kann der Cursor nicht weiter nach rechts bewegt werden.
  Do While oLCursor.goRight(1, True)

    REM Es ist möglich, dass der String die Länge null hat.
    REM Das kann vorkommen, wenn man auf im Text verankerte Objekte stößt,
    REM die keinen Text enthalten. Darauf muss besonders geachtet werden,
```

```
      REM weil diese Routine solche Objekte löschen könnte.
      REM Denn der Cursor findet sie, aber sie haben die Länge null.
      REM Ich verleihe ihnen einen normalen ASCII-Wert, ohne den String.
      s = oLCursor.getString()
      If Len(s) = 0 Then
        oLCursor.goRight(0, False)
        iThisChar = 65
      Else
        iThisChar = Asc(oLCursor.getString())
      End If

      REM Ein weißes Zeichen am Ende der Auswahl wird immer gelöscht.
      i = oText.compareRegionEnds(oLCursor, oRCursor)
      If i = 0 Then
        If IsWhiteSpace(iThisChar) Then oLCursor.setString("")
        Exit Do
      End If

      REM Wenn der Schritt über das Ende der Auswahl geht, wird die Schleife beendet.
      If i < 0 Then Exit Do

      iRank = RankChar(iLastChar, iThisChar)
      If iRank = 1 Then
        REM Das aktelle Zeichen wird gelöscht.
        REM iLastChar bleibt unverändert.
        REM Durch die Ersetzung des aktuellen Zeichens mit einem leeren String
        REM ist kein Text mehr ausgewählt.
        oLCursor.setString("")
      ElseIf iRank = -1 Then
        REM Das vorhergehende Zeichen wird gelöscht. Ein Zeichen wurde schon bei der
        REM Cursorbewegung nach rechts ausgewählt. Der Cursor wird nun zwei Zeichen
        REM nach links bewegt. Dadurch wird das aktuelle Zeichen deselektiert
        REM und das Zeichen links ausgewählt.
        oLCursor.goLeft(2, True)
        oLCursor.setString("")
        REM Nun wird der Cursor wieder über das aktuelle Zeichen hinweg bewegt,
        REM aber ohne es auszuwählen.
        oLCursor.goRight(1, False)
        REM Das aktelle Zeichen wird zum vorhergehenden Zeichen.
        iLastChar = iThisChar
      Else
        REM Das aktuelle Zeichen wird ignoriert. Jeglicher Text wird deselektiert
        REM und das aktelle Zeichen wird zum vorhergehenden Zeichen.
        oLCursor.goRight(0, False)
        iLastChar = iThisChar
      End If
    Loop
  End Sub
```

14.6.5. Textauswahl, abschließende Gedanken

Jeder, der sich mit Algorithmen beschäftigt hat, wird bestätigen, dass ein besserer Algorithmus fast immer besser ist als ein schnellerer Rechner. Schon früh hatte ich das Problem der Wortzählung in einer Textauswahl gelöst. Ich fand drei Wege mit unterschiedlichem Erfolg.

Meine erste Lösung konvertierte den Auswahltext in Basic-Strings und bearbeitete sie. Das ging sehr schnell: 8000 Wörter in 2,7 Sekunden. Diese Lösung hatte aber ihre Grenze, wenn Strings länger als 64 KB wurden, war also ungeeignet für große Dokumente.

Meine zweite Lösung war ein Cursor, der den gesamten Text Zeichen für Zeichen durchlief. Das war für jede Textlänge geeignet, brauchte aber für dieselben 8000 Wörter geschlagene 47 Sekunden, war also für den Benutzer unerträglich langsam.

Schließlich verwendete ich einen Wortcursor, der die Wörter in 1,7 Sekunden zählte. Schade nur, dass der Wortcursor nicht immer zuverlässig ist.

Achtung Mittlerweile bietet OOo eine bequeme und vor allem superschnelle Wortzählung über das Menü an. Falls Sie dennoch ein Makro für eine eigene Wortzählung planen sollten – vielleicht weil Sie nicht wollen, dass Nummerierungen oder Bullets mitgezählt werden –, beachten Sie, dass der Wortcursor nur für das Textobjekt existiert, von dem er erzeugt wurde. Nicht nur das gesamte Dokument, sondern auch jede Zelle einer Tabelle hat ein eigenes Textobjekt, ebenso wie jeder Textrahmen. Je nachdem, ob Ihr Dokument Tabellen und/oder Textrahmen enthält, müssen Sie diese mit berücksichtigen.

14.7. Text suchen

Der Suchprozess wird von einem Suchdeskriptor gesteuert, der nur das Objekt durchsuchen kann, das ihn erzeugt hat. Mit anderen Worten, Sie können nicht mit ein und demselben Suchdeskriptor mehrere Dokumente durchsuchen. Der Suchdeskriptor spezifiziert den Suchtext und die Art der Textsuche, s. Tabelle 144. Er ist der komplizierteste Teil der Suche.

Tabelle 144. *Eigenschaften im Service com.sun.star.util.SearchDescriptor.*

Eigenschaft	Beschreibung
SearchBackwards	Falls True, wird das Dokument rückwärts durchsucht.
SearchCaseSensitive	Falls True, wird die Groß- und Kleinschreibung beachtet.
SearchWords	Falls True, werden nur ganze Wörter gefunden.
SearchRegularExpression	Falls True, wird der Suchstring als regulärer Ausdruck behandelt.
SearchStyles	Falls True, wird Text gefunden, der die im Suchstring genannte Vorlage verwendet.
SearchSimilarity	Falls True, wird eine Ähnlichkeitssuche durchgeführt.
SearchSimilarityRelax	Falls True, werden die Eigenschaften SearchSimilarityRemove, SearchSimilarityAdd und SearchSimilarityExchange kombiniert genutzt.
SearchSimilarityRemove	Short Integer. Anzahl der Zeichen, die der Treffer kürzer sein darf als der Suchstring.
SearchSimilarityAdd	Short Integer. Anzahl der Zeichen, die der Treffer länger sein darf als der Suchstring.
SearchSimilarityExchange	Short Integer. Anzahl der Zeichen, die im Treffer anders sein dürfen als im Suchstring.

Obwohl nicht in der Tabelle 144 aufgeführt, unterstützt ein Suchdeskriptor die String-Eigenschaft SearchString, die den zu suchenden Text enthält. Das Interface XSearchDescriptor definiert die Methoden getSearchString() und setSearchString() zum Auslesen oder Setzen der Eigenschaft, wenn Sie es vorziehen, dazu eine Methode zu verwenden statt sie direkt zu verwenden. Das Interface XSearchable definiert die Methoden zum Suchen und zum Erzeugen des Suchdeskriptors, s. Tabelle 145.

Tabelle 145. *Methoden im Interface com.sun.star.util.XSearchable.*

Methode	Beschreibung
createSearchDescriptor()	Erzeugt einen neuen Suchdeskriptor.
findAll(XSearchDescriptor)	Gibt einen XIndexAccess zurück, der alle Treffer enthält.
findFirst(XSearchDescriptor)	Gibt vom Start des durchsuchten Objekts an einen Textrange zurück, der den ersten Treffer enthält.
findNext(XTextRange, XSearchDescriptor)	Gibt vom Start des angegebenen Textrange an einen Textrange zurück, der den ersten Treffer enthält.

Das Makro im Listing 376 ist sehr simpel. Es setzt die Zeicheneigenschaft CharWeight in allen Vorkommen des Textes „hallo“ auf fett. Das ist die Konstante com.sun.start.awt.FontWeight.BOLD. Ein Textrange unterstützt Zeichen- und Absatzeigenschaften.

Listing 376. *Setzt alle Vorkommen des Wortes „hallo“ in Fettschrift.*

```
Sub SetHelloToBold
  Dim oDescriptor  'Der Suchdeskriptor
  Dim oFound       'Der Trefferrange

  oDescriptor = ThisComponent.createSearchDescriptor()
  With oDescriptor
    .SearchString = "hallo"
    .SearchWords = True           'Der Standardwert der Eigenschaften ist False.
    .SearchCaseSensitive = False  'Sie auf False zu setzen ist also redundant.
  End With

  ' Der erste Treffer
  oFound = ThisComponent.findFirst(oDescriptor)
  Do While Not IsNull(oFound)
    Print oFound.getString()
    oFound.CharWeight = com.sun.star.awt.FontWeight.BOLD
    oFound = ThisComponent.findNext(oFound.End, oDescriptor)
  Loop
End Sub
```

14.7.1. Eine Textauswahl oder einen bestimmten Range durchsuchen

Der Trick, einen bestimmten Textrange zu durchsuchen, liegt in der Erkenntnis, dass man in der findNext-Methode jeden Textrange, einschließlich Textcursor, verwenden kann. Nach jedem findNext()-Aufruf überprüfen wir die Endposition, um herauszufinden, ob wir in der Suche zu weit gegangen sind. Man kann daher eine Suche auf irgendeinen Textrange beschränken. Der Hauptzweck der Methode findFirst ist, für findNext den Start-Textrange zu finden. Mit der Unterstützung der Textauswahl-Rahmenstruktur können Sie ganz leicht einen Textrange durchsuchen.

Listing 377. *Durchläuft alle Texttreffer zwischen zwei Cursors.*

```
Sub SearchSelectedWorker(oLCursor, oRCursor, oText, oDescriptor)
  If oText.compareRegionEnds(oLCursor, oRCursor) <= 0 Then
    'Kein Text ausgewählt.
    Exit Sub
  End If
  oLCursor.goRight(0, False)   'Gibt dem Cursor die Laufrichtung.
  Dim oFound
  REM Es gibt gar keinen Grund für findFirst.
  oFound = oDoc.findNext(oLCursor, oDescriptor)
  Do While Not IsNull(oFound)
    REM Sind wir über das Ende hinaus?
    If -1 = oText.compareRegionEnds(oFound, oRCursor) Then Exit Do
    Print oFound.getString()
    oFound = ThisComponent.findNext(oFound.End, oDescriptor)
  Loop
End Sub
```

Das Textobjekt kann zwei Abschnitte nur dann vergleichen, wenn sie beide zu eben diesem Textobjekt gehören. Text, der in einem anderen Rahmen, Bereich oder sogar Texttabelle steckt, verwendet ein anderes Textobjekt als das des Hauptdokuments. Untersuchen Sie einmal zur Übung, was geschieht, wenn der gefundene Text in einem anderen Textobjekt ist als in dem Textobjekt, das den oRCursor im Listing 377 enthält. Ist der Code des Listing 377 robust?

Suche nach allen Treffern

Die Suche nach allen Treffern auf einmal mit der findAll()-Objektmethode ist erheblich schneller als der wiederholte Aufruf von findNext(). Man sollte jedoch vorsichtig sein, wenn man alle Treffer des Suchtextes nutzt. Das Makro im Listing 378 ist ein extremes Beispiel dafür, dass ein Code mit Absicht schiefgehen kann.

Listing 378. *Sucht und ersetzt jedes Vorkommen des Worts "halloxyzzy".*

```
Sub SimpleSearchHalloXyzzy
  Dim oDescriptor  'Der Suchdeskriptor
  Dim oFound       'Der Trefferrange
  Dim oFoundAll    'Liste aller Trefferranges
  Dim n%           'Indexvariable
  oDescriptor = ThisComponent.createSearchDescriptor()
  oDescriptor.SearchString = "halloxyzzy"
  oFoundAll = ThisComponent.findAll(oDescriptor)
  For n% = 0 To oFoundAll.getCount()-1
    oFound = oFoundAll.getByIndex(n%)
    'Print oFound.getString()
    oFound.setString("hallo" & n%)
  Next
End Sub
```

Das Makro im Listing 378 erstellt eine Liste der Textranges, die den Text „halloxyzzy" einschließen. Dieser Text wird dann durch einen kürzeren Textstring ersetzt. In einer perfekten Welt würden die Stellen mit „halloxyzzy" durch „hallo0", „hallo1", „hallo2", ... ersetzt. Jedesmal, wenn der Text durch den kürzeren Text ausgetauscht wird, ändert sich die Gesamtlänge des Dokuments. Denken Sie daran, dass die Textrange-Objekte alle erstellt wurden, bevor der erste Text geändert wurde. Obwohl das Textrange-Interface klar definiert ist, die internen Abläufe sind es nicht. Ich habe dieses Beispiel gezielt mit der Erwartung geschrieben, dass es Fehler erzeugt, und weil das resultierende Verhalten nicht definiert ist, war dieser Test meine einzige Option, es herauszufinden. Im Experiment beobachtete ich, dass wenn „halloxyzzy" mehrfach im selben Wort vorkommt, die Resultate fehlerhaft sind. Ich beobachtete aber auch, dass wenn alle Vorkommen von „halloxyzzy" in verschiedenen Wörtern sind, alles großartig funktioniert: es werden nur die „halloxyzzy"-Strings ersetzt, der restliche Text bleibt intakt. Ich kann nur den Hut ziehen in Würdigung der brillanten Leistung der Programmierer, die dieses Verhalten ermöglicht haben. Dabei bleibe ich aber vorsichtig paranoid und argwöhne, dass auf dieses Verhalten vertrauender Code irgendwann in der Zukunft versagen könnte.

14.7.2. Suchen und ersetzen

Man kann suchen und ersetzen auf die Weise, dass man sucht und dann manuell jede Fundstelle durch den neuen Text ersetzt. OOo definiert aber auch das Interface XReplaceable, das die Möglichkeit bietet, alle Fundstellen mit einer Objektmethode auf einmal zu ersetzen. Anstelle des XSearchDescriptor benötigen Sie jedoch den XReplaceDescriptor. Alle Fundstellen auf einmal zu ersetzen, ist sehr einfach, s. Listing 379.

Tipp	Das Interface XReplaceable ist vom Interface XSearchable abgeleitet, und das Interface XReplaceDescriptor ist vom Interface XSearchDescriptor abgeleitet.

Listing 379. *Ersetzt „hallo du" durch „hallo ich".*

```
oDescriptor = oDoc.createReplaceDescriptor()
With oDescriptor
  .SearchString = "hallo du"
  .ReplaceString = "hallo ich"
```

```
End With
oDoc.ReplaceAll(oDescriptor)
```

14.7.3. Erweitertes Suchen und Ersetzen

Im OOo-GUI-Dialog Suchen & Ersetzen ist es möglich, zu dem Text auch Attribute als Suchelement anzugeben. Eine Inspizierung der Suchen-und-Ersetzen-Deskriptoren fördert die Objektmethoden setSearchAttributes() und setReplaceAttributes() zutage. Ich entdeckte, wie man diese Methoden anwendet, als ich Code fand, der von dem mir unbekannten Alex Savitsky und von dem mir bekannten Laurent Godard stammte.

Das Makro im Listing 380 sucht jeden fett formatierten Text, konvertiert ihn ins Standardformat und umrahmt den Text mit doppelten geschwungenen Klammern. Die Konvertierung von Attributen in Textauszeichnungen wird häufig vorgenommen, wenn formatierter Text in einfachen ASCII-Text ohne Formatierungsmöglichkeiten umgesetzt wird. Wenn Sie das Listing 380 lesen, achten Sie auf folgende interessante Techniken:

- Um nach Text zu suchen, der unabhängig vom Inhalt einfach nur fett ist, müssen Sie reguläre Ausdrücke verwenden. In OOo steht der Punkt für jedes beliebige einzelne Zeichen und das Sternchen bedeutet „Suche null oder mehr des vorhergehenden Zeichens“. So zusammengesetzt steht der reguläre Ausdruck „.*“ für jeden Text. Man braucht reguläre Ausdrücke, um „jeden Text“ zu finden, der fett ist.
- Bei der Suche mit regulären Ausdrücken steht das kaufmännische Und (&) beim Ersetzen für den gefundenen Text. Im Listing 380 bewirkt der Ersetzungstext „{{ & }}“, dass beispielsweise der gefundene Text „hallo“ zu „{{ hallo }}“ wird.
- Text, der über eine verknüpfte Vorlage auf fett gesetzt ist, wird nur gefunden, wenn auch das Attribut SearchStyles auf True steht. Wenn SearchStyles auf False gesetzt ist, wird nur Text gefunden, der hart auf fett formatiert wurde.
- Die Suche nach Text mit bestimmten Attributen führt über ein Struct-Array des Typs PropertyValue. Für jedes zu suchende Attribut muss ein Arrayelement vorhanden sein. Der Name der Property ist der Attributname und der Wert der Property ist der zu suchende Wert. Das klingt zwar kompliziert, ist aber im Listing 380 leicht nachzuvollziehen.
- Auf dieselbe Weise, wie Sie die Suchattribute setzen, können Sie auch die Ersetzungsattribute angeben. Es wird immer der längste Text innerhalb des Absatzes gefunden.

Achtung Ein Absatzende wird von LO und AOO nicht in den gefundenen String übernommen. Ein Treffer kann sich über mehrere Wörter erstrecken, er wird aber immer in demselben Absatz sein.

Listing 380. *Ersetzt fett formatierten Text.*

```
Sub ReplaceFormatting
  REM Originalcode    : Alex Savitsky
  REM Modifiziert von : Laurent Godard
  REM Modifiziert von : Andrew Pitonyak
  REM Das Ziel dieses Makros ist, alle mit einem regulären Ausdruck gefundenen
  REM FETTEN Teile durch {{ }} zu ersetzen und das Attribut Fett in NORMAL zu ändern.

  Dim oReplace
  Dim SrchAttributes(0) As New com.sun.star.beans.PropertyValue
  Dim ReplAttributes(0) As New com.sun.star.beans.PropertyValue

  oReplace = ThisComponent.createReplaceDescriptor()

  oReplace.SearchString = ".*"            'Regulärer Ausdruck. Steht für jeden Text.
  oReplace.ReplaceString = "{{ & }}"      'Das & nimmt den gefundenen Text auf.
  oReplace.SearchRegularExpression = True  'Reguläre Ausdrücke verwenden? Ja.
```

```
    oReplace.searchStyles = True            'Vorlagen mit einschließen? Ja.
    oReplace.searchAll = True               'Das ganze Dokument durchsuchen? Ja.

    REM Nach diesem Attribut soll gesucht werden (Textauszeichnung Fett).
    SrchAttributes(0).Name = "CharWeight"
    SrchAttributes(0).Value = com.sun.star.awt.FontWeight.BOLD

    REM Mit diesem Attribut soll ersetzt werden (Textauszeichnung Normal).
    ReplAttributes(0).Name = "CharWeight"
    ReplAttributes(0).Value = com.sun.star.awt.FontWeight.NORMAL

    REM Setzt die Attribute im Ersetzen-Deskriptor.
    oReplace.SetSearchAttributes(SrchAttributes())
    oReplace.SetReplaceAttributes(ReplAttributes())

    REM Nun an die Arbeit!
    ThisComponent.replaceAll(oReplace)
End Sub
```

Tabelle 146 stellt die unterstützten Zeichen für die regulären Ausdrücke vor. Für manche Zeichengruppen definiert der Posix-Standard benannte Klassenausdrücke, die alternativ verwendet werden können. Sie beginnen immer mit „[:“ und enden mit „:]“, zum Beipiel „[:any:]“ für jedes beliebige Zeichen. Die eckigen Klammern sind hier Bestandteil des Ausdrucks. Sie werden auch in einer Zeichenauswahlgruppe („[…]“) ohne Kennzeichnung verwendet, zum Beispiel „[[:digit:]+-]“ für Ziffer oder „+“ oder „-“.

Tabelle 146. *Die unterstützten Zeichen für reguläre Ausdrücke.*

Zeichen	Beschreibung
. [:any:]	Joker. Steht für jedes beliebige Einzelzeichen. Der Ausdruck „H.se“ bzw. „H[:any:]se“ findet sowohl „Hase“ als auch „Hose“.
*	Quantifizierer. Steht für jede Anzahl des davor stehenden Zeichens (einschließlich keinmal). Der Ausdruck „Se*le“ findet Übereinstimmungen in den Wörtern „Slevogt“, „Selenit“, „Seele“ und „Seeelefant“, um nur einige zu nennen.
^	Steht für den Anfang eines Absatzes. Der Ausdruck „^Bob“ findet das Wort „Bob“ nur, wenn es am Anfang eines Absatzes steht. Der Ausdruck „^.“ findet das erste Zeichen in einem Absatz.
$	Steht für das Ende eines Absatzes. Der Ausdruck „Bob$“ findet das Wort „Bob“ nur, wenn es am Ende eines Absatzes steht. Das Absatzende selbst ist nicht Bestandteil des Treffers.
^$	Steht für einen leeren Absatz. Ist als Kombination hier nur aufgeführt, weil es so häufig gebraucht wird.
+	Quantifizierer. Steht für jede Anzahl des davor stehenden Zeichens (aber mindestens einmal). Ist auch praktisch mit dem Joker „.“. Zum Beispiel findet der Ausdruck „t.+s“ einen Textabschnitt, der mit „t“ beginnt und mit „s“ endet, wie „ttwäs“ in „Bettwäsche“.
?	Quantifizierer. Steht für das optionale Vorkommen des davor stehenden Zeichens. Sie können also Wörter finden, in denen an einer Stelle ein bestimmtes Zeichen vorkommt oder nicht vorkommt: zum Beispiel findet der Ausdruck „Autos?“ sowohl „Auto“ als auch „Autos“.
.*? .+?	Ein Quantifizierer („*“ oder „+“) findet normalerweise das größtmögliche Ergebnis, er ist „gierig“. Ein „?“ hinter dem Quantifizierer findet jedoch das kleinstmögliche Ergebnis, es macht ihn „träge“. Beispielstext: „Schutzhütte“ Gierige Suche: RegExp: „h.*t“ 1 Treffer: „hutzhütt“ Träge Suche: RegExp: „h.*?t“ 2 Treffer: „hut“ und „hüt“

(?=…) (?!…)	Lookahead. Definiert eine Teilzeichenfolge, die hinter einer Zeichenfolge vorhanden sein muss [positiv: „(?=…)“] oder nicht vorhanden sein darf [negativ: „(?!…)“], damit eine Übereinstimmung vorliegt. Sie ist jedoch nicht im Ergebnisstring enthalten. Beispielstext: „48 Std. = 2 Tage“ Positiver Lokahead: RegExp: “\d+(?= Tage)“ Treffer: „2“ Negativer Lookahead: RegExp: „\d+(?! Tage)“ Treffer: „48“
(?<=…) (?<!…)	Lookbehind. Definiert eine Teilzeichenfolge, die vor einer Zeichenfolge vorhanden sein muss [positiv: „(?<=…)“] oder nicht vorhanden sein darf [negativ: „(?<!…)“], damit eine Übereinstimmung vorliegt. Sie ist jedoch nicht im Ergebnisstring enthalten. Beispielstext: „Pos. 4 = € 350“ Positiver Lookbehind: RegExp: „(?<=Pos.)\d+“ Treffer: „4“ Negativer Lookbehind: RegExp: „(?<!Pos.)\d+“ Treffer: „350“
\	Der Backslash hat zwei Aufgaben. Zusammen mit den unten beschriebenen Folgezeichen „b“, „d“, „D“, „n“, „t“, „s“, „S“,„w“, „W“, „>“, „<“ oder „x“ ist er Teil eines Spezialausdrucks. In allen anderen Fällen steuert er, dass das folgende Zeichen als Literal und nicht als Spezialzeichen gewertet wird. Er ist also nur wirklich erforderlich bei der Suche nach Zeichen, die wie zum Beispiel „$“ oder „+“ in regulären Ausdrücken spezielle Aufgaben haben. Ein „M“ findet man mit „\M“ genauso wie mit „M“, aber ein Pluszeichen findet man nur mit „\+“.
\d [:digit:]	Findet eine Ziffer. Gleichbedeutend mit „[0-9]“. Zum Beispiel findet „\d“ bzw „[:digit:]“ eine Zahl mit nur einer Ziffer und „\d+“ bzw. „[:digit:]+“ eine Zahl mit einer oder mehreren Ziffern.
\D	Negierung von „\d“. Findet alles außer Ziffern.
\s [:space:]	Findet Leerraum, also normales und geschütztes Leerzeichen, Tabulator und manuellen Zeilenumbruch.
\S	Negierung von „\s“. Findet alles außer Leerraum.
\w [:word:]	Findet ein Zeichen, das in einem Wort vorkommen kann: Buchstaben (auch mit Akzenten), Ziffern und „_“. Gleichbedeutend mit „[A-Za-z0-9_]“. „[:word:]“ ist nur in LO verfügbar.
\W	Negierung von „\w“. Findet jedes Zeichen, das nicht in einem Wort vorkommen kann: Leerzeichen, Satzzeichen usw.
\t	Findet einen Tabulatorschritt. Im Ersetzen-Feld wird damit ein Tabulatorschritt eingefügt.
\b	Steht für eine Wortbegrenzung. Kann vor oder nach einer Zeichenfolge verwendet werden. Beispielstext: „Textbuch buchstäblich“ RegExp: „.*buch\b“ Treffer: „Textbuch“ RegExp: „\bbuch.*“ Treffer: „buchstäblich“
\>	Steht für ein Wortende hinter den davor angegebenen Zeichen. Der Ausdruck „.*buch\>“ ist funktionsgleich mit „.*buch\b“ (s.o.).
\<	Steht für einen Wortanfang vor den dahinter angegebenen Zeichen. Der Ausdruck „\<buch.*“ ist funktionsgleich mit „\bbuch.*“ (s.o.).
\xXXXX	Steht für ein Zeichen, dessen Unicodewert durch die Hexadezimalzahl bestimmt wird. Statt XXXX sind die Ziffern des Unicodewerts anzugeben, immer vierstellig.
\n	Mit unterschiedlicher Bedeutung beim Suchen und Ersetzen. Beim Suchen wird ein manueller Zeilenumbruch gefunden – der mit Shift+Enter eingefügt wurde. Beim Ersetzen wird ein neuer Absatz erzeugt. Man kann damit alle manuellen Zeilenumbrüche durch eine Absatzschaltung ersetzen.
&	Steht im Ersetzungstext als Platzhalter für den gefundenen Text. Im Listing 380 wird „&“ verwendet, um fetten Text mit doppelten geschwungenen Klammern zu umrahmen: „{{“ und „}}“.
[abc123]	Zeichenauswahl. Steht für jedes einzelne Zeichen innerhalb der Klammern (nicht als Zeichenfolge). Der Ausdruck „t[ex]+t“ findet zum Beispiel „text“, „teet“ und „txeeet“; um nur einige zu nennen.
[a-e]	Zeichenauswahl als Von-Bis-Bereich. Der Ausdruck „[a-e]“ ist identisch mit „[abcde]“, und „[a-ex-z]“ ist identisch mit „[abcdexyz]“. „[A-Za-z0-9]“ steht für einen beliebigen Buchstaben oder eine beliebige Ziffer.
[^a-e]	Der Zirkumflex innerhalb einer Zeichenauswahl steht für alle Zeichen, die nicht zu der Auswahl gehören. Zum Beispiel findet „[^a-e]” alle Zeichen, die nicht zur Auswahl a-e gehören.

|	Oder-Zeichen. Steht zwischen zwei Strings, die beide gefunden werden sollen. Zum Beispiel findet „Paul|Anne“ sowohl Paul als auch Anne.
{2}	Häufigkeitsangabe. Gibt an, wie oft das vorangehende Zeichen vorkommen soll. Zum Beispiel findet „[0-9]{3}“ jede dreistellige Zahl. Dabei ist aber zu beachten, dass auch die ersten drei Ziffern einer längeren Zahl gefunden werden, wenn nicht gleichzeitig die Suche nach ganzen Wörtern aktiviert ist.
{1,2}	Von-Bis-Häufigkeitsangabe. Die Zahl vor dem Komma gibt an, wie oft das Zeichen mindestens vorkommen muss. Die Zahl nach dem Komma gibt an, wie oft das Zeichen höchstens vorkommen darf. Zum Beispiel findet „[0-9]{1,4}“ jede Zahl, die 1-4 Ziffern hat und die ersten 4 Ziffern längerer Zahlen.
( )	Gruppierung. Text in runden Klammern wird als Gruppe behandelt, auf die rückwärts referenziert werden kann. Die Referenzierungen werden in der Reihenfolge der Gruppen als „\1“, „\2“ ... angegeben. Zum Beispiel findet „([0-9]{3})-[0-9]{2}-\1“ den Text „123-45-123“, aber nicht „123-45-678“. Gruppierungen sind auch ohne Rückwärtsreferenzierung hilfreich. Zum Beispiel findet „(er|sie|es)$“ jeden Absatz, der mit „er“, „sie“ oder „es“ endet.
[:print:]	Benannte Zeichenklasse. Findet druckbare Zeichen.
[:cntrl:]	Benannte Zeichenklasse. Findet nicht-druckbare Zeichen.
[:alnum:]	Benannte Zeichenklasse. Findet alphanumerische Zeichen (Ziffern und Buchstaben). Gleichbedeutend mit „[A-Za-z0-9]“.
[:alpha:]	Benannte Zeichenklasse. Findet Groß- oder Kleinbuchstaben. Gleichbedeutend mit „[A-Za-z]“.
[:lower:]	Benannte Zeichenklasse. Findet Kleinbuchstaben, wenn gleichzeitig die Option „Groß-/Kleinschreibung“ aktiviert ist.
[:upper:]	Benannte Zeichenklasse. Findet Großbuchstaben, wenn gleichzeitig die Option „Groß-/Kleinschreibung“ aktiviert ist.

14.8. Textcontent

Der Hauptinhalt eines Writer-Dokuments ist einfacher Text, der in Absätzen gespeichert und hintereinander aufgeführt wird. Wenn Absatzteile aufgelistet werden, nutzt jeder Teil denselben Satz von Eigenschaften. Generell muss Textcontent von dem Dokument erzeugt werden, das ihn aufnimmt. Nach der Erzeugung wird der Textcontent an einer bestimmten Position in das Dokument eingefügt. Absätze werden jedoch nicht gesondert erzeugt und dann eingefügt (s. Listing 345 vorne in diesem Kapitel). Der Absatztext wird als String eingefügt und neue Absätze als Steuerzeichen (s. Tabelle 133). Komplexerer Textcontent wird normalerweise mit der Objektmethode insertTextContent() eingefügt (s. Listing 346). Es gibt auch andere, seltener genutzte Methoden zum Einfügen von Textcontent – zum Beispiel Inhalt aus der Zwischenablage (s. Listing 393 weiter unten in diesem Kapitel) und ganze Dokumente einzufügen (s. Listing 381).

Listing 381. *Fügt ein Dokument an einem Textcursor ein.*

```
oCursor.insertDocumentFromURL(sFileURL, Array())
```

Der meiste Textcontent ist benannt und über den Namen erreichbar (s. Tabelle 147). Der am meisten verwendete Textcontent-Typ der Tabelle 147 ist zweifellos die Texttabelle.

Tabelle 147. *In einem Textdokument vorkommender Content.*

Content-Typ	Mechanismus	Zugriffsmethode
Fußnoten	Indexierter Zugriff	getFootnotes()
Endnoten	Indexierter Zugriff	getEndnotes()
Querverweise	Benannter Zugriff	getReferenceMarks()
Grafikobjekte	Benannter Zugriff	getGraphicObjects()
Eingebettete Objekte	Benannter Zugriff	getEmbeddedObjects()
Texttabellen	Benannter Zugriff	getTables()

Content-Typ	Mechanismus	Zugriffsmethode
Textmarken	Benannter Zugriff	getBookmarks()
Vorlagenfamilien	Benannter Zugriff	getStyleFamilies()
Verzeichnisse	Indexierter Zugriff	getDocumentIndexes()
Textfelder	Enumerierter Zugriff	getTextFields()
Text-Masterfelder	Benannter Zugriff	getTextFieldMasters()
Textrahmen	Benannter Zugriff	getTextFrames()
Textbereiche	Benannter Zugriff	getTextSections()

Tipp Content, der über Namen erreichbar ist, bietet auch den indexierten Zugriff.

14.9. Texttabellen

Writer-Dokumente unterstützen auch Texttabellen, somit kann man aus den Dokumenten direkt auf die Tabellen zugreifen. Obwohl Texttabellen als Textcontent zusammen mit den Absätzen enumeriert werden (s. Listing 348), greift man normalerweise über ihren Namen oder Index zu, s. Listing 382 und Bild 100.

Listing 382. *Zeigt die Enumeration von Texttabellen.*

```
Sub EnumerateAllTextTables
  Dim oTables     'Alle Texttabellen
  Dim s$, sNames$ 'Ausgabestrings
  Dim i%          'Indexvariable

  oTables = ThisComponent.TextTables
  REM Zuerst der indexbasierte Zugriff auf die Tabellen
  s = "Tabellen indexiert:" & Chr$(10)
  For i = 0 To oTables.getCount() - 1
    s = s & "Tabelle " & (i + 1) & " = " & oTables(i).Name & Chr$(10)
  Next

  REM Jetzt die Liste der Tabellennamen
  s = s & Chr$(10) & "Tabellen benannt:" & Chr$(10)
  sNames = Join(oTables.getElementNames(), Chr$(10))
  MsgBox s & sNames, 0, "Tabellen"
End Sub
```

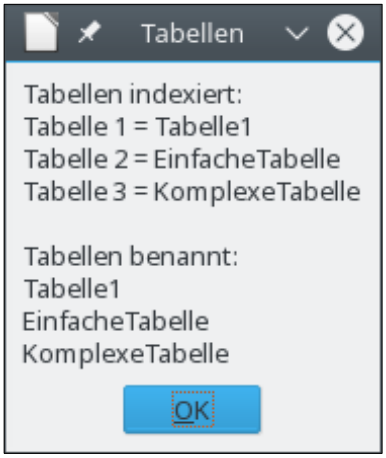


Bild 100. *Tabellen in einem Dokument, aufgelistet nach Index und nach Namen.*

Tipp Bei den meisten Textcontenttypen erfolgen Zugriff, Erzeugen, Einfügen und Löschen auf dieselbe Weise. Lernen Sie, wie das bei Tabellen geschieht, und Sie werden auch mit anderem Textcontent, zum Beispiel Textmarken, vertraut sein.

Wie das so meistens mit Textcontent ist, müssen Tabellen erst vom Dokument erzeugt werden, bevor sie ins Dokument eingefügt werden. Das Makro im Listing 383 fügt eine Tabelle mit dem Namen „Beispieltabelle“ ein, falls sie nicht schon existiert, und löscht sie, falls sie existiert.

Listing 383. *Fügt eine Texttabelle ein und löscht sie wieder.*

```
Sub InsertDeleteTable
  Dim oTable       'Neu erzeugte Tabelle
  Dim oTables      'Alle Texttabellen
  Dim oInsertPoint 'Wo die Tabelle eingefügt wird
  Dim sTableName As String

  sTableName = "Beispieltabelle"
  oTables = ThisComponent.TextTables

  If oTables.hasByName(sTableName) Then
    oTable = oTables.getByName(sTableName)

    REM Obwohl dies der korrekt Weg zu sein scheint, Textcontent zu löschen:
    REM Was ist, wenn die Tabelle gar nicht in das Textobjekt des Dokuments
    REM eingefügt wurde? oTable.dispose() mag wohl der sicherere Weg sein.
    ThisComponent.Text.removeTextContent(oTable)
  Else

    REM Das Dokument muss die Texttabelle erzeugen.
    oTable = ThisComponent.createInstance("com.sun.star.text.TextTable")
    oTable.initialize(2, 3) 'Zwei Zeilen, drei Spalten.

    REM Wenn eine Textmarke "TabelleEinfügenHier" existiert, wird diese Stelle
    REM als Einfügeposition genommen. Wenn diese Textmarke nicht existiert,
    REM dann wird das Dokumentende genommen.
    If ThisComponent.getBookmarks().hasByName("TabelleEinfügenHier") Then
      oInsertPoint =_
        ThisComponent.getBookmarks().getByName("TabelleEinfügenHier").getAnchor()
    Else
      oInsertPoint = ThisComponent.Text.getEnd()
    End If

    REM Nun wird die Texttabelle eingefügt.
    REM Man beachte, dass das Textobjekt des oInsertPoint-Textrange
    REM statt des Dokument-Textobjekts verwendet wird.
    oInsertPoint.getText().insertTextContent(oInsertPoint, oTable, False)

    REM Die Objektmethode setData() funktioniert NUR mit numerischen Daten.
    REM Die Objektmethode setDataArray() erlaubt jedoch auch Strings.
    oTable.setDataArray(Array(Array(0, "Eins", 2), Array(3, "Vier", 5)))
    oTable.setName(sTableName)
  End If
End Sub
```

Tipp Grundsätzlich ist es besser, die Tabelleneigenschaften vor dem Einfügen der Tabelle in das Dokument zu setzen. Dadurch wird verhindert, dass der Bildschirm flackert, wenn das Objekt modifiziert und auf dem Bildschirm neu gezeichnet wird. Doch wenn die Daten im Listing 383 vor dem Einfügen modifiziert werden, wird aufgrund eines Bugs seit OOo 1.1.0 der Tabellenname geändert. Er erhält am Ende ein zusätzliches Müllzeichen. Beachten Sie, dass der Name manchen Textcontents, zum Beispiel des Text-Masterfeldes, nach dem Einfügen in das Dokument nicht mehr geändert werden kann.

Das Makro im Listing 383 zeigt ein paar nützliche Techniken:

- Ein benannter Textcontent wird gesucht und referenziert. Beachten Sie, dass die Suche nach der Tabelle und der Textmarke sehr ähnlich sind.
- Eine Textmarke wird genutzt.
- Eine Texttabelle wird erzeugt, initialisiert und an einer Position eingefügt, die durch eine Textmarke gegeben ist.
- Textcontent wird entfernt.
- Eine Tabelle wird mit Daten initialisiert.
- Der Tabellenname wird gesetzt.

14.9.1. Das richtige Textobjekt nutzen

Es ist sehr wichtig, dass Sie das richtige Textobjekt nutzen. Es kann sein, dass das Textobjekt in einem Textbereich oder in einer Tabellenzelle nicht dasselbe wie das vom Dokument zurückgegebene ist. Jeder Textrange ist mit einem eigenen Textobjekt ausgestattet. Wenn Sie versuchen, Objektmethoden eines Textobjekts auf einen Textrange mit einem anderen Textobjekt anzuwenden, erzeugen Sie einen Fehler. Im Listing 383 wird eine Tabelle gelöscht, mit einer einzigen Zeile, s. Listing 384.

Listing 384. *Was ist, wenn die Tabelle nicht im Textobjekt des Dokuments enthalten ist?*

```
ThisComponent.Text.removeTextContent(oTable)
```

Der Code im Listing 384 geht davon aus, dass die Tabelle im Textobjekt des Dokuments enthalten ist. Falls sie es nicht ist, wird der Code einen Fehler auslösen. Wenn es auch nur sehr selten zu einem Fehler mit dem Code im Listing 384 kommt, wird er aber mit Sicherheit zum unpassendsten Augenblick auftreten. Das Makro funktioniert, weil das Beispieldokument so aufgebaut war, dass die Tabelle in das Textobjekt des Dokuments eingefügt wurde. Jede der beiden Lösungen im Listing 385 ist wohl eine bessere Wahl zum Löschen einer Tabelle.

Listing 385. *Zwei sichere Methoden, die Tabelle zu löschen.*

```
oTable.getAnchor().getText().removeTextContent(oTable)
oTable.dispose()
```

Wenn im Listing 383 zum zweiten Mal ein Textobjekt genutzt wird, kommt es aus dem von einer Textmarke zurückgegebenen Anker – das ist ein sicherer Weg.

Listing 386. *Ein sicherer Weg, ein Textobjekt zu holen.*

```
oInsertPoint.getText().insertTextContent(oInsertPoint, oTable, False)
```

Sollte der Textrange oInsertPoint nicht im Textobjekt des Dokuments enthalten sein, wird der Versuch fehlschlagen, die Tabelle über das Textobjekt des Dokuments einzufügen. Nur Sie allein können entscheiden, welche Absicherungen Sie beim Zugriff auf Textobjekte brauchen. Betrachten Sie die Auswahltext-Rahmenstruktur. Mit welchem Textobjekt werden Textcursors erzeugt und verglichen? Können Sie den Code robuster machen?

14.9.2. Methoden und Eigenschaften

Die von Texttabellen unterstützten Methoden sind denen sehr ähnlich, die in Calc-Dokumenten von Tabellenblättern unterstützt werden (s. Kapitel 15. Tabellendokumente). Tabelle 148 listet die von Texttabellen unterstützten Methoden auf.

Tabelle 148. *Von Texttabellen unterstützte Objektmethoden.*

Methode	Beschreibung
autoFormat(name)	Wendet das genannte Autoformat auf die Tabelle an.
createCursorByCellName(name)	XTextTableCursor an der genannten Zelle positioniert.
createSortDescriptor()	Sortierkriterien als Array von Property-Werten.
dispose()	Zerstört ein Textobjekt, wodurch es auch aus dem Dokument entfernt wird.
getAnchor()	Gibt einen Textrange mit der Verankerung der Tabelle zurück. Dadurch kann Textcontent leicht vor oder hinter einer Texttabelle eingefügt werden.
getCellByName(name)	Gibt ein XCell zurück, basierend auf dem Zellnamen, zum Beispiel "B3".
getCellByPosition(spalte, zeile)	Die Zählung beginnt mit Null. Kann bei komplexen Tabellen zu Fehlern führen.
getCellNames()	Stringarray der Namen der in der Tabelle enthaltenen Zellen.
getCellRangeByName(name)	XCellRange, basierend auf Zellnamen, zum Beispiel "A1:B4". Kann bei komplexen Tabellen zu Fehlern führen.
getCellRangeByPosition(links, oben, rechts, unten)	XCellRange, basierend auf numerischen Angaben.
getColumnDescriptions()	Stringarray mit Spaltenbeschreibungen. Kann bei komplexen Tabellen zu Fehlern führen.
getColumns()	XTableColumns-Objekt, das die Spalten indexiert. Unterstützt auch insertByIndex(index, anzahl) und removeByIndex(index, anzahl).
getData()	Liest numerische Daten als verschachtelte Wertsequenz aus (Arrays in Arrays). Kann bei komplexen Tabellen zu Fehlern führen.
getDataArray()	Wie getData(), kann aber auch String oder Double enthalten.
getName()	Tabellenname als String.
getRowDescriptions()	Stringarray mit Zeilenbeschreibungen. Kann bei komplexen Tabellen zu Fehlern führen.
getRows()	XTableRows-Objekt, das die Zeilen indexiert. Unterstützt auch insertByIndex(index, anzahl) und removeByIndex(index, anzahl).
initialize(zeilen, spalten)	Legt die Anzahl der Zeilen und Spalten fest. Muss gemacht werden, bevor die Tabelle eingefügt wird (s. Listing 346).
setColumnDescriptions(string())	Setzt die Spaltenbeschreibungen aus einem Stringarray.
setData(Double())	Fügt numerische Daten als verschachtelte Wertsequenz ein. Kann bei komplexen Tabellen zu Fehlern führen.
setDataArray(array())	Wie setData(), kann aber auch String oder Double enthalten.
setName(name)	Legt den Tabellennamen fest.
setRowDescriptions(string())	Setzt die Zeilenbeschreibungen aus einem Stringarray.
sort(array())	Sortiert die Tabelle auf der Basis eines Sortierdeskriptors.

Texttabellen unterstützen auch eine Anzahl von Eigenschaften, s. Tabelle 149. Viele von denen werden auch von Absätzen unterstützt, s. Tabelle 136.

Tabelle 149. *Eigenschaften im Service com.sun.star.text.TextTable.*

Eigenschaft	Beschreibung
BreakType	Art des Umbruchs, der am Tabellenanfang vorgenommen wird (s. BreakType in Tabelle 136).
LeftMargin	Abstand der Tabelle nach links in 1/100 mm als Long Integer. Dabei darf die Eigenschaft HoriOrient nicht auf FULL stehen.
RightMargin	Abstand der Tabelle nach rechts in 1/100 mm als Long Integer. Dabei darf die Eigenschaft HoriOrient nicht auf FULL stehen.
HoriOrient	Horizontale Ausrichtung als Konstante der Gruppe com.sun.star.text.HoriOrientation. Standardwert ist com.sun.star.text.HoriOrientation.FULL. • NONE = 0 – Keine Ausrichtung. • RIGHT = 1 – Rechts ausgerichtet. • CENTER = 2 – Zentriert ausgerichtet. • LEFT = 3 – Links ausgerichtet. • INSIDE = 4 – (Noch nicht unterstützt) • OUTSIDE = 5 – (Noch nicht unterstützt) • FULL = 6 – Volle Breite (nur für Texttabellen). • LEFT_AND_WIDTH = 7 – Abstand nach links und Breite sind definiert.
KeepTogether	Falls True, wird ein Seiten- oder Spaltenumbruch zwischen dieser Tabelle und dem folgenden Absatz oder der folgenden Tabelle verhindert.
Split	Falls False, wird die Tabelle nicht auf zwei Seiten umbrochen.
PageDescName	String, der vor der Tabelle einen Seitenwechsel bewirkt. Die neue Seite nutzt diesen String als Namen der Seitenvorlage (s. PageDescName in Tabelle 136).
PageNumberOffset	Neue Seitennummer bei einem Seitenwechsel (s. PageNumberOffset in Tabelle 136).
RelativeWidth	Tabellenbreite, relativ zur Umgebung, als Short Integer.
IsWidthRelative	Falls True, ist die relative Breite wirksam.
RepeatHeadline	Falls True, wird die erste Tabellenzeile auf jeder neuen Seite wiederholt.
ShadowFormat	Schattenformat – Typ, Farbe und Größe (s. ParaShadowFormat in Tabelle 136).
TopMargin	Abstand der Tabelle nach oben in 1/100 mm als Long Integer.
BottomMargin	Abstand der Tabelle nach unten in 1/100 mm als Long Integer.
BackTransparent	Falls True, ist die Hintergrundfarbe transparent.
Width	Absolute Tabellenbreite als Long Integer – nur-lesbare Eigenschaft.
ChartRowAsLabel	Falls True, wird die erste Zeile als Achsenbeschriftung genommen, falls ein Diagramm erstellt wird.
ChartColumnAsLabel	Falls True, wird die erste Spalte als Achsenbeschriftung genommen, falls ein Diagramm erstellt wird.
TableBorder	Tabellenumrandung als Struct com.sun.star.table.TableBorder. Das Struct enthält eine Anzahl komplizierter Eigenschaften: • Die Eigenschaften TopLine, BottomLine, LeftLine, RightLine, HorizontalLine und VerticalLine sind Structs des Typs BorderLine, wie bei der Eigenschaft LeftBorder in Tabelle 136 gezeigt. • Die Eigenschaft Distance enthält den Abstand zwischen Umrandung und Inhalt. • Jede Umrandungslinie kann mit True oder False an- oder abgeschaltet werden. Es sind die Eigenschaften IsTopLineValid, IsBottomLineValid, IsLeftLineValid, IsRightLineValid, IsHorizontalLineValid, IsVerticalLineValid und IsDistanceValid.

Eigenschaft	Beschreibung
TableColumnSeparators	Die Spaltenbreiten als Array von Tabellenspaltenseparatoren. Jeder Separator ist ein Struct des Typs com.sun.star.text.TableColumnSeparator: • Position als Short Integer definiert die horizontale Position eines Zellseparators. • IsVisible bestimmt, ob der Separator sichtbar ist. Die Breite einer Zelle wird von der Position des Separators zwischen benachbarten Zellen definiert. Wenn zwei Zellen miteinander verbunden sind, ist der Separator verborgen, nicht entfernt. Die Positionswerte sind relativ zur Eigenschaft TableColumnRelativeSum. Diese Eigenschaft ist nur gültig für Tabellen, in denen jede Zeile dieselbe Struktur hat. Ist das nicht der Fall, müssen die Separatoren von den einzelnen Zeilenobjekten geholt werden.
TableColumnRelativeSum	Summe der Spaltenbreite-Werte aus TableColumnSeparators als Short Integer.
BackColor	Hintergrundfarbe der Tabelle als Long Integer.
BackGraphicURL	URL der Hintergrundgrafik der Tabelle.
BackGraphicFilter	Name des Grafikfilters für die Hintergrundgrafik der Tabelle.
BackGraphicLocation	Positionierung der Hintergrundgrafik (s. ParaBackGraphicLocation in Tabelle 136).

14.9.3. Einfache und komplexe Tabellen

Vereinfacht gesagt ist eine Texttabelle ein Satz von Textzeilen und -spalten. Alle Tabellen in diesem Buch haben die Form einfacher Texttabellen. OOo unterstützt sowohl einfache als auch komplexe Tabellen. Wie ihr Name schon sagt, sind die Zellen einer einfachen Tabelle in einer einfachen Gitterstruktur angeordnet (s. Tabelle 150). Die Benennung der Spalten ist alphabetisch mit A startend, und die Benennung der Zeilen numerisch mit 1 startend. Die Objektmethode getCellByName() verwendet diese Benennung zur Rückgabe einer bestimmten Zelle. Die ähnliche Objektmethode getCellByPosition() verwendet die fortlaufende Zählung der Spalten und Zeilen. Diese Zählung startet mit null. Daher wird mit der Anforderung (1, 2) die Zelle mit dem Namen „B3“ zurückgegeben.

Tabelle 150. *Die Benennung der Zeilen ist numerisch, die der Spalten alphabetisch.*

A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4

Mit getRows() erhält man die Zeilen der Texttabelle. Über das zurückgegebene Objekt kann man erfahren, wie viele Zeilen vorhanden sind, und man kann einzelne Zeilen referenzieren, einfügen und löschen.

Tabelle 151. *Hauptzugriffsmethoden mit einem Tabellenzeilenobjekt.*

Methode	Beschreibung
getByIndex(index)	Referenziert eine bestimmte Zeile.
getCount()	Anzahl der Zeilen in der Tabelle.
hasElements()	Gibt zurück, ob es überhaupt Zeilen in der Tabelle gibt.
insertByIndex(index, anzahl)	Fügt einer Tabelle Zeilen hinzu.
removeByIndex(index, anzahl)	Entfernt Zeilen aus einer Tabelle.

Obwohl es nicht schwierig ist, einzelne Zeilen aufzulisten, so ist eine Zeile doch nicht dazu geeignet, auf die darin enthaltenen Zellen zuzugreifen. Das Objekt einer einzelnen Zeile wird hauptsächlich für folgende Zwecke genutzt:

- die Zeilenhöhe über das Attribut Height zu setzen,
- IsAutoHeight auf True zu setzen, so dass die Zeilenhöhe automatisch angepasst wird,
- IsSplitAllowed auf False zu setzen, so dass eine Zeile nicht beim Seitenumbruch geteilt werden kann,
- die TableColumnSeparators zu bearbeiten, um die Spaltenbreite zu ändern.

Mit getColumns() erhält man die Spalten der Texttabelle. Die Methoden des Spaltenobjekts sind dieselben wie die des Zeilenobjekts. Es ist jedoch nicht möglich, aus dem Spaltenobjekt eine bestimmte Spalte anzusprechen. Die Methode getByIndex() existiert zwar, gibt aber Null zurück. Obwohl ich eigentlich erwartet hätte, die Spaltenbreite über das Objekt einer bestimmten Spalte zu setzen, so muss man doch die Tabellenspaltentrenner modifizieren, die es im Zeilenobjekt gibt.

Eine komplexe Texttabelle ist eine Texttabelle, die nicht so einfach ist. Genauer gesagt, sie enthält Zellen, die entweder geteilt oder verbunden sind. Zur Demonstration einer komplexen Texttabelle starten Sie mit Tabelle 150 und führen die folgenden Schritte aus, um Tabelle 155 zu erhalten.

1. Rechtsklicken Sie in Zelle A2 und wählen Sie aus dem Menü **Zelle | Teilen | Horizontal**. Aus der Zelle A2 werden nun zwei Zellen. Aus der Sicht der Zellbenennungen wurde eine neue dritte Zeile eingefügt, die nur eine Spalte enthält. Für die API sind es fünf Zeilen und vier Spalten, s. Tabelle 152.

   Tabelle 152. *Zelle A2 wird horizontal geteilt.*

A1	B1	C1	D1
A2	B2	C2	D2
(A3)			
A3=>A4	B3=>B4	C3=>C4	D3=>D4
A4=>A5	B4=>B5	C4=>C5	D4=>D5

2. Rechtsklicken Sie in Zelle B2 und wählen Sie **Zelle | Teilen | Vertikal**. Aus Zelle B2 werden nun die beiden getrennten Zellen B2 und C2. Augenscheinlich wurde eine neue Spalte an der Zelle B2 eingefügt. Für die API sind es aber immer noch fünf Zeilen und vier Spalten, s. Tabelle 153.

   Tabelle 153. *Zelle B2 wird vertikal geteilt.*

A1	B1		C1	D1
A2	B2	(C2)	C2=>D2	D2=>E2
A3				
A4	B4		C4	D4
A5	B5		C5	D5

3. Wählen Sie die Zellen A4 und B4 aus, rechtsklicken Sie darauf und wählen Sie **Zelle | Verbinden**.

   Tabelle 154. *Komplexe Tabelle nach dem Verbinden von Zellen in derselben Zeile.*

A1	B1		C1	D1
A2	B2	C2	D2	E2

A3				
A4, (B4) => A4			C4=>B4	D4=>C4
A5	B5		C5	D5

4. Wählen Sie die Zellen B4 und C5 aus, rechtsklicken Sie darauf und wählen Sie **Zelle | Verbinden**. Die Zelle C5 verschwindet einfach.

Tabelle 155. *Komplexe Tabelle nach dem Verbinden von Zellen in derselben Spalte.*

A1	B1		C1	D1
A2	B2	C2	D2	E2
A3				
A4			B4, (C5)=>B4	C4
A5	B5			D5

Tipp Nicht alle Objektmethoden funktionieren mit komplexen Tabellen. Zum Beispiel bewirken getData() und setData() für komplexe Tabellen einen Laufzeitfehler, der durchaus sinnvoll ist.

Obwohl die Objektmethode getCellByName() wie erwartet auch für komplexe Tabellen funktioniert, kann getCellByPosition() nicht alle Zellen zurückgeben, denn es kann nur eine Spalten- und eine Zeilennummer akzeptieren. Die Objektmethode getCellNames() gibt aber die Namen der Zellen einer Tabelle zurück (s. Listing 387). Danach können Sie über die Zellnamen jede einzelne Zelle der Tabelle erreichen.

Listing 387. *Mit Join wird aus dem Array der Zellnamen einer Tabelle ein String.*

```
MsgBox Join(oTable.getCellNames(), "|")
```

Positionieren Sie den Textcursor in einer Texttabelle und starten Sie das folgende Makro, um den Text jeder Zelle in den Zellnamen zu ändern. Dadurch wird jeglicher existierende Text in der Tabelle gelöscht

Listing 388. *Ändert den Text jeder Zelle in den Zellnamen.*

```
Sub SetCellNames
  Dim oTable           'Die Tabelle
  Dim sNames           'Die Namen der Zellen
  Dim i As Integer

  oTable = ThisComponent.CurrentController.ViewCursor.TextTable
  If IsNull(oTable) Then
    Exit Sub
  End If
  sNames = oTable.getCellNames()
  For i = LBound(sNames) To UBound(sNames)
    'Schreibt in jede Zelle den jeweiligen Zellnamen.
    oTable.getCellByName(sNames(i)).setString(sNames(i))
  Next
End Sub
```

14.9.4. Tabellen enthalten Zellen

Die Zellen in einer Texttabelle sind äußerst vielseitige Objekte, die alle Datentypen aufzunehmen vermögen. Die Zellobjekte binden sowohl das Interface XText (s. Tabelle 133 zu Beginn dieses Kapitels) als auch das Interface XCell (s. Tabelle 156) ein.

Tipp Tabellen sind in der Lage, einen Texttabellencursor zu erzeugen, der Methoden und Eigenschaften speziell für das Durchlaufen und Selektieren von Zellen mitbringt, und jede einzelne Zelle wiederum kann einen Textcursor für das lokale Zelltextobjekt produzieren.

Tabelle 156. *Methoden im Interface com.sun.star.table.XCell.*

Methode	Beschreibung
getFormula()	Liest den in die Zelle eingegebenen String aus (auch wenn es keine Formel ist).
setFormula(String)	Gibt eine neue Zellformel ein (für normalen Text: setString() vom Interface XText).
getValue()	Liest einen Fließkommawert (Double) aus der Zelle aus.
setValue(Double)	Gibt den Fließkommawert in die Zelle ein.
getType()	Gibt einen Wert der Enumeration com.sun.star.table.CellContentType aus. Gültige Werte sind EMPTY, VALUE, TEXT und FORMULA.
getError()	Fehlerwert der Zelle als Long Integer. Ist die Zelle keine Formel, ist der Fehlerwert immer null.

Jedes Zellobjekt besitzt zahlreiche Eigenschaften. Diese Eigenschaften sind Ihnen schon vertraut, denn sie werden auch in anderen Objekten genutzt. Zum Beispiel sind BackColor, BackGraphicFilter, BackGraphicLocation, BackGraphicURL, BackTransparent, BorderDistance, BottomBorder, BottomBorderDistance, LeftBorder, LeftBorderDistance, RightBorder, RightBorderDistance, TopBorder und TopBorderDistance auch für Texttabellen in der Tabelle 149 und/oder für Absätze in der Tabelle 136 definiert. Eine der ziemlich nützlichen, nur bei Zellobjekten vorkommenden Eigenschaften ist CellName. Diese brauchen Sie zur Lokalisierung des aktuellen Cursors. Das Makro im Listing 389 zeigt ein paar neue Manipulationsmöglichkeiten für Tabellen:

- Die Texttabellen, Zeilen, Spalten und Zellen unterstützen alle die Eigenschaft BackColor. Listing 389 setzt die Hintergrundfarbe der ersten Zeile auf das Hellgrau, das vielfach zur Hervorhebung von Überschriften verwendet wird.
- Die Objektmethode insertByIndex(index, anzahl) fügt neue Zeilen am Ende der Tabelle ein. Zeilen können auch inmitten oder am Anfang der Tabelle eingefügt werden.
- Der Zugriff auf einzelne Zellen erfolgt mit ihren Namen. Es werden sowohl numerische Werte als auch Strings in die Zellen eingetragen. Beachten Sie, dass für Strings die Methode setString() verwendet wird und nicht setFormula().

Tipp Obwohl in der webbasierten OOo-Dokumentation kein Unterschied zwischen Zellen in Texttabellen und solchen in Tabellenblättern gemacht wird, so unterstützen die beiden Zelltypen doch nicht dieselbe Eigenschaftsmenge. Zum Beispiel werden in einem Textdokument die Eigenschaften CellStyle, CellBackColor und RotateAngle nicht unterstützt.

Listing 389. *Einfache Manipulationen einer Texttabelle.*

```
Sub SimpleTableManipulations
  Dim oTable       'Neu erzeugte Tabelle
  Dim oTables      'Alle Texttabellen
  Dim oInsertPoint 'Wo die Tabelle eingefügt wird
  Dim sTableName As String

  sTableName = "Beispieltabelle"
  oTables = ThisComponent.TextTables
```

```
  REM Löscht die Tabelle, wenn sie existiert!
  If oTables.hasByName(sTableName) Then
    ThisComponent.Text.removeTextContent(oTables.getByName(sTableName))
  End If
  REM Das Dokument muss die Texttabelle erzeugen.
  oTable = ThisComponent.createInstance("com.sun.star.text.TextTable")
  oTable.initialize(4, 3) 'Vier Zeilen, drei Spalten.

    REM Wenn eine Textmarke "TabelleEinfügenHier" existiert, wird diese Stelle
    REM als Einfügeposition genommen. Wenn diese Textmarke nicht existiert,
    REM dann wird das Dokumentende genommen.
  If ThisComponent.getBookmarks().hasByName("TabelleEinfügenHier") Then
    oInsertPoint =_
      ThisComponent.getBookmarks().getByName("TabelleEinfügenHier").getAnchor()
  Else
    oInsertPoint = ThisComponent.Text.getEnd()
  End If

  oInsertPoint.getText().insertTextContent(oInsertPoint, oTable, False)

  oTable.setDataArray(Array(Array("Name", "Punkte", "Turnier"),_
    Array("Robin", 932, "SKB"), Array("Jennifer", 885, "FSK"),_
    Array("Marcel", 862, "SSKH")))
  oTable.setName(sTableName)
  REM Die erste Zeile erhält einen grauen Hintergrund.
  oTable.getRows().getByIndex(0).BackColor = RGB(235, 235, 235)

  REM removeByIndex nutzt dieselben Argumente wie insertByIndex, nämlich
  REM den Index der Stelle, an der eingefügt oder entfernt wird, gefolgt von
  REM der Anzahl der einzufügenden oder zu löschenden Zeilen. Die folgende Zeile
  REM fügt eine Zeile am Index 4 ein.
  oTable.getRows().insertByIndex(4, 1)

  REM Greift auf die einzelnen Zellen zu und gibt die Werte ein.
  oTable.getCellByName("A5").setString("Sigrid")
  oTable.getCellByName("B5").setValue(872)
  oTable.getCellByName("C5").setString("KGH")
End Sub
```

14.9.5. Handhabung eines Texttabellencursors

Obwohl Texttabellencursors spezifische Methoden zum Durchlaufen von Texttabellen einsetzen, verhalten sie sich prinzipiell nicht viel anders als ihre Textcursor-Kollegen. Sie können Zellranges auswählen und bearbeiten und Zelleigenschaften setzen.

Tipp Sie können nicht einfach eine Texttabelle aus dem Dokument holen und sie dann wieder an anderer Stelle einfügen. Wie kopiert man denn nun eine Texttabelle? S. Listing 393.

Wie bei Textcursors gibt es auch bei den Bewegungsmethoden der Texttabellencursors ein boolesches Argument, das angibt, ob die gewünschte Bewegung als Auswahlerweiterung (True) oder als einfacher Schritt (False) ausgeführt werden soll. Die Bewegungsmethoden geben auch einen booleschen Wert zurück, der meldet, ob die Bewegung erfolgreich war. Tabelle 157 listet die vom Interface XTextTableCursor definierten Methoden auf.

Tabelle 157. *Methoden im Interface com.sun.star.text.XTextTableCursor.*

Methode	Beschreibung
getRangeName()	Gibt den vom Cursor ausgewählten Zellrange als String zurück, zum Beispiel „B3:D5“.
GotoCellByName(String, boolean)	Bewegt den Cursor zu der Zelle mit dem spezifizierten Namen, gibt einen booleschen Wert zurück.
goLeft(n, boolean)	Bewegt den Cursor n Zellen nach links, gibt einen booleschen Wert zurück.
goRight(n, boolean)	Bewegt den Cursor n Zellen nach rechts, gibt einen booleschen Wert zurück.
goUp(n, boolean)	Bewegt den Cursor n Zellen nach oben, gibt einen booleschen Wert zurück.
goDown(n, boolean)	Bewegt den Cursor n Zellen nach unten, gibt einen booleschen Wert zurück.
gotoStart(boolean)	Bewegt den Cursor zur Zelle links oben.
gotoEnd(boolean)	Bewegt den Cursor zur Zelle rechts unten.
mergeRange()	Verbindet den ausgewählten Zellrange, gibt bei Erfolg True zurück.
splitRange(n, boolean)	Erzeugt n neue Zellen in jeder vom Cursor ausgewählten Zelle. Beim booleschen Argument steht True für horizontales, False für vertikales Trennen. Gibt bei Erfolg True zurück.

Mit Texttabellencursors werden Tabellenzellen getrennt und verbunden. Generell betrachte ich das als die Hauptaufgabe eines Tabellencursors. Man kann sich in der Tabelle mit einem Texttabellencursor mit Hilfe der Methoden aus der Tabelle 157 bewegen. Das Makro im Listing 390 holt sich die Zellnamen, erzeugt einen Zellcursor, der die erste Tabellenzelle enthält, und bewegt den Cursor zur letzten Zelle der Tabelle. Basierend auf dem Rangenamen wird ein Zellrange erzeugt, und dann wird die gesamte Tabelle vom aktuellen Controller ausgewählt.

Listing 390. *Wählt mit Hilfe eines Cursors eine ganze Tabelle aus.*

```
oCellNames = oTable.getCellNames()
oCursor = oTable.createCursorByCellName(oCellNames(0))
oCursor.gotoCellByName(oCellNames(UBound(oCellNames())), True)
oRange = oTable.getCellRangeByName(oCursor.getRangeName())
ThisComponent.getCurrentController.select(oRange)
```

Listing 390 zeigt, wie man alle Zellen in einer Tabelle mit Hilfe eines Tabellenzellcursors auswählt. Danach kann man mit diesem Cursor die gesamte Tabelle bearbeiten.

Mit früheren OOo-Versionen kann es aber schiefgehen, die Tabelle im aktuellen View auszuwählen. Tabellenzellcursors haben keine Probleme mit komplexen Tabellen. Aber nicht alle von Tabellen unterstützten Objektmethoden unterstützen komplexe Tabellen. Das galt auch für die Objektmethode getCellRangeByName(), s. Listing 390. Die Tabelle konnte keinen Zellrange zurückgeben, der eine geteilte Zelle am Anfang oder Ende hat. Der Zellrange „A1.2.1:C4“ hat beispielsweise einen Fehler erzeugt.

Es gibt keinen einfachen Weg, eine ganze Texttabelle zu kopieren, sei es innerhalb eines Dokuments oder zwischen verschiedenen Dokumenten. Die Zwischenablage war immer die Methode der Wahl, allgemeinen Text zu kopieren (s. Listing 393), aber falls verfügbar, ist übertragbarer Content vorzuziehen (s. Listing 394). Wählen Sie zuerst mit dem Viewcursor oder dem aktuellen Controller das zu kopierende Objekt aus, kopieren Sie dann das Objekt mit einem Dispatcher in die Zwischenablage, bewegen anschließend den Viewcursor an die Stelle der Einfügung und fügen das Objekt schließlich mit einem Dispatcher aus der Zwischenablage ein.

Wie Sie sich wohl schon gedacht haben, ist der schwierigste Teil des Prozesses die Auswahl der Tabelle mit dem Viewcursor. Paolo Mantovani, ein reger Aktiver der OOo-Mailinglisten, hat eine brillante Methode vorgestellt. Den Anfang macht die Erkenntnis, dass bei der Auswahl einer ganzen Tabelle mit dem aktuellen Controller der Viewcursor an den Anfang der ersten Zelle gesetzt wird, s. Listing 391.

Listing 391. *Setzt den Cursor an den Anfang der ersten Zelle der Tabelle.*

```
ThisComponent.CurrentController.select(oTable)
```

Listing 391 ist zwar nicht die ganze Lösung, aber doch ein Anfang, denn der Viewcursor steht an einer bekannten Position. Paolo liefert dann eine äußerst prägnante Methode, die ganze Tabelle zu selektieren, s. Listing 392.

Listing 392. *Wählt die gesamte Tabelle im aktuellen View aus.*

```
ThisComponent.CurrentController.select(oTable)
oVCursor.gotoEnd(True)   'Bewegt den Cursor an das Ende der aktuellen Zelle.
oVCursor.gotoEnd(True)   'Bewegt den Cursor an das Ende der Tabelle.
```

Tipp Testen Sie bitte sorgfältig jeden Code, der mit Tabellen zu tun hat. Paolo hat auch eine andere Methode vorgeschlagen – die aber nicht zuverlässig funktionierte –, nämlich auf der Basis von Zeilen und Spalten goRight() und dann goDown() zu verwenden.

Das Makro im Listing 393 wählt eine Tabelle mit Namen aus, kopiert sie in die Zwischenablage und fügt sie am Ende des Dokuments wieder ein.

Listing 393. *Kopiert eine Texttabelle über die Zwischenablage.*

```
Sub CopyNamedTableToEndWithClipboard(sName As String)
  Dim oTable        'Die zu kopierende Tabelle
  Dim oText         'Das Textobjekt des Dokuments
  Dim oFrame        'Der aktuelle Frame für den Dispatcher
  Dim oVCursor      'Der aktuelle Viewcursor
  Dim oDispatcher   'Dispatcher für Zwischenablage-Befehle
  oVCursor = ThisComponent.CurrentController.getViewCursor()
  oText = ThisComponent.getText()
  oFrame = ThisComponent.CurrentController.Frame
  oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")

  If Not ThisComponent.getTextTables().hasByName(sName) Then
    MsgBox "Oh, das Dokument enthält nicht die Tabelle " & sName
    Exit Sub
  End If

  oTable = ThisComponent.getTextTables().getByName(sName)

  REM Platziert den Cursor an den Anfang der ersten Zelle.
  REM So simpel!
  ThisComponent.CurrentController.select(oTable)
  oVCursor.gotoEnd(True)   'Bewegt den Cursor an das Ende der aktuellen Zelle.
  oVCursor.gotoEnd(True)   'Bewegt den Cursor an das Ende der Tabelle.

  REM Kopiert die Tabelle in die Zwischenablage.
  oDispatcher.executeDispatch(oFrame, ".uno:Copy", "", 0, Array())

  REM Setzt den Cursor an das Endes des Dokuments und fügt die Tabelle ein.
  oVCursor.gotoRange(oText.getEnd(), False)
  oDispatcher.executeDispatch(oFrame, ".uno:Paste", "", 0, Array())
End Sub
```

Die Zwischenablage wird von allen Anwendungen genutzt. Es könnte daher eine andere Anwendung die Zwischenablage ändern, während das Makro läuft. Der aktuelle Controller bietet den Zugriff auf übertragbaren Content ohne Nutzung der Zwischenablage.

Listing 394. *Kopiert eine Texttabelle als übertragbaren Content.*

```
Sub CopyNamedTableToEndUsingTransferable(sName As String)
  Dim oTable         'Die zu kopierende Tabelle
  Dim oText          'Das Textobjekt des Dokuments
  Dim oVCursor       'Der aktuelle Viewcursor
  Dim o              'Übertragbarer Content der aktuellen Auswahl

  oVCursor = ThisComponent.CurrentController.getViewCursor()
  oText = ThisComponent.getText()

  If Not ThisComponent.getTextTables().hasByName(sName) Then
    MsgBox "Oh, das Dokument enthält nicht die Tabelle " & sName
    Exit Sub
  End If

  oTable = ThisComponent.getTextTables().getByName(sName)

  REM Platziert den Cursor an den Anfang der ersten Zelle.
  REM So simpel!
  ThisComponent.CurrentController.select(oTable)
  oVCursor.gotoEnd(True)  'Bewegt den Cursor an das Ende der aktuellen Zelle.
  oVCursor.gotoEnd(True)  'Bewegt den Cursor an das Ende der Tabelle.
  o = ThisComponent.CurrentController.getTransferable()

  REM Setzt den Cursor an das Endes des Dokuments und fügt die Tabelle ein.
  oVCursor.gotoRange(oText.getEnd(), False)
  ThisComponent.CurrentController.insertTransferable(o)
End Sub
```

14.9.6. Formatierung einer Texttabelle

Ich formatiere Texttabellen mit Absatzvorlagen, wechsele den Hintergrund in jeder Zeile und ändere die Zellumrandungen.

Listing 395. *Formatiert eine Texttabelle.*

```
Sub FormatTable(Optional oUseTable)
  Dim oTable           'Die zu bearbeitende Tabelle
  Dim oCell            'Die jeweils zu bearbeitende Zelle
  Dim nRow As Long     'Zeilenindex
  Dim nCol As Long     'Spaltenindex

  If IsMissing(oUseTable) Then
    'Ohne Angabe einer Tabelle wird die Tabelle genommen, in der der Cursor steht.
    oTable = ThisComponent.CurrentController.getViewCursor().TextTable
  Else
    oTable = oUseTable
  End If
  If IsNull(oTable) Or IsEmpty(oTable) Then
    Print "FormatTable: Keine Tabelle angegeben"
    Exit Sub
  End If

  REM Falls Ihnen die folgenden Anweisungen zu umständlich erscheinen sollten,
  REM denken Sie daran, dass Structs mit ihrem Wert und nicht als Referenz kopiert
  REM werden.
  Dim v
  Dim x
  v = oTable.TableBorder  'Struct Tabellenrahmen (Kopie)
```

```
  x = v.TopLine           'Struct Obere Rahmenlinie (Kopie)
  x.OuterLineWidth = 2    'Äußere Linienstärke (InnerLineWidth = 0, also nur eine Linie)
  v.TopLine = x           'Obere Rahmenlinie geändert zurückkopiert

  x = v.LeftLine          'Ebenso mit der linken Rahmenlinie
  x.OuterLineWidth = 2
  v.LeftLine = x

  x = v.RightLine         'Dto rechte Rahmenlinie
  x.OuterLineWidth = 2
  v.RightLine = x

  x = v.BottomLine        'Dto untere Rahmenlinie
  x.OuterLineWidth = 2
  v.BottomLine = x

  x = v.VerticalLine      'Dto senkrechte Zelltrennlinien
  x.OuterLineWidth = 2
  v.VerticalLine = x

  x = v.HorizontalLine    'Keine waagerechten Zelltrennlinien
  x.OuterLineWidth = 0
  v.HorizontalLine = x

  oTable.TableBorder = v 'Tabellenrahmen geändert zurückkopiert

  'Jetzt wird jede Zelle, Zeile für Zeile, Spalte für Spalte, formatiert.
  For nRow = 0 To oTable.getRows().getCount() - 1
    For nCol = 0 To oTable.getColumns().getCount() - 1
      oCell = oTable.getCellByPosition(nCol, nRow)
      If nRow = 0 Then
        'Tabellenkopf
        oCell.BackColor = 128
        SetParStyle(oCell.getText(), "OOoTableHeader")  'Absatzvorlage
      Else
        'Tabelleninhalt
        SetParStyle(oCell.getText(), "OOoTableText")    'Absatzvorlage
        'Hintergrundfarbe abwechselnd weiß und grau
        If nRow Mod 2 = 1 Then
          oCell.BackColor = -1
        Else
          REM color is (230, 230, 230)
          oCell.BackColor = 15132390
        End If
      End If
    Next
  Next
End Sub
```

Jede Zelle hat ihr eigenes Textobjekt. Das folgende Makro verknüpft jeden Absatz in einem Textobjekt mit derselben Absatzvorlage.

Listing 396. *Legt die Absatzvorlage für jeden Absatz eines Textobjekts fest.*

```
Sub SetParStyle(oText, sParStyle As String)
  Dim oEnum  'Enumeration
  Dim oPar   'Absatz
```

```
    oEnum = oText.createEnumeration()
    Do While oEnum.hasMoreElements()
      oPar = oEnum.nextElement()
      If oPar.supportsService("com.sun.star.text.Paragraph") Then
        'oPar.ParaConditionalStyleName = sParStyle
        oPar.ParaStyleName = sParStyle
      End If
    Loop
  End Sub
```

14.10. Textfelder

Ein Textfeld ist Textcontent, der sich im Normalfall übergangslos in den laufenden Text einpasst, den aktuellen Inhalt aber woanders herholt – zum Beispiel die Gesamtzahl der Seiten oder ein Datenbankfeld. Die Tabelle 158 listet die Standard-Feldtypen auf.

Tabelle 158. *Textfeld-Services, die mit com.sun.star.text.TextField starten.*

Feldtyp	Beschreibung
Annotation	Eingefügter Kommentar mit den String-Eigenschaften Author und Content. Die Eigenschaft Date,Typ com.sun.star.util.Date, enthält das Erstellungsdatum des Kommentars.
Author	Autor des Dokuments. Mit folgenden optionalen Eigenschaften: • IsFixed – Falls False, wird der Autor bei jedem Speichern des Dokuments angepasst. • Content – Der Inhalt des Textfeldes als String. • AuthorFormat – Wert aus der Konstantengruppe com.sun.star.text.AuthorDisplayFormat: FULL (0), LAST_NAME (1), FIRST_NAME (2) oder INITIALS (3). • CurrentPresentation – Der aktuelle Text des Feldes als String. Vor allem nützlich für Import und Export. • FullName – Falls False, werden statt des vollen Namens nur die Initialen angezeigt.
Bibliography	Enthält die Eigenschaft Fields, ein PropertyValue-Array. Dieses Feld hängt vom Textmasterfeld Bibliography ab.
Chapter	Information über das Kapitel mit folgenden Eigenschaften: • Level (Byte Integer) – Bestimmt die Hierarchieebene (0 = oberste Ebene). • ChapterFormat – Art der Darstellung, aus der Konstantengruppe com.sun.star.text.ChapterFormat: NAME (0), NUMBER (1), NAME_NUMBER (2), NO_PREFIX_SUFFIX (3) oder DIGIT (4).
CharacterCount	Anzahl der Zeichen im Dokument. Es gibt nur eine Eigenschaft zum Zahlenformat: NumberingType, aus der Konstantengruppe com.sun.star.style.NumberingType. Die verfügbaren Werte zeigt die Tabelle 159.
CombinedCharacters	Kombination von 1 bis zu 6 Zeichen, die zweizeilig als 1 Zeichen dargestellt werden.
ConditionalText	Text, dessen Wortlaut von einer Bedingung im Textfeld abhängt. • TrueContent – String, falls die Bedingung True ist. • FalseContent – String, falls die Bedingung False ist. • Condition – Bedingung als String. • IsConditionTrue – Boolesches Ergebnis der Bedingung (nur lesbar).
DDE	Ergebnis einer DDE-Verbindung. Nutzt das Textmasterfeld DDE.
Database	Datenbanktextfeld als Feld für Serienbriefe. Dieses Feld ist abhängig von einem Textmasterfeld und hat die folgenden Eigenschaften: • Content – Aus der Datenbank eingefügter Text als String. • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • DataBaseFormat – Falls True, wird das Zahlenformat der Datenbank genutzt. • NumberFormat – Das Zahlenformat des Feldes: com.sun.star.util.NumberFormatter.

Feldtyp	Beschreibung
DatabaseName	Name der Datenbank bei Datenbankoperationen. Dieses Feld ist abhängig von einem Textmasterfeld und hat die folgenden Eigenschaften: • DataBaseName – Name der Datenbank als String. • DataCommandType – Bedeutung von DataTableName aus der Konstantengruppe com.sun.star.sdb.CommandType: TABLE (0), QUERY(1) oder COMMAND (2). • DataTableName – String mit entweder dem Tabellennamen, dem Abfragenamen oder dem SQL-Befehl.
DatabaseNextSet	Nächster Datensatz einer Auswahl. Dieses Feld ist abhängig von einem Textmasterfeld und hat die folgenden Eigenschaften: • DataBaseName – Name der Datenbank als String. • DataCommandType – Bedeutung von DataTableName aus der Konstantengruppe com.sun.star.sdb.CommandType: TABLE (0), QUERY(1) oder COMMAND (2). • DataTableName – String mit entweder dem Tabellennamen, dem Abfragenamen oder dem SQL-Befehl. • Condition – Bedingung (String), ob die nächste Position ausgewählt wird.
DatabaseNumberOfSet	Nummer des aktuellen Datenbanksatzes. Dieses Feld ist abhängig von einem Textmasterfeld und hat die folgenden Eigenschaften: • DataBaseName – Name der Datenbank als String. • DataCommandType – Bedeutung von DataTableName aus der Konstantengruppe com.sun.star.sdb.CommandType: TABLE (0), QUERY(1) oder COMMAND (2). • DataTableName – String mit entweder dem Tabellennamen, dem Abfragenamen oder dem SQL-Befehl. • NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount). • SetNumber – Nummer des Datensatzes (Long Integer).
DatabaseSetNumber	Legt die Nummer des Datenbanksatzes fest. Dieses Feld ist abhängig von einem Textmasterfeld und hat die folgenden Eigenschaften: • DataBaseName – Name der Datenbank als String. • DataCommandType – Bedeutung von DataTableName aus der Konstantengruppe com.sun.star.sdb.CommandType: TABLE (0), QUERY(1) oder COMMAND (2). • DataTableName – String mit entweder dem Tabellennamen, dem Abfragenamen oder dem SQL-Befehl. • Condition – Bedingung (String), ob SetNumber angewendet wird. • SetNumber – Nummer des anzuwendenden Datensatzes (Long Integer).
DateTime	Datum oder Uhrzeit mit den folgenden optionalen Eigenschaften: • IsFixed – Falls False, wird das aktuelle Datum oder die aktuelle Uhrzeit angezeigt. • IsDate – Falls False, ist es nur eine Uhrzeit. Falls True, ist es ein Datum mit optionaler Uhrzeit. • DateTimeValue – Der Feldinhalt als com.sun.star.util.DateTime-Objekt. • NumberFormat – Format des Feldes als com.sun.star.util.NumberFormatter. • Adjust – Zeitversatz in Minuten (Long Integer). • IsFixedLanguage – Falls False, kann die Änderung der Sprache des umgebenden Textbereichs eine Änderung der Felddarstellung bewirken.
DropDown	Aufklappliste mit den folgenden Eigenschaften: • Name – Feldname. • Items – Stringarray mit den Aufklappwerten. • SelectedItem – Der ausgewählte Eintrag (String) oder ein leerer String, falls nichts ausgewählt ist.

Feldtyp	Beschreibung
EmbeddedObjectCount	Anzahl der im Dokument eingebetteten Objekte, mit der Eigenschaft NumberFormat, Zahlenformat (s. Feldbeschreibung CharacterCount).
ExtendedUser	Benutzerdaten (aus Extras | Optionen | LibreOffice | Benutzerdaten) wie Name, Adresse oder Telefonnummer. • Content – Der Inhalt als String. • CurrentPresentation – Der aktuelle Text des Feldes als String. • IsFixed – Falls False, wird der Inhalt aktualisiert. • UserDataType – Der auszugebende Teil der Benutzerdaten als Wert der Konstantengruppe com.sun.star.text.UserDataPart: COMPANY, FIRSTNAME, NAME, SHORTCUT, STREET, COUNTRY, ZIP, CITY, TITLE, POSITION, PHONE_PRIVATE, PHONE_COMPANY, FAX, EMAIL, STATE.
FileName	Dateiname des Dokuments (URL) mit folgenden Eigenschaften: • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • FileFormat – Format des Dateinamens als Wert der Konstantengruppe com.sun.star.text.FilenameDisplayFormat: FULL, PATH, NAME und NAME_AND_EXT. • IsFixed – Falls False, wird der Inhalt aktualisiert.
GetExpression	Gibt das Ergebnis eines SetExpression-Textfeldes aus. (GUI-Entsprechung: Feldbefehl, Register Querverweise). Hat folgende Eigenschaften: • Content – Der Name des Masterfeldes des SetExpression-Textfeldes als String. • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • NumberFormat – Format des Feldes als com.sun.star.util.NumberFormatter. • NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount). • IsShowFormula – Falls True, wird statt des Inhalts die Formel dargestellt. • SubType – Typ der Variablen aus der Konstantengruppe com.sun.star.text.SetVariableType mit folgenden Werten: VAR (Variable), SEQUENCE (Nummernkreis), FORMULA (Formel) und STRING. • Value – Numerischer Wert des Feldes als Double. • IsFixedLanguage – Falls False, kann die Änderung der Sprache des umgebenden Textbereichs eine Änderung der Felddarstellung bewirken. *Ein GetExpression-Feld referenziert ein SetExpression-Feld. Wenn Sie das zugehörige SetExpression-Feld löschen, wird Ihnen von CurrentPresentation ein Fehler etwa wie „Invalid reference“ präsentiert. Ich kenne aber keinen anderen Weg festzustellen, dass der Querverweis nicht mehr gültig ist.*
GetReference	Querverweis mit folgenden Eigenschaften: • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • ReferenceFieldSource – Aus der Konstantengruppe com.sun.star.text.ReferenceFieldSource mit folgenden Werten: REFERENCE_MARK, SEQUENCE_FIELD, BOOKMARK, FOOTNOTE und ENDNOTE. • SourceName – Referenzname als String, zum Beispiel der Name einer Textmarke. • ReferenceFieldPart – Aus der Konstantengruppe com.sun.star.text.ReferenceFieldPart mit folgenden Werten: PAGE, CHAPTER, TEXT, UP_DOWN, PAGE_DESC, CATEGORY_AND_NUMBER, ONLY_CAPTION und ONLY_SEQUENCE_NUMBER. • SequenceNumber – Nummerierung (Short Integer) für einen Nummernkreis oder als ReferenceId-Eigenschaft einer Fußnote oder Endnote.
GraphicObjectCount	Anzahl der im Dokument eingebetteten grafischen Objekte, mit der Eigenschaft NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount).

Feldtyp	Beschreibung
HiddenParagraph	Verbirgt einen Absatz. Damit kann man zum Beispiel einen Test mit Fragen und Antworten in einem Dokument erstellen: Die Antworten werden für den Ausdruck für die Studenten auf verborgen gesetzt. • Condition – Auszuwertende Bedingung als String. • IsHidden – Boolesches Ergebnis der letzten Auswertung der Bedingung.
HiddenText	Verborgener Text. Unterscheidet sich vom verborgenen Absatz darin, dass nur der Text des Feldes verborgen ist und nicht der gesamte Absatz. • Content – Textinhalt als String. • Condition – Auszuwertende Bedingung als String. • IsHidden – Boolesches Ergebnis der letzten Auswertung der Bedingung.
Input	Texteingabefeld. • Content – Textinhalt als String. • Hint – Hinweistext als String.
InputUser	Benutzerdefiniertes Textfeld, das von einem Masterfeld abhängt. • Content – Textinhalt als String. • Hint – Hinweistext als String.
JumpEdit	Platzhalter. • Hint – Hinweistext als String. • PlaceHolder – Text des Platzhalters als String. • PlaceHolderType – Aus der Konstantengruppe com.sun.star.text.PlaceholderType mit den Werten: TEXT, TABLE, TEXTFRAME, GRAPHIC oder OBJECT.
Macro	Makrotextfeld. • Hint – Hinweistext als String. • MacroName – Name des auszuführenden Makros als String. • MacroLibrary – Name der das Makro enthaltenen Bibliothek als String.
PageCount	Anzahl der Seiten im Dokument, mit der Eigenschaft NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount).
PageNumber	Aktuelle Seitenzahl. • Offset – Korrekturwert als Short Integer. • SubType – Bezug zur vorherigen, aktuellen oder folgenden Seite als Wert der Enumeration com.sun.star.text.PageNumberType: PREV, CURRENT oder NEXT. • UserText – Ausgabestring, wenn NumberingType gleich CHAR_SPECIAL ist. • NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount).
ParagraphCount	Anzahl der Absätze im Dokument, mit der Eigenschaft NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount).
ReferencePageGet	Seitenvariable anzeigen, mit der Eigenschaft NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount).
ReferencePageSet	Seitenvariable setzen. Mit folgenden Eigenschaften: • Offset – Korrekturwert, der die Anzeige eines ReferencePageGet-Feldes ändert (Short Integer). • NameOn – Falls True, werden ReferencePageGet-Felder angezeigt.
Script	Text, der aus einem Skript stammt. Mit folgenden Eigenschaften: • Content – Der Text selbst oder der URL des Skripts (String). • ScriptType – Der Skripttyp, zum Beispiel JavaScript (String). • URLContent – True: Content ist ein URL. False: Content ist der Skripttext

Feldtyp	Beschreibung
SetExpression	Textfeld eines Wert-„Ausdrucks“. (GUI-Entsprechung: Feldbefehl, Register Variablen). Benötigt ein SetExpression-Masterfeld. Hat folgende Eigenschaften: • Content – Textinhalt als String. • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • NumberFormat – Format des Feldes als com.sun.star.util.NumberFormatter. • NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount). • IsShowFormula – Falls True, wird statt des Inhalts die Formel dargestellt. • Hint – Hinweistext als String, falls es ein Eingabefeld ist. • IsInput – Falls True, ist es ein Eingabefeld. • IsVisible – Falls True, ist das Feld sichtbar. • SequenceValue – Nummer, falls es ein Nummernkreisfeld ist. • SubType – Typ der Variablen aus der Konstantengruppe com.sun.star.text.SetVariableType mit folgenden Werten: VAR (Variable), SEQUENCE (Nummernkreis), FORMULA (Formel) und STRING. • Value – Numerischer Wert des Feldes als Double. • VariableName – Name des verknüpften SetExpression-Masterfeldes. • IsFixedLanguage – Falls False, kann die Änderung der Sprache des umgebenden Textbereichs eine Änderung der Felddarstellung bewirken.
TableCount	Anzahl der Tabellen im Dokument, mit der Eigenschaft NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount).
TemplateName	Der Name der dem Dokument zugrundeliegenden Dokumentvorlage, mit der Eigenschaft FileFormat (s. Textfeld Filename).
URL	URL-Darstellung. Mit folgenden Eigenschaften: • Format – URL-Ausgabezeichenformat (Short Integer). • URL – Der ungeparste originale URL-String. • Representation – Ausgabestring für den Benutzer. • TargetFrame – Name des Frames (String), in dem der URL geöffnet wird.
User	Benutzerdefiniertes Feld mit einem Masterfeld. Mit folgenden Eigenschaften: • IsShowFormula – Falls True, wird statt des Inhalts die Formel dargestellt. • IsVisible – Falls True, ist das Feld sichtbar. • NumberFormat – Format des Feldes als com.sun.star.util.NumberFormatter. • IsFixedLanguage – Falls False, kann die Änderung der Sprache des umgebenden Textbereichs eine Änderung der Felddarstellung bewirken.
WordCount	Anzahl der Wörter im Dokument, mit der Eigenschaft NumberingType – Zahlenformat (s. Feldbeschreibung CharacterCount).
docinfo.ChangeAuthor	Der Name des Autors, der als letzter das Dokument bearbeitet hat. • Author – Der Name als String. • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • IsFixed – Falls False, wird der Inhalt beim Speichern des Dokuments aktualisiert.

Feldtyp	Beschreibung
docinfo.ChangeDateTime	Datum und Uhrzeit der letzten Änderung des Dokuments. Mit folgenden Eigenschaften: • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • IsFixed – Falls False, wird das aktuelle Datum oder die aktuelle Uhrzeit angezeigt. • IsDate – Falls False, ist es nur eine Uhrzeit. Falls True, ist es ein Datum mit optionaler Uhrzeit. • DateTimeValue – Der Feldinhalt als com.sun.star.util.DateTime-Objekt. • NumberFormat – Format des Feldes als com.sun.star.util.NumberFormatter. • IsFixedLanguage – Falls False, kann die Änderung der Sprache des umgebenden Textbereichs eine Änderung der Felddarstellung bewirken.
docinfo.CreateAuthor	Name des Autors, der das Dokument erstellt hat (Eigenschaften s. docinfo.ChangeAuthor).
docinfo.CreateDateTime	Datum/Uhrzeit der Dokumenterstellung (Eigenschaften s. docinfo.ChangeDateTime).
docinfo.Custom	Inhalt eines benutzerdefinierten Feldes in den Dokumenteigenschaften (Eigenschaften s. docinfo.Description).
docinfo.Description	Dokumentbeschreibung aus den Dokumenteigenschaften (Datei | Eigenschaften), mit folgenden Eigenschaften: • Content – Textinhalt als String. • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • IsFixed – Falls False, wird der Inhalt beim Speichern des Dokuments aktualisiert.
docinfo.EditTime	Gesamtbearbeitungszeit des Dokuments, mit folgenden Eigenschaften: • CurrentPresentation – Der aktuelle Inhalt des Feldes als String. • IsFixed – Falls False, wird das aktuelle Datum oder die aktuelle Uhrzeit angezeigt. • DateTimeValue – Datum und Uhrzeit als Double. • NumberFormat – Format des Feldes als com.sun.star.util.NumberFormatter. • IsFixedLanguage – Falls False, kann die Änderung der Sprache des umgebenden Textbereichs eine Änderung der Felddarstellung bewirken.
docinfo.Info0	Dokument-Info 0 (Eigenschaften s. docinfo.Description). Veraltet ab OOo 3.0.
docinfo.Info1	Dokument-Info 1 (Eigenschaften s. docinfo.Description). Veraltet ab OOo 3.0.
docinfo.Info2	Dokument-Info 2 (Eigenschaften s. docinfo.Description). Veraltet ab OOo 3.0.
docinfo.Info3	Dokument-Info 3 (Eigenschaften s. docinfo.Description). Veraltet ab OOo 3.0.
docinfo.Keywords	Schlüsselwörter aus den Dokumenteigenschaften (Eigenschaften s. docinfo.Description).
docinfo.PrintAuthor	Name des Autors, der das Dokument gedruckt hat (Eigenschaften s. docinfo.ChangeAuthor).
docinfo.PrintDateTime	Datum und Uhrzeit des letzten Dokumentdrucks (Eigenschaften s. docinfo.ChangeDateTime).
docinfo.Revision	Aktuelle Dokumentversion (Eigenschaften s. docinfo.Description).
docinfo.Subject	Dokumentthema aus den Dokumenteigenschaften (Eigenschaften s. docinfo.Description).
docinfo.Title	Dokumenttitel aus den Dokumenteigenschaften (Eigenschaften s. docinfo.Description).

Tipp Das Feld Annotation ist ein Service vom Typ com.sun.star.text.TextField.Annotation. Manche Dokumentationsquellen enthalten den Text „textfield“ komplett in Kleinbuchstaben. Das ist falsch. Der Code im Listing 397 zeigt die korrekte Form.

Tabelle 159 enthält eine Auswahl der gültigen Werte für die Eigenschaften, die in der Tabelle 158 auf ...Count enden, beispielsweise CharacterCount. Die Konstantengruppe com.sun.star.style.NumberingType umfasst momentan 56 Konstanten, die einen breiten Rahmen von Zahlen und Schriftzeichen verschiedener Sprachen und Schriftsysteme bilden, angefangen mit lateinischer Schrift und arabischen sowie römischen Zahlen über Sprachen mit kyrillischer Schrift, hebräisch, griechisch, arabisch bis hin zu asiatischen Sprachen und Schriften. In der Tabelle 159 sind nur die ersten zehn Konstanten aufgeführt, die für die Leser dieser deutschen Übersetzung wohl am meisten Gewicht haben. Die Gesamtliste finden Sie in der API-Referenz:
AOO: http://www.openoffice.org/api/docs/common/ref/com/sun/star/style/NumberingType.html
LO: https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1style_1_1NumberingType.html

Tabelle 159. *Die Konstantengruppe com.sun.star.style.NumberingType.*

Wert	Konstante	Beschreibung
0	CHARS_UPPER_LETTER	Zählung in Großbuchstaben: „A, B, C, D, ...“.
1	CHARS_LOWER_LETTER	Zählung in Kleinbuchstaben: „a, b, c, d, ...“.
2	ROMAN_UPPER	Zählung in Römischen Zahlen als Großbuchstaben: „I, II, III, IV, ...“.
3	ROMAN_LOWER	Zählung in Römischen Zahlen als Kleinbuchstaben: „i, ii, iii, iv, ...“.
4	ARABIC	Zählung in Arabischen Zahlen: „1, 2, 3, 4, ...“.
5	NUMBER_NONE	Zählung ist unsichtbar.
6	CHAR_SPECIAL	Ein Zeichen aus einer bestimmten Schriftart.
7	PAGE_DESCRIPTOR	Zählung wie in der Seitenvorlage festgelegt.
8	BITMAP	Zählung als Bitmap-Grafik.
9	CHARS_UPPER_LETTER_N	Zählung in Großbuchstaben: „A, B, ..., Y, Z, AA, BB, CC, ... AAA, ...“.
10	CHARS_LOWER_LETTER_N	Zählung in Kleinbuchstaben: „a, b, ..., y, z, aa, bb, cc, ... aaa, ...“.

Mit der Objektmethode getTextFields() erhält man die im Dokument enthaltenen Textfelder, s. Tabelle 147. Jedes Textfeldobjekt unterstützt die Objektmethode getPresentation(boolean), die einen String zurückgibt, der abhängig vom booleschen Argument entweder den Feldtyp (True) oder den Text der Darstellung (False) enthält (s. Listing 397 und Bild 101).

Listing 397. *Ausgabe der Textfelder.*

```
Sub DisplayFields
  Dim oEnum     'Enumeration der Textfelder
  Dim oField    'Ein einzelnes Textfeld
  Dim s$        'Ausgabestring
  oEnum = ThisComponent.getTextFields().createEnumeration()
  Do While oEnum.hasMoreElements()
    oField = oEnum.nextElement()
    s = s & oField.getPresentation(True)  & " = "    'Feldtyp
    If oField.supportsService("com.sun.star.text.TextField.Annotation") Then
      REM Etwas kryptischer wäre: If oField.getPresentation(True) = "Note" ...
      REM Der Typ "Note" hat keinen Ausgabeinhalt, getPresentation(False) würde einen
      REM leeren String ergeben. Stattdessen wird "Author" und "Content" ausgegeben.
      s = s & oField.Author & " schreibt " & oField.Content
    Else
      s = s & oField.getPresentation(False)          'Feldinhalt als String
    End If
    s = s & Chr$(13)
  Loop
  MsgBox s, 0, "Textfelder"
End Sub
```

Bild 101. *Textfelder in einem Dokument.*

Das Dokument, das die Quelle für Bild 101 ist, enthält ein Datumsfeld (DateTime), ein Seitennummernfeld (PageNumber), ein Kommentarfeld (Annotation) und ein Benutzerfeld (User). Der Code im Listing 397 hat eine Sonderbehandlung für das Kommentarfeld zur Ausgabe des Autors und des Kommentartextes. Das Feld wird mit der Methode supportsService(Object) daraufhin überprüft, ob es den Service Annotation unterstützt.

Die normale Methode, ein bestimmtes Textfeld zu suchen, ist die Enumeration der Textfelder, s. Listing 397. Doch wenn das Dokument sehr groß ist und eine Menge Textfelder enthält, kann es lang dauern, bis das gesuchte Textfeld gefunden ist. Wenn Sie aber wissen, an welcher Stelle sich das Textfeld im Dokument befindet, verwenden Sie die Methode des Listing 366 und enumerieren den Textcontent eines Absatzes.

Tipp	Textfelder setzen die Objektmethode update() ein. Diese Methode bewirkt, dass sich ein Textfeld mit den aktuellsten verfügbaren Informationen aktualisiert. Zum Beispiel aktualisieren sich all die Textfelder Datum/Uhrzeit, Dateiname und Dokumentinfo auf die aktuellsten Inhalte.

14.10.1. Textmasterfelder

Manche Textfelder haben ihren eigenen Inhalt, andere wiederum beziehen sich auf eine externe Quelle für die dargestellten Informationen. Diese externe Quelle heißt Masterfeld. Tabelle 160 listet die Textfeldtypen auf, die ein Masterfeld benötigen. Außer den in Tabelle 160 genannten Eigenschaften unterstützen Masterfelder auch die Eigenschaften der Tabelle 161.

Achtung	Die Objektmethode getTextFieldMaster() gibt das Masterfeld eines Textfeldes zurück. Leider setzen alle Felder – auch die, die kein Masterfeld benötigen – diese Methode ein und geben ein Masterfeld zurück, das nicht Null ist. Achtung, OOo könnte abstürzen, wenn Sie ein Masterfeld von einem Feld abrufen, das gar keines unterstützt, und es dann manipulieren.

Obwohl Textfelder nur durch Enumeration verfügbar sind, kann man auf Masterfelder sowohl über ihre Namen als auch über Enumeration zugreifen, s. Tabelle 147. Den Namen eines Masterfeldes erhalten Sie, wenn Sie den Feldnamen mit Punkt an den Masterfeldtyp anhängen. Zum Beispiel hat das Benutzerfeld „Volkerfeld“ im Bild 101 den Masterfeldnamen com.sun.star.text.FieldMaster.User.Volkerfeld. Datenbankmasterfelder werden anders als die anderen Masterfelder benannt. Bei ihnen werden der Name der Datenbank, der Name der Datentabelle und der Name der Datenspalte an den Servicenamen angefügt. Listing 398 zeigt, wie man auf die Textmasterfelder eines Dokuments zugreift. Das Ergebnis sehen Sie im Bild 102.

Tabelle 160. *Textfeld-Services, die mit com.sun.star.text.FieldMaster starten.*

Feldtyp	Beschreibung
Bibliography	Masterfeld für ein Bibliography-Textfeld. Mit folgenden Eigenschaften: • IsNumberEntries – Falls True, sind die Felder nummeriert, ansonsten wird der Kurzname des Eintrags verwendet. • IsSortByPosition – Falls True, wird der Bibliography-Index anhand der Position sortiert (s. SortKeys). • BracketBefore – Die öffnende Klammer im Bibliography-Textfeld. • BracketAfter – Die schließende Klammer im Bibliography-Textfeld. • SortKeys – Dieses PropertyValue-Array wird zur Sortierung verwendet, wenn IsSortByPosition False ist. Es besteht aus den beiden Properties SortKey (das Feld wird über die Konstantengruppe com.sun.star.text.BibliographyDataField bestimmt) und IsSortAscending (boolescher Wert für auf- oder absteigende Sortierung). • Locale – com.sun.star.lang.Locale des Masterfeldes. • SortAlgorithm – Name (String) des Sortieralgorithmus zum Sortieren der Textfelder.
DDE	Masterfeld für ein DDE-Textfeld. Mit folgenden Eigenschaften: • DDECommandElement – DDE-Befehl als String. • DDECommandFile – Datei, die den DDE-Befehl enthält, als String. • DDECommandType – DDE-Befehlstyp als String. • IsAutomaticUpdate – Falls True, wird der DDE-Link automatisch aktualisiert.
Database	Masterfeld für ein Datenbanktextfeld. Mit folgenden Eigenschaften: • DataBaseName – Name (String) der Datenquelle. • CommandType – Befehlstyp als Long Integer (0 = Tabelle, 1 = Abfrage, 2 = Anweisung). • DataTableName – Befehlsstring. Entweder der Name einer Datenbanktabelle oder eine Datenbankabfrage oder eine Datenbankanweisung. In der Eigenschaft CommandType wird der passende Typ zugeordnet. • DataColumnName – Spaltenname (String) der Datenbanktabelle.
SetExpression	Masterfeld für ein SetExpression-Textfeld. Mit folgenden Eigenschaften: • ChapterNumberingLevel – Kapitelzählungsebene als Byte, falls es sich um eine Nummerierung handelt. • NumberingSeparator – Nummerierungstrennstring, falls es sich um eine Nummerierung handelt. • SubType – Typ der Variablen aus der Konstantengruppe com.sun.star.text.SetVariableType mit folgenden Werten: VAR (Variable), SEQUENCE (Nummernkreis), FORMULA (Formel) und STRING.
User	Masterfeld für ein Benutzer-Textfeld. Mit folgenden Eigenschaften: • IsExpression – Falls True, enthält das Feld einen Ausdruck. • Value – Numerischer Wert vom Typ Double. • Content – Feldinhalt als String.

Tabelle 161. *Eigenschaften im Service com.sun.star.text.FieldMaster.*

Eigenschaft	Beschreibung
Name	Optionaler String mit dem Feldnamen. Muss gesetzt werden, bevor das Feld ins Dokument eingefügt wird.
DependentTextFields	Array von Textfeldern, die dieses Masterfeld nutzen.
InstanceName	Name (String) der Instanz, wie er von XTextFieldsSupplier verwendet wird.

Listing 398. *Listet Textmasterfelder auf.*

```
Sub ShowFieldMasters
  Dim oMasters       'Alle Textmasterfelder
  Dim oMasterNames   'Array der Textmasterfeldnamen
  Dim i%, j%         'Indexvariablen
  Dim sMasterName$   'Voller Name des Masterfeldes
  Dim s$             'Ausgabestring
  Dim oMaster        'Masterfeld

  REM Zugriff auf die Kollektion der Textmasterfeld-Objekte.
  oMasters = ThisComponent.getTextFieldMasters()

  REM Holt ALLE Textmasterfeldnamen.
  REM Das ist ein Stringarray.
  oMasterNames = oMasters.getElementNames()
  For i = LBound(oMasterNames) To UBound(oMasterNames)

    REM Für jeden vorhandenen Namen wird das Masterfeld-Objekt referenziert,
    REM dessen Eigenschaft DependentTextFields abgerufen wird,
    REM die ein Array der Textfelder ist,
    REM die von diesem Masterfeld abhängen.
    sMasterName = oMasterNames(i)
    oMaster = oMasters.getByName(sMasterName)
    s = s & "***" & sMasterName & "***" & Chr$(10)
    s = s & oMaster.Name & " enthält " &_
        CStr(UBound(oMaster.DependentTextFields) + 1) &_
        " abhängige Felder" & Chr$(10)
    s = s & Chr$(13)
  Next i
  REM Der direkte Zugriff auf ein Masterfeld über seinen Namen.
  REM Dies hier ist ein Benutzerfeld, das ich in das Beispieldokument eingefügt habe.
  If oMasters.hasByName("com.sun.star.text.FieldMaster.User.Volkerfeld") Then
    oMaster = oMasters.getByName("com.sun.star.text.FieldMaster.User.Volkerfeld")
    s = s & "Direkter Zugriff auf das Masterfeld " & oMaster.Name & Chr$(10) &_
        "Das Feld enthält den Text " & oMaster.Content
  End If
  MsgBox s, 0, "Textmasterfelder"
End Sub
```

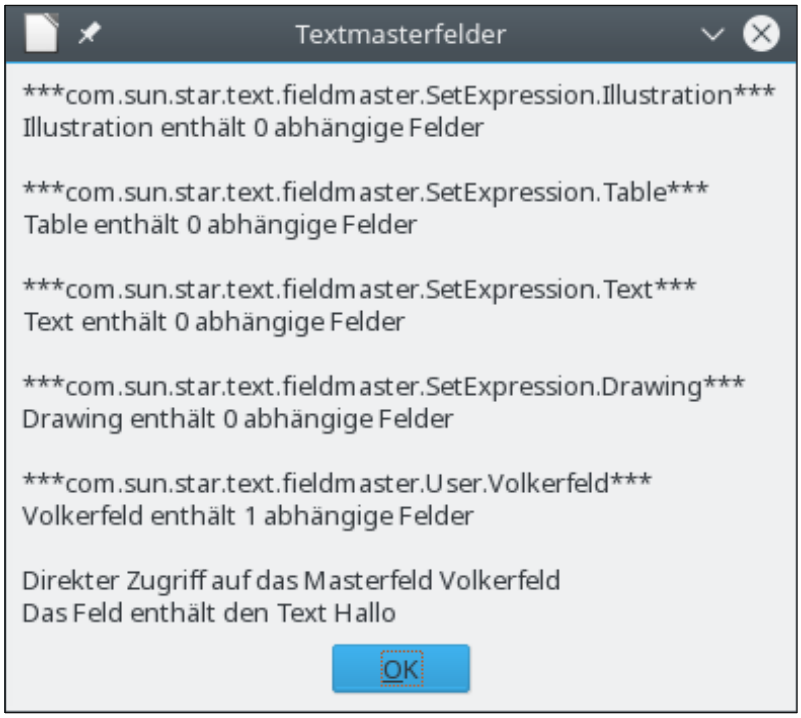


Bild 102. *Textmasterfelder in einem Dokument.*

14.10.2. Textfelder erzeugen und einfügen

Textfelder müssen von dem Dokument erzeugt werden, in das sie einfließen sollen. Wenn Sie sie also aus dem Dokument entfernen, dann wird das Dokument sie auch zerstören. Listing 399 zeigt, wie ein Datum/Uhrzeit-Textfeld (DateTime) und ein Kommentar-Textfeld (Annotation) erzeugt, konfiguriert und eingefügt wird. Die Textfelder werden ans Dokumentende angehängt. Das Textfeld DateTime wird in ungewöhnlicher Weise formatiert, so dass ein neues Zahlenformat gebildet wird, falls es noch nicht existiert.

Tipp	Vom Listing 399 wird die beliebte Routine FindCreateNumberFormatStyle (s. Listing 403) genutzt.

Was zum korrekten Einfügen eines Kommentarfelds getan werden muss, hat sich über die Zeit geändert und kann auch zwischen AOO und LO differieren. Mit OOo wird beim Einfügen eines Kommentars das Datum auf den aktuellen Wert gesetzt. In meinen neuesten Tests mit LO 4.0.1.2 wird gar kein Datum gesetzt. Es gibt dort zwar eine Eigenschaft Date, die aber offenbar ignoriert wird. Um das Datum zu setzen, müssen Sie stattdessen die Eigenschaft DateTimeValue setzen. Die Datum/Uhrzeit-Einfügung muss gesetzt sein, bevor das Textfeld eingefügt wird, der Wert kann danach nicht mehr korrigiert werden. Das Makro funktioniert auch mit AOO (jedenfalls in der Version 4.0).

Listing 399. *Einfügung von Textfeldern.*

```
Sub InsertFields
  Dim oText   'Textobjekt für das aktuelle Dokument
  Dim oField  'Einzufügendes Feld
  Dim oDoc    'oDoc ist schneller zu schreiben als ThisComponent

  oDoc = ThisComponent
  oText = oDoc.Text

  REM Zuerst wird ein Datum/Uhrzeit-Textfeld am Ende des Dokuments eingefügt.
  REM Es wird als "TT. MMM JJJJ" formatiert.
  REM Vor das neu eingefügte Feld wird ein kurzer Einleitungstext gesetzt.
  oText.insertString(oText.getEnd(), "Heute ist ", False)
```

```
    REM Erzeugt ein Datum/Uhrzeit-Feld.
    oField = oDoc.createInstance("com.sun.star.text.TextField.DateTime")
    oField.IsFixed = True
    oField.NumberFormat = FindCreateNumberFormatStyle("TT. MMM JJJJ", oDoc)
    oText.insertTextContent(oText.getEnd(), oField, False)

    REM Nun wird hinter das Textfeld ein Kommentar eingefügt.
    Dim oDate As New com.sun.star.util.Date
    REM Ich versuche, das Datum zu verfälschen: es wird so getan,
    REM als wäre es schon eine Zeit her, vor 10 Jahren, 10 Monaten und 10 Tagen!
    Dim oRightNow   'Gerade jetzt!
    oRightNow = Now
    With oDate
      .Day   = Day(oRightNow) - 10
      .Month = Month(oRightNow) - 10
      .Year  = Year(oRightNow) - 10
    End With

    Dim oDT As New com.sun.star.util.DateTime
    With oDT
      .Day   = Day(oRightNow) - 10
      .Month = Month(oRightNow) - 10
      .Year  = Year(oRightNow) - 10
      .Hours = 4
      .Minutes = 5
      .Seconds = 6
      'Für AOO:
      '.HundredthSeconds = 0
      'Für LO:
      .NanoSeconds = 0
    End With

    REM Wie fast jeder Textcontent muss das Feld von dem Dokument erzeugt werden,
    REM in dem es enthalten sein soll.
    REM Es passiert nichts, wenn man das Datumsfeld setzt.
    oField = oDoc.createInstance("com.sun.star.text.TextField.Annotation")
    With oField
      .Author  = "AP"
      .Content = "Dieser Kommentar ist neben einem gerade eingefügten Textfeld."
      .Date    = oDate
      .DateTimeValue = oDT
    End With
    oText.insertTextContent(oText.getEnd(), oField, False)
    MsgBox "Am Dokumentende wurden zwei Felder eingefügt."
End Sub
```

Ein Feld einzufügen, das ein Masterfeld benötigt, ist nur wenig schwieriger, als ein normales Textfeld einzufügen. Sowohl das Masterfeld als auch das abhängige Textfeld müssen vom Dokument mit der Objektmethode createInstance() erzeugt werden. Das Masterfeld muss vor der Benutzung benannt werden. Nachdem ein Feld in das Dokument eingefügt ist, kann man den Namen nicht mehr ändern. Das abhängige Feld wird mit dem Masterfeld verknüpft, das seinerseits den Inhalt dem abhängigen Feld zur Verfügung stellt. Das abhängige Feld wird als Textcontent in das Dokument eingefügt, nicht das Masterfeld. Das abhängige Feld kann mit der Objektmethode removeTextContent()

entfernt werden. Das Masterfeld hingegen wird mit der dispose()-Methode des Masterfeldes gelöscht.

Listing 400 demonstriert den Gebrauch von Masterfeldern anhand mehrerer Operationen an dem Masterfeld „TestFeld“. Je nach Situation stehen die folgenden drei verschiedenen Wege zur Verfügung:

- Wenn das Masterfeld nicht existiert, werden das Masterfeld und auch auch ein abhängiges Feld erzeugt, das in das Dokument eingefügt wird. Das Feld ist nun über das Menü **Einfügen | Feldbefehl | Andere** im Reiter Variablen sichtbar.
- Wenn das Masterfeld mit einem verknüpften abhängigen Feld existiert, wird das abhängige Feld aus dem Dokument gelöscht. Das Masterfeld bleibt erhalten, es gibt aber kein ins Dokument eingefügtes abhängiges Feld. Man kann das Masterfeld über den Feldbefehl-Dialog sehen.
- Wenn das Masterfeld existiert, aber kein abhängiges Feld verknüpft ist, wird das Masterfeld mit der Objektmethode dispose() gelöscht. Der Feldbefehl-Dialog zeigt das Masterfeld nicht mehr an.

Listing 400. *Nutzung eines Masterfeldes.*

```
Sub InsertFieldMaster
  Dim oMasters  'Liste aller Masterfelder.
  Dim oText     'Textobjekt für das aktuelle Dokument.
  Dim oUField   'Einzufügendes Benutzerfeld.
  Dim oMField   'Masterfeld für das Benutzerfeld.
  Dim oDoc      'oDoc ist schneller geschrieben als ThisComponent
  Dim sLead$    'Formaler Anfang des Feldnamens.
  Dim sName$    'Name des zu löschenden oder einzufügenden Feldes.
  Dim sTotName$ 'Der vollständige Name.

  REM Initialisierung der Namen.
  sName = "TestFeld"
  sLead = "com.sun.star.text.FieldMaster.User"
  sTotName = sLead & "." & sName

  REM Initialisierung einiger Werte.
  oDoc = ThisComponent
  oText = oDoc.Text
  oMasters = ThisComponent.getTextFieldMasters()

  REM Sonderbehandlung, falls das Masterfeld existiert.
  REM Die Sonderbehandlung ist rein illustrativ, sie bietet
  REM keinen besonderen Spaß und kein spannendes Problem.
  If oMasters.hasByName(sTotName) Then
    REM Zugriff auf das Masterfeld und auf die Liste der davon abhängigen Felder.
    oMField = oMasters.getByName(sTotName)

    REM Wenn es von diesem Feld abhängige Felder gibt, dann
    REM ist auch das Array der abhängigen Felder nicht leer!
    If UBound(oMField.DependentTextFields) >= 0 Then
      REM Der Textcontent wird gelöscht, damit verschwindet er auch
      REM aus dem Dokument. Das Masterfeld bleibt jedoch erhalten!
      oUField = oMField.DependentTextFields(0)
      oText.removeTextContent(oUField)
      MsgBox "Eine Instanz wurde aus dem Dokument gelöscht."
    Else
      REM Ich habe ganz willkürlich entschieden, das Masterfeld zu löschen.
      REM Ich könnte auch einfach ein neues Benutzerfeld erzeugen,
```

```
      REM es mit dem bestehenden Masterfeld verknüpfen und dann
      REM das neue Feld ins Dokument einfügen.
      oMField.content = ""
      oMField.dispose()
      MsgBox "Keine Instanzen im Dokument. Das Masterfeld wurde gelöscht."
    End If
  Else
    REM Erzeugt ein Benutzerfeld, das ein Masterfeld benötigt.
    oUField = oDoc.createInstance("com.sun.star.text.TextField.User")

    REM Nun wird das Masterfeld erzeugt.
    Dim oMasterField
    oMasterField = oDoc.createInstance(sLead)

    REM Den Namen eines Masterfeldes KANN MAN NICHT ÄNDERN, NACHDEM
    REM es Bestandteil des Dokuments ist, also muss es jetzt gemacht werden.
    oMasterField.Name = sName

    REM Und nun die Ausgabedaten. Beachten Sie, dass das Benutzerfeld
    REM nur ausgibt, was ihm der Master vorgibt.
    oMasterField.Content = "Hallo"

    REM Ein Benutzerfeld muss mit einem Masterfeld verknüpft sein.
    REM Danach ist es ein abhängiges Textfeld (DependentTextField).
    oUField.attachTextFieldMaster(oMasterField)

    REM Das Benutzerfeld wird in das Dokument eingefügt.
    oText.insertTextContent(oText.getEnd(), oUField, False)
    MsgBox "Am Ende des Dokuments wurde ein Feld eingefügt."
  End If
End Sub
```

14.11. Textmarken (Bookmarks)

Eine Textmarke ist Textcontent, auf den man über seinen Namen zugreifen kann. Eine Textmarke kann einen Einzelpunkt oder einen Textrange umfassen. Listing 383 fügt Textcontent an dem Punkt ein, an dem eine Textmarke verankert ist. Mit der Objektmethode getString() holt man den Stringinhalt der Textmarke, mit setString() wird der Textinhalt neu bestimmt. Wenn die Textmarke nur ein Punkt ist, wird der Text einfach vor der Textmarke eingefügt. Wird eine neu erzeugte Textmarke in den Text eingefügt, so bestimmt die Einfügeposition die Ankerposition der Textmarke.

Listing 401. *So fügt man eine Textmarke ein.*

```
Sub AddBookmark
  Dim oBookmark  'Einzufügende Textmarke
  Dim oCurs      'Textcursor

  REM Erzeugt einen Textcursor, der die letzten vier Zeichen
  REM des Dokuments enthält.
  oCurs = ThisComponent.Text.createTextCursor()
  oCurs.gotoEnd(False)
  oCurs.goLeft(4, True)

  REM Erzeugt eine Textmarke.
  oBookmark = ThisComponent.createInstance("com.sun.star.text.Bookmark")

  REM Wenn die Textmarke keinen Namen erhält, wird er automatisch von OOo vergeben.
```

```
    REM Wenn der Name schon existiert, wird dem Namen eine Zählung angehängt.
    oBookmark.setName("Bobert")

    REM Weil der Cursor mit True bewegt wurde, enthält die Textmarke die letzten vier
    REM Zeichen des Dokuments. Bei einer Bewegung mit False enthielte die Textmarke
    REM keine Zeichen, und sie würde vor das viertletzte Zeichen des Dokuments
    REM positioniert.
    ThisComponent.Text.insertTextContent(oCurs, oBookmark, False)
End Sub
```

14.12. Nummernkreise, Querverweise und Formatierung

Nehmen wir einmal die Beschriftung für Listing 400. Ohne das Format zu beachten, fügen Sie die Beschriftung folgendermaßen ein:

1. Schreiben Sie den Text „Listing „.
2. Öffnen Sie über **Einfügen | Feldbefehl | Andere** den Felderdialog.
3. Wählen Sie den Reiter Variablen.
4. Markieren Sie den Feldtyp Nummernkreis.
5. Geben Sie als Name „Listing“ ein und als Wert „Listing + 1“.
6. Klicken Sie auf Einfügen.
7. Schreiben Sie den Rest der Beschriftung.

Nummernkreise (Sequence-Felder) sind sehr praktisch, denn sie werden automatisch neu berechnet, wenn Beschriftungen hinzugefügt oder entfernt werden. Auch ein Querverweis auf ein Nummernkreisfeld wird neu berechnet, wenn das Feld selbst den Wert ändert, weil ein Feld hinzugefügt oder entfernt wurde.

Ich hatte einmal eine Reihe von Dokumenten mit Hunderten von Beschriftungen und Querverweisen. Unglücklicherweise bestanden alle Beschriftungen und Querverweise aus Text und nicht aus Feldern. Ich musste mir ein Makro schreiben, um die Beschriftungen und Querverweise in Nummernkreisfelder und damit verlinkte Querverweise zu konvertieren.

14.12.1. Zahlen und Datumsangaben formatieren

Nummernkreise können in vielfacher Weise formatiert werden. Im Endeffekt wird die Formatierung durch einen Formatstring definiert. OOo kennt von Hause aus viele allgemeine Zahlen- und Datumsformate, und Sie können nach Belieben neue Formate erzeugen. Zu jedem Zahlenformat gehört eine numerische ID. Objekte, die Zahlen und Datumsangaben formatieren – zum Beispiel Felder und Tabellenzellen –, enthalten den numerischen Schlüssel des Formatstrings, der für die Ausgabe des betreffenden Wertes herangezogen wird.

Auflistung der dem aktuellen Dokument bekannten Formate

Wenn Sie ein neues Zahlenformat erstellen, wird es im aktuellen Dokument gespeichert. Die unterstützten Zahlenformate und die IDs für bestimmte Formate differieren also von Dokument zu Dokument.

Listing 402. *Auflistung der Zahlenformate im aktuellen Dokument.*

```
Sub ListFormatsInCurrentDocument()
  Dim oDoc           'Neues Dokument für die Formatstrings
  Dim oFormats       'Liste der Formate im aktuellen Dokument
  Dim oFormat        'Das aktuelle Formatobjekt
  Dim oData          'Liste der Schlüssel zu den Formaten
  Dim i%             'Indexvariable
  Dim sFormat$       'Aktueller Formatstring
```

```
  Dim sPrevFormat$   'Vorheriger Formatstring
  Dim aLocale As New com.sun.star.lang.Locale  'Gebietsschema (Standard)

  oFormats = ThisComponent.getNumberFormats()

  'Erzeugt ein Dokument zur Datenausgabe.
  oDoc = StarDesktop.loadComponentFromURL _
                    ("private:factory/swriter", "_blank", 0, Array())
  'Liste der Schlüssel aller Zahlenformate
  oData = oFormats.queryKeys(com.sun.star.util.NumberFormat.ALL, aLocale, False)
  For i = LBound(oData) To UBound(oData)
    oFormat = oFormats.getByKey(oData(i))
    sFormat = oFormat.FormatString
    If sFormat<>sPrevFormat Then
      sPrevFormat = sFormat
      oDoc.getText().insertString(oDoc.getText().End, _
                        CStr(oData(i)) & Chr$(9) & sFormat & Chr$(10), False)
    End If
  Next
End Sub
```

Ein Zahlenformat suchen und erstellen

Wenn ein Zahlenformat hinzugefügt wird, kann es sein, dass es in einem leicht modifizierten Format gespeichert wird. QueryKey durchsucht die intern modifizierten Formatstrings, ohne vorher auch den Suchstring zu modifizieren. Das hat nach dem Speichern eines Schlüssels zur Folge, dass queryKey behauptet, es gebe diesen Formatstring gar nicht. Wenn Sie dann pflichtbewusst den Formatstring hinzufügen, kommt ein Laufzeitfehler, weil der String schon existiert. Zwei Beispiele dafür:

1. Wenn Sie „#.##0,00_);[Rot](#.##0,000)“ hinzufügen, wird „#.##0,00_);[ROT](#.##0,000)“ gespeichert, denn „Rot“ wird in Großbuchstaben konvertiert (s. https://issues.apache.org/ooo/show_bug.cgi?id=72380).
2. Beim Speichern wird „##0.0#\h“ zu „##.00#\h“.

Der übliche Rat ist, ein Zahlenformat zu wählen, das sich nicht ändert.

Listing 403 gibt die ID des spezifizierten Formats zurück. Diese ID wird zum Formatieren eines Wertes genutzt. Wenn Sie ein Format hinzufügen und das Format dann nicht verfügbar sein sollte, merken Sie sich die ID und holen Sie sich wie im Listing 402 das Format in der modifizierten Form.

Listing 403. *Ein Zahlenformat suchen und erstellen.*

```
'sFormat - Zu suchendes oder zu erstellendes Format.
'oDoc    - Das Dokument. Ohne Angabe wird das aktuelle Dokument genommen.
'locale  - Das Gebietsschema. Ohne Angabe wird das Standardgebietsschema genommen.
Function FindCreateNumberFormatStyle(sFormat$, Optional oDoc, Optional locale)
  Dim oDocument
  Dim aLocale As New com.sun.star.lang.Locale
  Dim oFormats
  Dim formatNum As Long
  oDocument = IIf(IsMissing(oDoc), ThisComponent, oDoc)
  oFormats = oDocument.getNumberFormats()
  'Wenn Sie eine typbezogene Abfrage wollen, müssen Sie die
  'Methode queryKeys() verwenden mit der Typangabe als Argument, zum Beispiel
  'com.sun.star.util.NumberFormat.DATE
  'Ich könnte das Gebietsschema mit den Werten aus folgenden Quellen festlegen:
  'http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt
```

```
    'http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
    'Mit einem NULL-Gebietsschema wird das eingestellte Standardschema genommen.
    'Als erstes wird geprüft, ob das Zahlenformat existiert.
    If (Not IsMissing(locale)) Then
      aLocale = locale
    End If
    formatNum = oFormats.queryKey(sFormat, aLocale, True)
    'MsgBox "Der gesuchte Formatschlüssel ist " & formatNum
    'Wenn das Zahlenformat nicht existiert, wird -1 zurückgegeben.
    'Dann wird es hinzugefügt.
    If (formatNum = -1) Then
      formatNum = oFormats.addNew(sFormat, aLocale)
      If (formatNum = -1) Then formatNum = 0
      'MsgBox "Der neue Formatschlüssel ist " & formatNum
    End If
    FindCreateNumberFormatStyle = formatNum
End Function
```

Standardformate

Mit der Methode getStandardFormat des Formats-Objekts ist es möglich, das Standardformat eines bestimmten Typs zu erhalten, ohne das Format manuell anzugeben. Der Formattyp ist ein Wert aus der Konstantengruppe com.sun.star.util.NumberFormat. Im folgenden Codeausschnitt werden Zellen in Calc auf die lokale Währungseinstellung formatiert.

```
oFormats = oDoc.NumberFormats          'In Basic identisch mit oDoc.getNumberFormats()
Dim aLocale As New com.sun.star.lang.Locale            'Standardgebietsschema
oRange = oSheet.getCellRangeByName("D2:F19")
oRange.NumberFormat = oFormats.getStandardFormat(_
             com.sun.star.util.NumberFormat.CURRENCY, aLocale)    'Zahlenformat Währung
```

Tabelle 162. *Die Konstantengruppe com.sun.star.util.NumberFormat.*

Wert	Name	Beschreibung
0	ALL	Alle Zahlenformate.
1	DEFINED	Nur benutzerdefinierte Zahlenformate.
2	DATE	Datumsformate.
4	TIME	Uhrzeitformate.
8	CURRENCY	Währungsformate.
16	NUMBER	Dezimale Zahlenformate.
32	SCIENTIFIC	Wissenschaftliche Zahlenformate.
64	FRACTION	Zahlenformate für Brüche.
128	PERCENT	Zahlenformate für Prozentangaben.
256	TEXT	Zahlenformate für Text.
6	DATETIME	Zahlenformate für Datum und Uhrzeit (= 2 + 4).
1024	LOGICAL	Boolesche Zahlenformate.
2048	UNDEFINED	Rückgabewert, wenn keine Formate existieren.

14.12.2. Ein Masterfeld erzeugen

Ein Feld des Nummernkreises Listing ist verknüpft mit dem Masterfeld com.sun.star.text.FieldMaster.SetExpression.Listing, das bestehen muss, bevor das Nummernfeld genutzt werden kann. Listing 404 zeigt, wie man ein noch nicht existierendes Masterfeld erzeugt und wie man auf ein existierendes zugreift.

Listing 404. *Ein Masterfeld erzeugen.*

```
oMasters = oDoc.getTextFieldMasters
If Not oMasters.hasByName("com.sun.star.text.FieldMaster.SetExpression.Listing") Then
  oMasterField = oDoc.createInstance("com.sun.star.text.FieldMaster.SetExpression")
  oMasterField.Name = "Listing"
  oMasterField.SubType = com.sun.star.text.SetVariableType.SEQUENCE
Else
  oMasterField = _
          oMasters.getByName("com.sun.star.text.FieldMaster.SetExpression.Listing")
End If
```

Der Subtyp des Masterfeldes legt den Feldtyp fest (s. Tabelle 163).

Tabelle 163. *Die Konstantengruppe com.sun.star.text.SetVariableType.*

Wert	Konstante	Beschreibung
0	com.sun.star.text.SetVariableType.VAR	Einfache Variable.
1	com.sun.star.text.SetVariableType.SEQUENCE	Nummernkreisfeld.
2	com.sun.star.text.SetVariableType.FORMULA	Formelfeld.
3	com.sun.star.text.SetVariableType.STRING	Stringfeld.

14.12.3. Ein Nummernkreisfeld einfügen

Zuerst wird das Nummernkreisfeld vom Dokument erzeugt, dann wird der Zahlentyp auf arabisch gesetzt und danach das Zahlenformat festgelegt. Das Feld wird mit dem Masterfeld verknüpft (das Masterfeld ist ein Nummernkreis), und der Feldinhalt wird gesetzt als Inkrement des Wertes (Listing + 1). Zu beachten ist, dass beim Einfügen des Feldes der Inhalt nicht sofort auch aktualisiert wird. Warten Sie entweder auf die Aktualisierung oder erzwingen Sie sie über **Extras | Aktualisieren | Felder**.

Listing 405. *Erzeugung eines SetExpression-Nummernkreisfeldes.*

```
oField = oDoc.createInstance("com.sun.star.text.TextField.SetExpression")
oField.NumberingType = com.sun.star.style.NumberingType.ARABIC
oField.NumberFormat = FindCreateNumberFormatStyle("###0", oDoc)
oField.attachTextFieldMaster(oMasterField)
oField.Content = name & " + 1"
```

Die Konstanten für den Zahlentyp sind in der Tabelle 159 aufgeführt.

14.12.4. Text durch ein Nummernkreisfeld ersetzen

Es folgt schließlich das Hauptarbeitsmakro, das Text sucht und durch ein Nummernkreisfeld ersetzt. Der zu ersetzende Text wird durch die Suche mit einem regulären Ausdruck gefunden.

„^Listing [:digit:]+\.[:space:]+“ sucht nach einer Zeile, die mit dem Text „Listing“ beginnt, dem ein Leerzeichen, eine oder mehrere Ziffern, ein Punkt und ein oder mehrere Leerzeichen folgen, zum Beispiel „Listing 12. “.

Listing 406. *Text durch ein Nummernkreisfeld ersetzen.*

```
Function FindReplaceTextFields(oDoc, name$) As Integer
  Dim oDescriptor  'Suchdeskriptor für die Suche nach den Beschriftungen
  Dim oFound       'Treffer bei der Suche
  Dim oMasters     'Masterfelder in diesem Dokument
  Dim oMasterField 'Das verknüpfte Masterfeld
  Dim oFrame       'Der Dokumentenframe, für den Dispatcher
  Dim oDispatcher  'Dispatcherobjekt
```

```
Dim oCurs        'Textcursor zum Einfügen von Textcontent
Dim oField       'Einzufügendes Nummernkreisfeld
Dim formatNum&   'ID (Schlüssel) für das Zahlenformat des Feldes
Dim i As Integer 'Indexvariable

'Ausgangspunkt: es wurden keine Textfelder eingefügt.
FindReplaceTextFields = 0

'Welches Zahlenformat für das Feld?
formatNum = FindCreateNumberFormatStyle("###0", oDoc)

'Falls erforderlich wird das Masterfeld erzeugt.
oMasters = oDoc.getTextFieldMasters
If Not oMasters.hasByName("com.sun.star.text.FieldMaster.SetExpression." & name) Then
  oMasterField = oDoc.createInstance("com.sun.star.text.FieldMaster.SetExpression")
  oMasterField.Name = name
  oMasterField.SubType = com.sun.star.text.SetVariableType.SEQUENCE
Else
  oMasterField = _
         oMasters.getByName("com.sun.star.text.FieldMaster.SetExpression." & name)
End If

'Suche nach Zeilen, die mit name, Leerzeichen, Zahl, Punkt, Leerzeichen beginnen.
oDescriptor = oDoc.createSearchDescriptor()
With oDescriptor
  .SearchString = "^" & name & " [:digit:]+\.[:space:]+"
  .SearchRegularExpression = True
End With

'Vorbereitung für einen Dispatch zum Entfernen der Formatierung.
oFrame = oDoc.CurrentController.Frame
oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")

'Start des Suchprozesses.
oFound = oDoc.findFirst(oDescriptor)
Do While Not IsNull(oFound)
  ' Zeichenvorlage "Stark betont" für zum Beispiel "Listing 7."
  oFound.CharStyleName = "OOoStrongEmphasis"

  'Erzeugt und konfiguriert das Nummernkreisfeld.
  oField = oDoc.createInstance("com.sun.star.text.TextField.SetExpression")
  oField.NumberingType = com.sun.star.style.NumberingType.ARABIC
  oField.NumberFormat = formatNum
  oField.attachTextFieldMaster(oMasterField)
  oField.Content = name & " + 1"

  'Erzeugt einen Textcursor mit dem gefundenen Text.
  oCurs = oFound.getText().createTextCursorByRange(oFound)

  'Fügt Text ein wie "Listing <sequence variable>. "
  'Dieser eingefügte Text ersetzt den gefundenen Text.
  oFound.getText().insertString(oCurs, name & " ", True)
  oCurs.collapseToEnd()
  oFound.getText().insertTextContent(oCurs, oField, True)
  oCurs.collapseToEnd()
  oFound.getText().insertString(oCurs, ".", True)
  oCurs.collapseToEnd()
  oFound.getText().insertString(oCurs, " ", True)
```

```
    oCurs.collapseToEnd()

    'Der eingefügte Text wird stark betont formatiert.
    'Cursor einen Schritt nach links, so dass er vor dem Leerzeichen steht.
    'Sprung ans Absatzende und das Standardformat wiederherstellen.
    oCurs.goLeft(1, False)
    oCurs.gotoEndofParagraph(True)
    oDoc.CurrentController.Select(oCurs)
    oDispatcher.executeDispatch(oFrame, ".uno:ResetAttributes", "", 0, Array())

    'Suche nach dem nächsten Treffer.
    oFound = oDoc.findNext(oFound.End, oDescriptor)
    i = i + 1
  Loop
  FindReplaceTextFields = i
End Function
```

Die oben aufgeführte Methode enthält Extracode, der das Standardformat hinter den Nummernkreisvariablen wiederherstellt.

14.12.5. Einen Querverweis (GetReference-Feld) erzeugen

Ein eingefügtes Nummernkreisfeld ist direkt mit einem Masterfeld verknüpft, das wiederum von vielen anderen Feldern referenziert werden kann. Jedes Listing in diesem Buch referenziert ein einziges Masterfeld. Ein GetReference-Feld kann jedoch nicht so einfach ein Feld referenzieren. Wenn ein Feld eingefügt wird, vergibt OOo automatisch einen Nummerierungswert. Das GetReference-Feld hat die Eigenschaft SequenceNumber, die vom referenzierten Feld die numerische Eigenschaft SequenceValue bezieht, s. Listing 407.

Die Eigenschaft ReferenceFieldSource gibt an, welcher Feldtyp referenziert wird. In der Tabelle 165 finden Sie die unterstützten Typen. Vieles davon ist schlecht dokumentiert, aber ich wage die Vermutung, dass es für jeden Feldquelltyp einen eindeutigen Nummerierungswert gibt.

Die Eigenschaft ReferenceFieldPart bestimmt, wie der Querverweis angezeigt wird. Die gültigen Werte finden Sie in der Tabelle 164. Im Beispiel im Listing 407 wird Kategorie und Nummer gewählt. In einer Beschriftung ist die Kategorie der gesamte Text vor der Zählung.

Listing 407. *Ein GetReference-Feld einfügen.*

```
oField = oDoc.createInstance("com.sun.star.text.textfield.GetReference")
oField.ReferenceFieldPart = com.sun.star.text.ReferenceFieldPart.CATEGORY_AND_NUMBER
oField.ReferenceFieldSource = com.sun.star.text.ReferenceFieldSource.SEQUENCE_FIELD
oField.SequenceNumber = oReferencedField.SequenceValue
oField.SourceName = sSeqName
oText.insertTextContent(oCurs, oField, True)
```

Tabelle 164. *Die Konstantengruppe com.sun.star.text.ReferenceFieldPart.*

Wert	Konstante	Beschreibung
0	PAGE	Die Seitenzahl in Arabischen Ziffern.
1	CHAPTER	Die Kapitelnummer.
2	TEXT	Der Referenztext = Kategorie und Nummer und Beschriftungstext.
3	UP_DOWN	Die Wörter für „oben“ und „unten“ in der aktuellen Sprache.
4	PAGE_DESC	Die Seitenzahl in dem Zahlenformat, das in der Seitenvorlage der referenzierten Position festgelegt ist.
5	CATEGORY_AND_NUMBER	Die Kategorie und die Zählung, zum Beispiel „Listing 7“.

Wert	Konstante	Beschreibung
6	ONLY_CAPTION	Der Beschriftungstext.
7	ONLY_SEQUENCE_NUMBER	Die Zählung ohne die Kategorie.
8	NUMBER	Die Absatznummerierung der Referenz und abhängig vom Kontext des Referenzfeldes noch weitere Nummerierungen übergeordneter Absatzebenen.
9	NUMBER_NO_CONTEXT	Die Absatznummerierung der Referenz.
10	NUMBER_FULL_CONTEXT	Die Absatznummerierung der Referenz und die Nummerierungen übergeordneter Absatzebenen.

Tabelle 165. *Die Konstantengruppe com.sun.star.text.ReferenceFieldSource.*

Wert	Konstante	Beschreibung
0	REFERENCE_MARK	Die Quelle ist eine Referenz.
1	SEQUENCE_FIELD	Die Quelle ist ein Nummernkreis.
2	BOOKMARK	Die Quelle ist eine Textmarke.
3	FOOTNOTE	Die Quelle ist eine Fußnote.
4	ENDNOTE	Die Quelle ist eine Endnote.

14.12.6. Text durch einen Querverweis ersetzen

Listing 408 übernimmt ein Stringarray und einen String, der in dem Array gesucht werden soll. Die Funktion gibt den Index des im Array gefundenen Strings zurück. Der Einsatzzweck dieses Makros wird weiter unten erläutert.

Listing 408. *Einen String in einem Array suchen.*

```
Function FindStringInArray(oData, s) As Integer
  Dim i As Integer
  FindStringInArray = -1      'Nicht gefunden
  For i = LBound(oData) To UBound(oData)
    If oData(i) = s Then
      FindStringInArray = i
      Exit Function
    End If
  Next
End Function
```

Wie das Hauptmakro im einzelnen arbeitet, wird in dessen Kommentaren erläutert, s. Listing 409.

Listing 409. *Text durch ein GetReference-Feld ersetzen.*

```
Function ReferenceSequenceVariables(oDoc, sSeqName) As Integer
  Dim oEnum             'Enumeration der Textfelder
  Dim oField            'Einzelnes Textfeld
  Dim s$                'Allgemeine Stringvariable
  Dim oFrame            'Frame des Dokuments, benötigt für Dispatches
  Dim oDispatcher       'Dispatcherobjekt
  Dim oFields()         'Die SetExpression-Felder, die referenziert werden könnten
  Dim sPresentations()  'Die Anzeigetexte der oFields()-Elemente, wie "Listing 7"
  Dim i As Integer      'Index einer Referenz (wie "Listing 7") in sPresentations()
  Dim n As Integer      'Anzahl der Felder
  Dim oCurs             'Cursor für die Zurücksetzung der Formatierung auf Standard
  Dim oDescriptor       'Der Suchdeskriptor
  Dim oFound            'Der Trefferrange

  'Grundannahme: Keine Querverweise erzeugt.
```

```
ReferenceSequenceVariables = 0

'Voraussetzungen für einen Dispatch.
oFrame = oDoc.CurrentController.Frame
oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")

'Enumeriert die Textfelder und ermittelt die entsprechenden Feldnamen.
'Die Feld-"Presentation" ist die angezeigte Nummerierung (mit Kategorie).
'Wenn der folgende Abschnitt beendet ist, dann sind zwei Arrays gefüllt:
'oFields() - Alle SetExpression-Felder des übergebenen Namens.
'sPresentations() - Die vollständigen Anzeigetexte der oFields()-Elemente, zum
'Beispiel "Listing 7" für den "Listing"-Nummernkreis mit der Zählung 7.
oEnum = oDoc.getTextFields().createEnumeration()
If Not IsNull(oEnum) Then
  Do While oEnum.hasMoreElements()
    oField = oEnum.nextElement()
    If oField.supportsService("com.sun.star.text.TextField.SetExpression") Then
      If oField.VariableName = sSeqName Then
        ReDim Preserve oFields(0 To n)
        ReDim Preserve sPresentations(0 To n)
        oFields(n) = oField
        sPresentations(n) = sSeqName & " " & oField.CurrentPresentation
        n = n + 1
      End If
    End If
  Loop
End If

'Erzeugt den Suchdeskriptor für die Suche nach Dingen wie "Listing 23".
'Strings in Feldern werden nicht gefunden, denn dort greift die Suche nicht.
oDescriptor = oDoc.createSearchDescriptor()
With oDescriptor
  .SearchString = sSeqName & " [:digit:]+"
  .SearchRegularExpression = True
End With

n = 0
oFound = oDoc.findFirst(oDescriptor)
Do While Not IsNull(oFound)
  'Sucht nach einem Feld mit einem Anzeigetext wie "Listing 7".
  i = FindStringInArray(sPresentations, oFound.getString())
  If i >= 0 Then
    'Erzeugt und konfiguriert das GetReference-Feld.
    oField = oDoc.createInstance("com.sun.star.text.textfield.GetReference")
    oField.ReferenceFieldPart = _
           com.sun.star.text.ReferenceFieldPart.CATEGORY_AND_NUMBER
    oField.ReferenceFieldSource = _
           com.sun.star.text.ReferenceFieldSource.SEQUENCE_FIELD
    oField.SequenceNumber = oFields(i).SequenceValue
    oField.SourceName = sSeqName

    'Erzeugt einen Cursor an der Fundstelle.
    'Der Text wird gelöscht, und das GetReference-Feld wird an der Stelle eingefügt.
    oCurs = oFound.getText().createTextCursorByRange(oFound)
    oCurs.setString("")
    oFound.getText().insertTextContent(oCurs, oField, True)
```

```
        'Manchmal wurde der Referenztext speziell formatiert.
        'Ein Cursor kann nicht in ein Feld hinein, also wird erst der Cursor auf einen
        'Punkt am Ende reduziert und dann um ein Zeichen nach links bewegt. Dadurch
        'wird das GetReference-Feld ausgewählt.
        oCurs.collapseToEnd()
        oCurs.goLeft(1, True)

        'Markiert das GetReference-Feld auf dem Display und setzt mit einem Dispatch
        'alle Attribute auf den Standard zurück.
        oDoc.CurrentController.select(oCurs)
        oDispatcher.executeDispatch(oFrame, ".uno:ResetAttributes", "", 0, Array())
        n = n + 1
      Else
        s = s & "Kein " & oFound.getString() & " gefunden." & Chr$(10)
      End If
      oFound = oDoc.findNext(oFound.End, oDescriptor)
    Loop
    ReferenceSequenceVariables = n
    If s <> "" Then
      Msgbox s
    End If
End Function
```

14.12.7. Das Makro, das alles zusammenfügt

Das Hauptmakro startet damit, dass alle Beschriftungen für Bild, Listing und Tabelle durch Nummernkreisvariablen ersetzt werden. Es wird stillschweigend vorausgesetzt, dass die erste Beschriftung die Nummer 1 hat und dass die Beschriftungen fortlaufend ohne Lücken in ihrem Auftreten im Dokument gezählt sind. Im Grunde genommen sind die Routinen ein wenig fragil, sie funktionierten aber ausgezeichnet in den von mir dafür vorgesehenen Dokumenten.

Mein erster Test ersetzte etwa 100 Beschriftungen. Alle Querverweise missglückten, weil die Felder noch nicht aktualisiert waren. Daher wird nach dem Ersetzen aller Nummernkreisfelder eine Aktualisierung durch einen Dispatch durchgeführt, bevor die nächste Phase starten kann. Dieser Schritt ist wichtig, denn sonst wäre die Eigenschaft CurrentPresentation in den Nummernfeldern unkorrekt.

Listing 410. *Ersetzt Beschriftungen und Querverweise in einem Dokument.*

```
Sub ReplaceCaptions()
  Dim oDoc         'Das zu bearbeitende Dokument
  Dim oFrame       'Frame des Dokuments, benötigt für Dispatches
  Dim oDispatcher  'Dispatcherobjekt
  Dim i As Integer 'Allgemeine Indexvariable
  Dim n As Integer 'Anzahl der erzeugten Felder
  Dim s$           'Arbeitszusammenfassung
  Dim fields()     'Namen der betreffenden Felder

  fields = Array("Listing", "Tabelle", "Bild")
  oDoc = ThisComponent
  oFrame = oDoc.CurrentController.Frame
  oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")

  For i = LBound(fields()) To UBound(fields())
    n = FindReplaceTextFields(ThisComponent, fields(i))
    s = s & n & " Beschriftungen für " & fields(i) & " erledigt."& Chr$(10)
  Next

  'Der nächste Teil wird NUR funktionieren, wenn alle Felder aktualisiert sind.
```

```
    'Also wird ein Update erzwungen.
    oDispatcher.executeDispatch(oFrame, ".uno:UpdateFields", "", 0, Array())

    For i = LBound(fields()) To UBound(fields())
      n = ReferenceSequenceVariables(ThisComponent, fields(i))
      s = s & n & " Querverweise auf " & fields(i) & " gesetzt." & Chr$(10)
    Next
    MsgBox s
  End Sub
```

14.13. Inhaltsverzeichnisse

Ein Inhaltsverzeichnis (TOC für englisch Table Of Contents) zu finden und einzufügen, ist einfach, wenn Sie die Standardeinstellungen nicht ändern wollen. Das folgende Makro überprüft, ob es im aktuellen Dokument ein Inhaltsverzeichnis gibt.

```
    REM Findet ein TOC, falls es existiert.
    oIndexes = ThisComponent.getDocumentIndexes()  'Liste aller Verzeichnisse
    bIndexFound = False
    For i = 0 To oIndexes.getCount() - 1
      oIndex = oIndexes.getByIndex(i)
      'Überprüfung, ob es ein Inhaltsverzeichnis ist.
      If oIndex.supportsService("com.sun.star.text.ContentIndex") Then
        bIndexFound = True
        Exit For
      End If
    Next
```

Mit der Objektmethode dispose entfernen Sie einen bestehenden Index aus dem Dokument.

Wenn ich ein TOC erzeuge, setze ich im Normalfall CreateFromOutline auf True, um das Verzeichnis auf der Basis der Überschriften-Absatzformatvorlagen aufzubauen, s. Listing 411.

Listing 411. *Fügt ein Standard-TOC in das Dokument ein.*

```
Sub InsertATOC
  REM Autor: Andrew Pitonyak
  Dim oCurs       'Wird zum Einfügen von Textcontent benötigt
  Dim oIndexes    'Alle existierenden Verzeichnisse
  Dim oIndex      'TOC, entweder das existierende oder ein neu erzeugtes
  Dim i%          'Zählwert zur Suche nach einem existierenden TOC
  Dim bIndexFound As Boolean 'Flag zur Kennzeichnung, ob ein TOC gefunden wurde

  REM Sucht ein existierendes TOC.
  oIndexes = ThisComponent.getDocumentIndexes()
  bIndexFound = False
  For i = 0 To oIndexes.getCount() - 1
    oIndex = oIndexes.getByIndex(i)
    If oIndex.supportsService("com.sun.star.text.ContentIndex") Then
      bIndexFound = True
      Exit For
    End If
  Next
  If Not bIndexFound Then
    Print "Ich habe kein Inhaltsverzeichnis finden können."
    REM Ein neues TOC wird erzeugt und eingefügt.
    REM Es muss von dem Dokument erzeugt werden, in dem es stehen wird.
    oIndex = ThisComponent.createInstance("com.sun.star.text.ContentIndex")
```

```
    oIndex.CreateFromOutline = True
    oCurs = ThisComponent.getText().createTextCursor()
    oCurs.gotoStart(False)
    ThisComponent.getText().insertTextContent(oCurs, oIndex, False)
  End If
  REM Auch das neu eingefügte Verzeichnis ist noch nicht aktualisiert, bis JETZT!
  oIndex.update()
End Sub
```

Derselbe Code kann ein Stichwort-, Inhalts-, Abbildungs-, Tabellen-, Objekt- oder benutzerdefiniertes Verzeichnis erzeugen. Tabelle 166 zeigt die allen Verzeichnistypen gemeinsamen Eigenschaften.

Tabelle 166. *Gemeinsame Verzeichniseigenschaften.*

Eigenschaft	Beschreibung
BackColor	Hintergrundfarbe. Keine Hintergrundfarbe = -1.
BackGraphicFilter	Filter für die als Hintergrund dienende Grafik.
BackGraphicLocation	Positionierung der Hintergrundgrafik als Wert der Konstantengruppe com.sun.star.style.GraphicLocation: NONE, LEFT_TOP, MIDDLE_TOP, RIGHT_TOP, LEFT_MIDDLE, MIDDLE_MIDDLE, RIGHT_MIDDLE, LEFT_BOTTOM, MIDDLE_BOTTOM, RIGHT_BOTTOM, AREA oder TILED.
BackGraphicURL	URL der als Hintergrund dienenden Grafik. Ich habe das nicht experimentell ausprobiert. Ist ein interner URL gemeint?
CreateFromChapter	Falls True, wird ein Verzeichnis basierend auf dem aktuellen Kapitel aufgebaut und nicht basierend auf dem gesamten Dokument.
IsProtected	Falls True, wird das Verzeichnis schreibgeschützt sein. Es kann dann nicht wie normaler Text editiert werden.
ParaStyleHeading	Absatzvorlagenname der Überschrift.
ParaStyleLevel1	Absatzvorlagenname für Ebene 1. Es gibt je eine Eigenschaft für die Ebenen 1 bis 10.
Title	Verzeichnistitel als Teil des TOC. Ich lasse diese Eigenschaft normalerweise leer und platziere den Titel vor das TOC mit dem Format „Überschrift 1“, damit der Titel Teil des TOC wird. Irgendwie töricht, ich weiß.

Ein TOC enthält Datenspalten, zum Beispiel den Nummerierungsbereich, den Titelbereich und die Seitenzahl. Jede Spalte wird durch ein Array benannter Eigenschaften (Properties) repräsentiert. Tabelle 167 zeigt die unterstützten Properties.

Tabelle 167. *Unterstützte Spaltenmerkmale.*

Property	Beschreibung
TokenType	Legt den Inhalt der Spalte fest. Jede Spalte hat einen Kürzeltyp (Token) als erste Property – ein String aus der in Tabelle 168 gezeigten Liste.
CharacterStyleName	Name der Zeichenvorlage für das Element. Nicht für Tabulatoren anzuwenden. Ist der Wert leer, wird die Standardvorlage genutzt.
TabStopRightAligned	Tabulator ist rechtsbündig. Nur für Token TabStop.
TabStopPosition	Position des Tabulators. Nur für Token TabStop.
TabStopFillCharacter	Füllzeichen in Tabulatoren. Nur für Token TabStop.
WithTab	Falls True, wird ein Tabulator eingefügt.
Text	Nur bei benutzerdefiniertem Text verwendet.
ChapterFormat	Nur für die Tokens ChapterInfo und EntryNumber. Das Kapitelformat wurde schon in der Tabelle 158 als Konstantengruppe com.sun.star.text.ChapterFormat erwähnt. Für ChapterInfo sind NUMBER und NAME_NUMBER erlaubt. Für EntryNumber sind NUMBER und DIGIT erlaubt.

Property	Beschreibung
ChapterLevel	Nur für die Tokens ChapterInfo und EntryNumber. Kennzeichnet die Ebene, bis zu der Kapitelinformationen gegeben werden. Werte von 1 bis 10.

Manche Tokentypen werden nicht von allen Verzeichnistypen unterstützt, s. Tabelle 168. Der dort angegebene Wert ist der den Tokentyp identifizierende String. Die Spalte Eintrag zeigt das Kürzel, das im GUI als Spaltenspezifizierung verwendet wird. Wenn in der Tabelle ein Spalteneintrag leer ist, dann weil ich nicht die Werte für jeden Typ überprüft habe und den Wert einfach nicht kenne.

Tabelle 168. *Unterstützte TokenType-Werte.*

Wert	Eintrag	Kommentar	Unterstützte Verzeichnistypen
TokenEntryNumber	E#	Kapitelnummer	Inhaltsverzeichnis
TokenEntryText	E	Eintragstext	Alle
TokenTabStop	T	Tabulator	Alle
TokenText		Benutzerdefinierter Text	
TokenPageNumber	#	Seitenzahl	Alle
TokenChapterInfo		Kapitelinformation	Abbildungs-, Tabellen-, Objekt-, Stichwort- und benutzerdefiniertes Verzeichnis
TokenHyperlinkStart	LS	Start eines Hyperlinks	
TokenHyperlinkEnd	LE	Ende eines Hyperlinks	
TokenBibliographyDataField		Datenfeld der Bibliografie	

Das Makro im Listing 412 inspiziert die Ebenenformatierung des TOC im aktuellen Dokument und fügt die Ausgabe als Text an das Ende des aktuellen Dokuments an.

Listing 412. *Inspizierung des TOC im aktuellen Dokument.*

```
Sub InspectCurrentTOCColumns
  'Inspiziert ThisComponent
  Dim oCurs       'Wird zum Einfügen von Textcontent benötigt
  Dim oIndexes    'Alle existierenden Verzeichnisse
  Dim oIndex      'TOC, entweder das existierende oder ein neu erzeugtes.
  Dim i%          'Zählwert zur Suche nach einem existierenden TOC
  Dim bIndexFound As Boolean 'Flag zur Kennzeichnung, ob ein TOC gefunden wurde
  Dim iLevel%     'Iteration über die Ebenen in der Eigenschaft LevelFormat
  Dim iCol%       'Iteration über die Spalten der jeweiligen Ebene
  Dim iProp%      'Iteration über die Properties der jeweiligen Spalte
  Dim oLevel      'Ebenenobjekt
  Dim oCol        'Spaltenobjekt
  Dim s$

  REM Sucht ein existierendes TOC.
  oIndexes = ThisComponent.getDocumentIndexes()
  bIndexFound = False
  For i = 0 To oIndexes.getCount() - 1
    oIndex = oIndexes.getByIndex(i)
    If oIndex.supportsService("com.sun.star.text.ContentIndex") Then
      bIndexFound = True
      Exit For
    End If
  Next
  If Not bIndexFound Then
```

```
    Exit Sub
  End If
  REM Die Spaltenüberschriften.
  s = "Ebene" & Chr$(9) & "Spalte" & Chr$(9) & "Property" & Chr$(9) & "Name" & _
              Chr$(9) & "Wert" & Chr$(13)
  For iLevel = 0 To oIndex.LevelFormat.getCount() - 1
    oLevel = oIndex.LevelFormat.getByIndex(iLevel)
    For iCol = LBound(oLevel) To UBound(oLevel)
      oCol = oLevel(iCol)
      For iProp = LBound(oCol) To UBound(oCol)
        s = s & iLevel & Chr$(9) & iCol & Chr$(9) & iProp  & Chr$(9) & _
              oCol(iProp).Name & Chr$(9) & oCol(iProp).Value & Chr$(13)
      Next
    Next
  Next
  ThisComponent.getText().insertString(ThisComponent.getText().End, s, False)
End Sub
```

Ich habe das Makro im Listing 412 laufen lassen und festgestellt, dass die Ebene 0 keine Einträge enthält und dass die Ebenen 1 – 10 identisch sind. Tabelle 169 zeigt die Ausgabe für die Ebene 1.

- Spalte 0 ist der Typ TokenEntryNumber, mit der Standardzeichenvorlage formatiert. In der TOC ist es die Kapitelnummerierung.
- Spalte 1 kennzeichnet den Beginn eines Hyperlinks, formatiert mit der Zeichenvorlage „Internetlink“.
- Spalte 2 enthält den Text der Überschrift. Es ist keine Zeichenvorlage festgelegt, weil die Formatierung aus dem Format des Hyperlinks resultiert.
- Spalte 3 kennzeichnet das Ende des Hyperlinks, der somit nur den Überschriftentext umfasst.
- Spalte 4 kennzeichnet einen rechtsbündigen Tabulator, der den freien Raum zwischen dem Text der Überschrift und der Seitenzahl mit Punkten füllt.
- Spalte 5 enthält die Seitenzahl.

Tabelle 169. *LevelFormat für die Ebene 1.*

Ebene	Spalte	Property	Name	Wert
1	0	0	TokenType	TokenEntryNumber
1	0	1	CharacterStyleName	
1	1	0	TokenType	TokenHyperlinkStart
1	1	1	CharacterStyleName	Internet link
1	2	0	TokenType	TokenEntryText
1	2	1	CharacterStyleName	
1	3	0	TokenType	TokenHyperlinkEnd
1	4	0	TokenType	TokenTabStop
1	4	1	TabStopRightAligned	True
1	4	2	TabStopFillCharacter	.
1	4	3	CharacterStyleName	
1	4	4	WithTab	True
1	5	0	TokenType	TokenPageNumber
1	5	1	CharacterStyleName	

Wenn man ein TOC manuell erstellt, werden die Spalten mit solchen Dingen editiert wie Kapitelnummer, Eintragstext, Hyperlink, Seitennummer und anderen. Diese Werte werden in der Eigenschaft LevelFormat gespeichert.

Listing 413. *Ein TOC mit Hyperlinks einfügen.*

```
Sub InsertATOCWithHyperlinks
  REM Autor: Andrew Pitonyak
  Dim oCurs       'Wird zum Einfügen von Textcontent benötigt.
  Dim oIndexes    'Alle existierenden Verzeichnisse.
  Dim oIndex      'TOC, entweder das existierende oder ein neu erzeugtes.
  Dim i%          'Zählwert zur Suche nach einem existierenden TOC.
  Dim bIndexFound As Boolean 'Flag zur Kennzeichnung, ob ein TOC gefunden wurde.
  Dim iLevel%

  REM Sucht ein existierendes TOC.
  oIndexes = ThisComponent.getDocumentIndexes()
  bIndexFound = False
  For i = 0 To oIndexes.getCount() - 1
    oIndex = oIndexes.getByIndex(i)
    If oIndex.supportsService("com.sun.star.text.ContentIndex") Then
      bIndexFound = True
      Exit For
    End If
  Next
  If Not bIndexFound Then
    REM Erzeugt ein neues TOC und fügt es an der Cursorposition ein.
    REM Das neue TOC muss von dem Dokument erzeugt werden, in dem es stehen soll.
    oIndex = ThisComponent.createInstance("com.sun.star.text.ContentIndex")

    oIndex.CreateFromOutline = True  'Die Gliederungsebenen werden verwendet.

    REM Ignoriert die Ebene 0.
    For iLevel = 1 To oIndex.LevelFormat.getCount() - 1
      REM Ersetzt das Ebenenformat.
      oIndex.LevelFormat.replaceByIndex(iLevel, CreateTOCColumnEntries())
    Next

    oCurs = ThisComponent.getText().createTextCursor()
    oCurs.gotoRange(ThisComponent.CurrentController.ViewCursor, False)
    ThisComponent.getText().insertTextContent(oCurs, oIndex, False)
  End If
  REM Auch das neu eingefügte Verzeichnis ist noch nicht aktualisiert, bis JETZT!
  oIndex.update()
End Sub

Function CreateTOCColumnEntries()
  Dim o
  o = Array(Array(MakeProperty("TokenType", "TokenEntryNumber"), _
                  MakeProperty("CharacterStyleName", "")), _
            Array(MakeProperty("TokenType", "TokenHyperlinkStart"), _
                  MakeProperty("CharacterStyleName", "Internet link")), _
            Array(MakeProperty("TokenType", "TokenEntryText"), _
                  MakeProperty("CharacterStyleName", "")), _
            Array(MakeProperty("TokenType", "TokenHyperlinkEnd")), _
            Array(MakeProperty("TokenType", "TokenTabStop"), _
                  MakeProperty("TabStopRightAligned", True), _
```

```
                MakeProperty("TabStopFillCharacter", "."), _
                MakeProperty("CharacterStyleName", ""),  _
                MakeProperty("WithTab", True)), _
            Array(MakeProperty("TokenType", "TokenPageNumber"), _
                MakeProperty("CharacterStyleName", "")))
  CreateTOCColumnEntries() = o
End Function

Function MakeProperty(sName$, value)
  REM Praktische Verkürzung lästiger Schreibarbeit.
  Dim oProp
  oProp = CreateObject("com.sun.star.beans.PropertyValue")
  oProp.Name = sName
  oProp.Value = value
  MakeProperty = oProp
End Function
```

14.14. Fazit

Obwohl dieses Kapitel nicht jedes vom Writer unterstützte Objekt und nicht jede Funktionalität behandelt hat, so enthält es doch die Funktionalitäten, die in den Mailinglisten am meisten nachgefragt werden. Viele der hier vorgestellten Techniken sind beispielhaft für die Techniken, die für nicht behandelte Objekte benötigt werden. Zum Beispiel wird auf alle Objekte, die einen namentlichen Zugriff unterstützen, auf dieselbe Art zugegriffen. Betrachten Sie das hier ausgebreitete Material als Ausgangspunkt Ihrer Erkundungsreise durch die OOo-Textdokumente.

15. Tabellendokumente

Ein Calc-Dokument dient hauptsächlich dazu, eine Reihe von Arbeitsblättern zu verwalten, die ihrerseits die Daten in Zeilen und Spalten aufnehmen – mit einem Wort gesagt, in Tabellen. Dieses Kapitel stellt Methoden vor, den Inhalt eines Calc-Dokuments zu ändern, zu inspizieren, zu formatieren und zu modifizieren.

OOo unterstützt drei wesentliche Tabellentypen: Texttabellen in Textdokumenten, Datenbanktabellen und Arbeitsblätter in Tabellendokumenten. Diese verschiedenen Tabellentypen sind jeweils auf einen bestimmten Anwendungsbereich zugeschnitten. Texttabellen in Textdokumenten unterstützen komplexe Textformatierung, aber nur vergleichsweise einfache Berechnungsmöglichkeiten. Arbeitsblattdokumente andererseits unterstützen komplexe Berechnungen, aber nur relativ einfache Textformatierungen.

Der Grundaufbau aller Dokumenttypen ist gleich. Sie haben zwei Komponenten: die enthaltenen Daten und den Controller, der die Darstellung der Daten organisiert. In OOo wird die Kollektion der Daten in einem Dokument das *Modell* genannt. Jedes Modell hat einen Controller, der für die visuelle Präsentation der Daten zuständig ist. Der Controller kennt die Position des sichtbaren Textcursors, die aktuelle Seite und den aktuell ausgewählten Bereich.

Jedes Tabellendokument unterstützt den Service com.sun.star.sheet.SpreadsheetDocument. Wenn ich ein benutzerfreundliches Makro schreibe, das ein Tabellendokument erfordert, überprüfe ich, ob das Dokument den richtigen Typ hat: ich rufe die Objektmethode supportsService() auf, s. Listing 414.

Listing 414. *Calc-Dokumente unterstützen den Service com.sun.star.sheet.SpreadsheetDocument.*

```
REM Hier wird ein Trick verwendet, um einen OOo-Bug zu umgehen.
REM Diese Funktion testet, ob das Argument einen Service unterstützt.
REM Wenn das Objekt keinen Service unterstützt, tritt ein Laufzeitfehler auf,
REM der sich darüber beklagt, dass die Variable nicht belegt sei.
REM Der Fehler wird vermieden, wenn das Argument einer temporären Variablen
REM zugewiesen wird und der supportsService-Aufruf mit dieser erfolgt.
Function IsCalcDocument(oDoc) As Boolean
  On Error Goto ErrorJumpPoint
  Dim s$ : s$ = "com.sun.star.sheet.SpreadsheetDocument"

  IsCalcDocument = False
  If oDoc.supportsService(s$) Then
    IsCalcDocument = True
  End If
  ErrorJumpPoint:
End Function
```

Ein Interface definiert eine Reihe von Methoden. Wenn ein Objekt ein Interface einbindet, so wird damit auch jede Methode dieses Interface eingesetzt. Ein Service definiert ein Objekt dadurch, dass er die eingebundenen Interfaces, die enthaltenen Eigenschaften und die exportierten anderen Services spezifiziert. Ein Service spezifiziert die eingesetzten Methoden indirekt durch die Spezifizierung der Interfaces. Die von Calc-Dokumenten unterstützten Interfaces bieten einen guten Überblick über die verfügbaren Funktionalitäten, s. Tabelle 170.

Tabelle 170. *Einige von Calc-Dokumenten unterstützte Interfaces.*

Service	Beschreibung
com.sun.star.document.XActionLockable	Blockiert zeitweilig das Dokument gegen Benutzereingaben und gegen automatische Zellaktualisierung. Dadurch wird es möglich, in schneller Abfolge verschiedene Objektteile zu ändern, ohne dass die Einzelteile in den Zwischenschritten durch automatische Aktualisierung in einen zeitweilig ungültigen Zustand versetzt würden.
com.sun.star.drawing.XDrawPagesSupplier	Zugriff auf alle Folien in diesem Dokument. Es gibt für jedes enthaltene Tabellenblatt je eine Folie.
com.sun.star.sheet.XCalculatable	Kontrolliert die automatische Zellberechnung.
com.sun.star.sheet.XConsolidatable	Datenkonsolidierung.
com.sun.star.sheet.XGoalSeek	Zielwertsuche für eine Zelle.
com.sun.star.sheet.XSpreadsheetDocument	Zugriff auf die enthaltenen Tabellenblätter.
com.sun.star.style.XStyleFamiliesSupplier	Zugriff auf die enthaltenen Vorlagen nach Typen.
com.sun.star.util.XNumberFormatsSupplier	Zugriff auf die Zahlenformate.
com.sun.star.util.XProtectable	Setzt und entfernt den Schreibschutz auf Dokumente.

Die Funktion CreateNewCalcDoc erzeugt ein neues leeres Calc-Dokument. Sie wird von anderen Routinen in diesem Kapitel verwendet.

Listing 415. *Erzeugt ein neues Calc-Dokument.*

```
Function CreateNewCalcDoc
  Dim noArgs()          'Ein leeres Array als Argument.
  Dim sURL As String    'URL des neuen Dokuments.
  Dim oDoc

  sURL = "private:factory/scalc"
  oDoc = StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, noArgs())
  CreateNewCalcDoc = oDoc
End Function
```

15.1. Zugriff auf Tabellenblätter

Im wesentlichen dient ein Tabellendokument durch das Interface XSpreadsheetDocument als Container für einzelne Tabellenblätter. In diesem Interface ist nur eine Methode definiert, nämlich getSheets(), die ein Spreadsheets-Objekt zurückgibt zur Manipulation der einzelnen Tabellenblätter, s. Listing 416.

Listing 416. *Zugriff auf den Service com.sun.star.sheet.Spreadsheets, einmal über eine Methode und einmal über eine Eigenschaft.*

```
ThisComponent.getSheets() 'Vom Interface XSpreadsheetDocument definierte Methode.
ThisComponent.Sheets      'Eigenschaft des Tabellendokuments.
```

Der Service Spreadsheets ermöglicht den Zugriff auf einzelne Tabellenblätter über Index, Enumeration und Namen (s. Tabelle 171). Er enthält auch Methoden zum Erzeugen, Verschieben und Löschen von Tabellenblättern. Einige der in der Tabelle 171 aufgeführten Methoden werden im Listing 417 demonstriert.

Tabelle 171. *Methoden im Service com.sun.star.sheet.Spreadsheets.*

Methode	Beschreibung
copyByName(srcName, destName, index)	Kopiert das Tabellenblatt srcName unter dem Namen destName zur Indexposition index.
createEnumeration()	Erzeugt ein Objekt, das die Tabellenblätter enumeriert.

Methode	Beschreibung
getByIndex(index)	Zugriff auf ein Tabellenblatt über den Blattindex.
getByName(name)	Zugriff auf ein Tabellenblatt über den Blattnamen.
getCount()	Gibt die Anzahl der Tabellenblätter als Long Integer zurück.
hasByName(name)	Gibt True zurück, wenn das genannte Tabellenblatt existiert.
hasElements()	Gibt True zurück, wenn das Dokument mindestens ein Tabellenblatt enthält.
insertNewByName(name, index)	Erzeugt ein neues Tabellenblatt und fügt es an der Indexposition mit dem angegebenen Namen ein.
moveByName(name, index)	Verschiebt das genannte Tabellenblatt an die angegebene Indexposition.
removeByName(name)	Löscht das genannte Tabellenblatt.

Listing 417. *Tabellenblattmanipulationen in einem Calc-Dokument.*

```
Sub AccessSheets
  Dim oSheets       'Das Sheets-Objekt, das alle Tabellenblätter enthält.
  Dim oSheet        'Einzelnes Tabellenblatt.
  Dim oSheetEnum    'Zum Zugriff über Enumeration.
  Dim s As String   'Stringvariable für temporäre Daten.
  Dim i As Integer  'Indexvariable.
  Dim oDoc

  oDoc = CreateNewCalcDoc()
  oSheets = oDoc.Sheets

  REM Fügt ein neues Tabellenblatt als zweites Blatt ein.
  oSheets.insertNewByName("NeueTabelle", 1)

  REM Fügt ein neues Tabellenblatt mit dem Namen "Erste" ganz vorne ein.
  oSheets.insertNewByName("Erste", 0)

  REM Test, ob das Tabellenblatt "Tabelle3" existiert.
  REM AOO startet ein neues Tabellendokument mit 3 Tabellen,
  REM LO startet mit nur einer Tabelle.
  If Not oSheets.hasByName("Tabelle3") Then
    REM Fügt das Tabellenblatt ganz am Ende ein.
    oSheets.insertNewByName("Tabelle3", oSheets.getCount())
  End If
  REM Referenziert das Tabellenblatt und schreibt in die Zelle A1.
  oSheet = oSheets.getByName("Tabelle3")
  oSheet.getCellByPosition(0, 0).setString("Test")
  REM Kopiert "Tabelle3" ans Ende. Das ist eine Kopie, kein Verschieben!
  oSheets.copyByName("Tabelle3", "Kopie1", oSheets.getCount())

  If oSheets.hasByName("Tabelle1") Then
    oSheets.removeByName("Tabelle1")
  End If

  REM Der Index der Tabellenblätter startet mit null, aber getCount() gibt
  REM die genaue Anzahl der vorhandenen Blätter zurück.
  For i = 0 To oSheets.getCount() - 1
    s = s & "Tabelle " & i & " = " & oSheets.getByIndex(i).Name & Chr$(10)
  Next
```

```
  Msgbox s, 0, "Nach dem Einfügen neuer Tabellen"

  REM Nun werden zwei eingefügte Tabellen wieder gelöscht.
  oSheets.removeByName("Erste")
  oSheets.removeByName("Kopie1")

  s = "" : i = 0
  oSheetEnum = oSheets.createEnumeration()
  Do While oSheetEnum.hasMoreElements()
    oSheet = oSheetEnum.nextElement()
    s = s & "Tabelle " & i & " = " & oSheet.Name & Chr$(10)
    i = i + 1
  Loop
  Msgbox s, 0, "Nach dem Löschen von Tabellen"
End Sub
```

15.2. Tabellenzellen enthalten die Daten

Ein Tabellendokument enthält einzelne Tabellenblätter, in denen Zellen in Zeilen und Spalten angeordnet sind. Jede Spalte wird alphabetisch benannt, beginnend mit A. Für Tabellen mit vielen Spalten reicht das Alphabet natürlich nicht aus. Nach Z kommt AA, AB, AC, ..., AZ, BA, BB, ..., und nach ZZ kommt AAA und so weiter bis AMJ Das ist nämlich die Höchstanzahl von 1024 Spalten. Jede Zeile wird numerisch benannt, beginnend mit 1. Eine Zelle kann über ihren Namen, das heißt Spaltenbuchstabe und Zeilenzahl, oder über ihre Position in der Gitterstruktur identifiziert werden. Die Zelle oben links ist „A1“ an der Position (0, 0), und die Zelle „B3“ ist an der Position (1, 2).

Das Calc-GUI identifiziert Zellen mit solchen Namen wie „Tabelle2.D5“. Andererseits identifiziert sich eine Zelle selbst durch den Zeilen- und Spaltenoffset, der mit ein wenig Aufwand in eine für Menschen lesbare Form umgesetzt werden kann. Diese Aufgabe übernimmt der Service CellAddressConversion, der jedoch noch nicht dokumentiert ist, so dass ich auf Tests angewiesen war. Wenn eine Zelladresse der Eigenschaft Address zugewiesen wird (s. Listing 418), wird die Eigenschaft PersistentRepresentation den vollen Namen inklusive Tabellenname enthalten. Die Eigenschaft UserInterfaceRepresentation enthält den Tabellennamen jedoch nur, wenn es keine Zelle der aktiven Tabelle ist.

Listing 418. *Der Zellname über den Service CellAddressConversion.*

```
  Dim oConv   'Der Konvertierungsservice
  Dim oCell   'Eine Zelle
  oConv = ThisComponent.createInstance("com.sun.star.table.CellAddressConversion")
  oCell = ThisComponent.Sheets(2).getCellByPosition(0, 0) 'Zelle A1
  oConv.Address = oCell.getCellAddress()
  Print oConv.UserInterfaceRepresentation   'A1
  Print oConv.PersistentRepresentation      'Tabelle3.A1
```

Ich hatte so meine Probleme mit dem Service CellAddressConversion, daher schrieb ich meine eigene Funktion. Ihr werden die Indexwerte der Tabelle, der Spalte und der Zeile übergeben, und sie gibt eine Adresse zurück. Ich habe ein klein wenig geschummelt, denn ich habe vorausgesetzt, dass die Tabellen „Tabelle1“, „Tabelle2“ usw. heißen. Das kann sehr leicht korrigiert werden, ich habe es aber so gelassen. Der einzige Trick an diesem Code ist die Erkenntnis (für eine mathematisch ausreichend informierte Person), dass die Spaltenbezeichnungen auf einem 26-Zahlensystem mit den Ziffern A-Z basieren und nicht auf den Dezimalziffern 0-9.

Listing 419. *Formatierung einer einzelnen Zelladresse unter der Annahme, dass alle Tabellenblätter „Tabelle“ heißen.*

```
Function AddressString(iSheet As Long, iCol As Long, iRow As Long, _
  Dim s$
  'Zuerst die Spaltenangabe. Sie ist im 26-Zahlensystem
  'mit den Repräsentanten  A - Z aufgebaut.
```

```
    'Baut den String von hinten nach vorne auf. Chr(65) ist A.
    Do
      s = Chr$((iCol Mod 26) + 65) & s
      iCol = iCol \ 26 - 1
    Loop Until iCol = -1

    'Tabellen- und Zeilenzählung sind einfach die Zahlen als String.
    If bWithSheet  Then
      AddressString = "Tabelle" & CStr(iSheet + 1)  & "." & s & CStr(iRow + 1)
    Else
      AddressString = s & CStr(iRow + 1)
    End If
  End Function
```

Diese nun folgende Funktion akzeptiert eine Zellbereichsadresse (s. Tabelle 180), die hier vorweggenommen wird, genauso wie die verwendete Zelladresse (s. Tabelle 172).

Listing 420. *Formatierung eines einzelnen Zellbereichs unter der Annahme, dass alle Tabellenblätter „Tabelle“ heißen.*

```
Function PrettyRangeAddressName(oRangeAddr)
  REM Das Struct CellRangeAddress für einen Zellbereich enthält die Indexwerte für
  REM Tabellenblatt, Startspalte, Startzeile, Endspalte und Endzeile.

  Dim s1$     'Adressstring für den Anfang des Zellbereichs
  Dim s2$     'Adressstring für das Ende des Zellbereichs
  Dim oCellAddr As New com.sun.star.table.CellAddress
  REM Das Struct CellAddress für eine Einzelzelle enthält die Indexwerte für
  REM Tabellenblatt, Spalte und Zeile.

  s1 = AddressString(oRangeAddr.Sheet, oRangeAddr.StartColumn, _
       oRangeAddr.StartRow, True)
  s2 = AddressString(oRangeAddr.Sheet, oRangeAddr.EndColumn, oRangeAddr.EndRow, False)
  PrettyRangeAddressName = s1 & ":" & s2
End Function
```

15.2.1. Zelladresse

In OOo besteht eine Zelladresse aus drei Teilen der Lokalisierung: dem Tabellenblatt, der Spalte und der Zeile. OOo kapselt eine Zelladresse in das Struct CellAddress, s. Tabelle 172. CellAddress steht direkt aus einer Zelle zur Verfügung und wird auch als Argument für eine Reihe von Objektmethoden verwendet.

Tabelle 172. *Eigenschaften des Structs com.sun.star.table.CellAddress.*

Eigenschaft	Beschreibung
Sheet	Index des Tabellenblatts, in dem sich die Zelle befindet (Short Integer).
Column	Index der Spalte, in der sich die Zelle befindet (Long Integer).
Row	Index der Zeile, in der sich die Zelle befindet (Long Integer).

15.2.2. Zellinhalte

Eine Zelle kann viererlei Datentypen enthalten. Mit der Methode getType() findet man den Datentyp des Zellinhalts heraus.

- Eine Zelle, die keine Daten enthält, wird als leer betrachtet.

- Eine Zelle kann einen Wert vom Fließkommatyp Double enthalten. Mit den Methoden getValue() und setValue(Double) wird der Wert einer Zelle ausgelesen beziehungsweise geschrieben.

Tipp Auch wenn die Zelle eine Formel enthält, können Sie dennoch über die Eigenschaft FormulaResultType den in der Zelle verwendeten Datentyp bestimmen (s. Tabelle 177).

- Eine Zelle kann Text enthalten. Die Standardmethoden zum Lesen und Schreiben von Text sind die Methoden getString() und setString(String). Das ist aber nicht die ganze Geschichte, denn der Service com.sun.star.table.Cell bindet das Interface com.sun.star.text.XText ein. Das ist nämlich das zentrale Textinterface in Writer-Dokumenten und ermöglicht einzelnen Zellen, sehr komplexe Daten aufzunehmen.

Tipp Tabellenblattzellen unterstützen das Interface com.sun.star.text.XText. So ist es keine Überraschung, dass Zellen auch das Interface com.sun.star.text.XTextFieldsSupplier unterstützen – für den Fall, dass Sie besondere Textfelder in einer Zelle unterbringen wollen.

- Eine Zelle kann eine Formel enthalten. Formeln werden mit den Methoden getFormula() und setFormula() gelesen beziehungsweise geschrieben. Mit der Methode getError() erfährt man, ob die Formel einen Fehler produziert hat – ohne Fehler ist der Long-Integer-Rückgabewert null. Wie man den Zelltyp inspiziert, zeigt Ihnen das Makro im Listing 421.

Tipp Wenn Sie eine Zellformel schreiben, müssen Sie das führende Gleichheitszeichen (=) mit angeben, und die Formel muss in Englisch sein. Sie können aber die Formel auch in der Sprache Ihres Gebietsschemas angeben: das geht mit der in der Tabelle 177 aufgeführten Eigenschaft FormulaLocal.

Listing 421. *Den Zelltyp als String ermitteln.*

```
Function GetCellType(oCell) As String
  Select Case oCell.getType()
    Case com.sun.star.table.CellContentType.EMPTY
      GetCellType = "Leer"
    Case com.sun.star.table.CellContentType.VALUE
      GetCellType = "Numerisch"
    Case com.sun.star.table.CellContentType.TEXT
      GetCellType = "Text"
    Case com.sun.star.table.CellContentType.FORMULA
      GetCellType = "Formel"
    Case Else
      GetCellType = "Unbekannt"
  End Select
End Function
```

Listing 422 zeigt, wie man Zellinhalte schreibt und liest. Es werden je ein numerischer Wert, ein String und eine Formel in eine Zelle geschrieben. Nachdem so der Typ festgelegt ist, wird die Information über die Zelle ausgegeben (s. Bild 103). Das Makro im Listing 422 ist sehr einfach, zeigt aber manches sehr wichtige Verhalten. Sehen Sie sich im Bild 103 genau an, was für die verschiedenen Zellinhaltstypen mit getString(), getValue() und getFormula() zurückgegeben wird.

Listing 422. *Zellinformationen abrufen.*

```
Function SimpleCellInfo(oCell) As String
  SimpleCellInfo = oCell.AbsoluteName & " hat den Typ " &_
      GetCellType(oCell) & " String(" & oCell.getString() & ") Numerisch(" &_
      oCell.getValue()   & ") Formel(" & oCell.getFormula() & ")"
End Function
```

```
Sub GetSetCells
  Dim oCell          'Eine Zelle
  Dim s As String    'Ausgabestring
  Dim oDoc           'Neues Tabellendokument

  oDoc = CreateNewCalcDoc()

  oCell = oDoc.Sheets(0).getCellByPosition(0, 0)    'Zelle A1
  oCell.setString("Andy")
  oCell = oDoc.Sheets(0).getCellByPosition(104, 0) 'Zelle DA1. Ist noch leer.
  s = SimpleCellInfo(oCell) & Chr$(10)
  oCell.setValue(23.2)                          'Wert
  s = s & SimpleCellInfo(oCell) & Chr$(10)
  oCell.setString("4")                          'Text
  s = s & SimpleCellInfo(oCell) & Chr$(10)
  oCell.setFormula("=A1")                       'Formel
  s = s & SimpleCellInfo(oCell) & Chr$(10)
  oCell.setFormula("")                          'Leer
  s = s & SimpleCellInfo(oCell) & Chr$(10)
  MsgBox s, 0, "Zellwerte und -typen"
End Sub
```

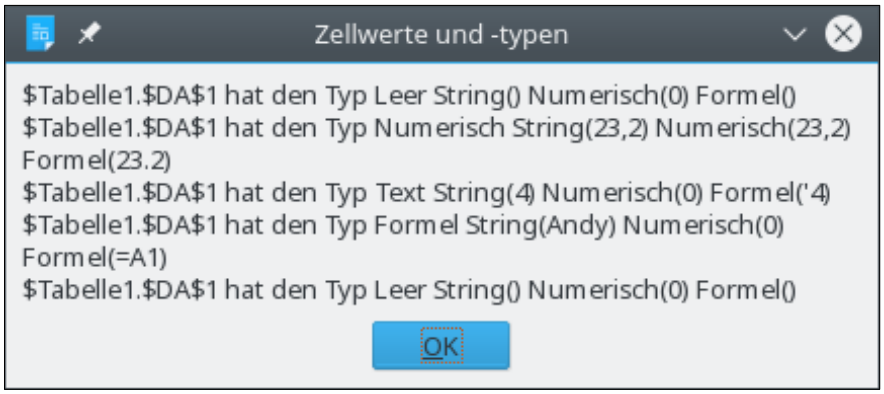


Bild 103. *Werte, die mit getType(), getString(), getValue() und getFormula() für verschiedene Inhaltstypen zurückgegeben werden.*

Tipp Die Methoden getString() und getFormula() geben relevante Werte zurück, auch wenn der Zelltyp nicht String oder Formel ist (s. Bild 103). Beachten Sie auch, dass wenn man 4 als String schreibt, kein numerischer Wert resultiert. Die Formelausgabe zeigt einen Apostroph vor der 4. Dies ist ein deutlicher Hinweis darauf, dass man auch so etwas tun kann wie setFormula("'Ich bin ein Text"), um Text in eine Zelle zu schreiben.

15.2.3. Zelleigenschaften

Zellen in einem Tabellenblatt sind durch den Service com.sun.star.sheet.SheetCell definiert, der eine Reihe von Eigenschaften zum Formatieren des Zellinhalts zur Verfügung stellt. Angesichts des Umstands, dass eine Zelle auch den Text-Service unterstützt, kann es nicht überraschen, dass auch Eigenschaften für textlichen Inhalt verfügbar sind, für Zeichen und Absätze: CharacterProperties, CharacterPropertiesAsian, CharacterPropertiesComplex und ParagraphProperties.

Das Format einer Zelle wird über den Service com.sun.star.table.CellProperties gesteuert, s. Tabelle 173.

Tabelle 173. *Eigenschaften im Service com.sun.star.table.CellProperties.*

Eigenschaft	Beschreibung
CellStyle	Optional. Der Name der Zellvorlage als String.
CellBackColor	Farbe des Zellhintergrunds als Long Integer (s. auch IsCellBackgroundTransparent).
IsCellBackgroundTransparent	Falls True, wird der Zellhintergrund transparent, und CellBackColor wird ignoriert.
HoriJustify	Horizontale Ausrichtung der Zelle als Wert der Enumeration com.sun.star.table.CellHoriJustify: • STANDARD – Links für Zahlen und rechts für Text. • LEFT – Am linken Zellrand. • CENTER – Zentriert. • RIGHT – Am rechten Zellrand. • BLOCK – Zu beiden Zellrändern. (Scheint nicht zu funktionieren.) • REPEAT – Ausgefüllt. Der Inhalt wird wiederholt, bis die Zelle gefüllt ist.
VertJustify	Vertikale Ausrichtung der Zelle als Wert der Enumeration com.sun.star.table.CellVertJustify: • STANDARD – Offenbar am unteren Rand. • TOP – Am oberen Zellrand. • CENTER – Zentriert. • BOTTOM – Am unteren Zellrand.
IsTextWrapped	Falls True, wird der Zellinhalt automatisch am rechten Zellrand umbrochen.
ParaIndent	Einzugsbreite des Zellinhalts als Short Integer in 1/100 mm.
Orientation	Schreibrichtung des Zellinhalts, wenn RotateAngle null ist. Als Wert der Enumeration com.sun.star.table.CellOrientation: • STANDARD – Von links nach rechts. • TOPBOTTOM – Von oben nach unten. • BOTTOMTOP – Von unten nach oben. • STACKED – Wie TOPBOTTOM, aber mit horizontal gestellten Zeichen.
RotateAngle	Rotationswinkel des Zellinhalts (in 1/100 Grad) als Long Integer. Der gesamte String wird als Einheit rotiert und nicht die einzelnen Zeichen.
RotateReference	Der Rand, an dem die rotierte Zelle ausgerichtet ist. In AOO als Wert der Enumeration com.sun.star.table.CellVertJustify (s.VertJustify). In LO als Wert der Konstantengruppe com.sun.star.table.CellVertJustify2.
AsianVerticalMode	Falls True, werden für den Fall, dass mit der Orientation-Eigenschaft die vertikale Schreibrichtung STACKED eingestellt ist, nur die asiatischen Schriftzeichen (CJK) horizontal gestellt, alle anderen Zeichen nicht. Diese Eigenschaft steht nur zur Verfügung, wenn in den Sprachoptionen die Unterstützung für asiatische Sprachen aktiviert ist.
TableBorder	Beschreibung der Umrandung für eine Einzelzelle oder einen Zellbereich (s.Tabelle 176). Wenn es ein Zellbereich ist, werden die Werte für oben, unten, links und rechts nur auf den äußeren Rand des Bereichs angewendet, nicht auf die einzelnen Zellen.
TopBorder	Beschreibung der oberen Umrandungslinie einer Zelle (s. Tabelle 174).
BottomBorder	Beschreibung der unteren Umrandungslinie einer Zelle (s. Tabelle 174).
LeftBorder	Beschreibung der linken Umrandungslinie einer Zelle (s. Tabelle 174).
RightBorder	Beschreibung der rechten Umrandungslinie einer Zelle (s. Tabelle 174).
DiagonalTLBR	Beschreibung der Diagonalen einer Zelle – von links oben nach rechts unten (s. Tabelle 174).

Eigenschaft	Beschreibung
DiagonalBLTR	Beschreibung der Diagonalen einer Zelle – von links unten nach rechts oben (s. Tabelle 174).
NumberFormat	Index des Zahlenformats der Zelle. Die Festlegung des Formats geschieht mit dem vom Dokument bereitgestellten Interface com.sun.star.util.XNumberFormatter.
ShadowFormat	Schattenformat als Struct com.sun.star.table.ShadowFormat: • Location – Die Position des Schattens als Wert der Enumeration com.sun.star.table.ShadowLocation: NONE (kein Schatten), TOP_LEFT (oben links), TOP_RIGHT (oben rechts), BOTTOM_LEFT (unten links)und BOTTOM_RIGHT (unten rechts). • ShadowWidth – Die Größe des Schattens als Short Integer. • IsTransparent – Falls True, ist der Schatten transparent. • Color – Die Farbe des Schattens als Long Integer.
CellProtection	Zellschutz als Struct com.sun.star.util.CellProtection: • IsLocked – Falls True, ist die Zelle gegen Änderungen geschützt. • IsFormulaHidden – Falls True, ist die Formel verborgen. • IsHidden – Falls True, ist die Zelle verborgen. • IsPrintHidden – Falls True, wird die Zelle nicht ausgedruckt.
UserDefinedAttributes	Weitere Eigenschaften über das Interface com.sun.star.container.XNameContainer.
ShrinkToFit	Falls True, wird der Zellinhalt auf die Zellgröße geschrumpft.
TableBorder2 [nur LO]	Erweiterte Beschreibung der Umrandung für eine Einzelzelle oder einen Zellbereich (s.Tabelle 176). Wenn es ein Zellbereich ist, werden die Werte für oben, unten, links und rechts nur auf den äußeren Rand des Bereichs angewendet, nicht auf die einzelnen Zellen.
TopBorder2 [nur LO]	Erweiterte Beschreibung der oberen Umrandungslinie einer Zelle (s. Tabelle 175).
BottomBorder2 [nur LO]	Erweiterte Beschreibung der unteren Umrandungslinie einer Zelle (s. Tabelle 175).
LeftBorder2 [nur LO]	Erweiterte Beschreibung der linken Umrandungslinie einer Zelle (s. Tabelle 175).
RightBorder2 [nur LO]	Erweiterte Beschreibung der rechten Umrandungslinie einer Zelle (s. Tabelle 175).
DiagonalTLBR2 [nur LO]	Erweiterte Beschreibung der Diagonalen einer Zelle – von links oben nach rechts unten (s. Tabelle 175).
DiagonalBLTR2 [nur LO]	Erweiterte Beschreibung der Diagonalen einer Zelle – von links unten nach rechts oben (s. Tabelle 175).

Umrandungen für Einzelzellen werden für jede Zellenbegrenzung gesondert durch das Struct BorderLine eingerichtet, s. Tabelle 174. Wird die Zelleigenschaft, zum Beispiel LeftBorder, auf einen Zellbereich angewendet, sind alle Zellen des Bereichs gleichermaßen betroffen. Die Eigenschaft TableBorder hingegen definiert über das Struct TableBorder, wie die äußeren und inneren Linien in einem Zellbereich angezeigt werden, s. Tabelle 176. Sie kann auch auf eine Einzelzelle angewendet werden.

Tabelle 174. *Eigenschaften des Structs com.sun.star.table.BorderLine.*

Eigenschaft	Beschreibung
Color	Farbe als Long Integer.
InnerLineWidth	Breite des inneren Teils einer Doppellinie (in 1/100 mm) als Short Integer – 0 für eine Einzellinie.
OuterLineWidth	Breite einer Einzellinie, oder Breite des äußeren Teils einer Doppellinie (in 1/100 mm) als Short Integer.

Eigenschaft	Beschreibung
LineDistance	Abstand zwischen dem inneren und dem äußeren Teil einer Doppellinie (in 1/100 mm) als Short Integer.

In AOO wird eine Doppellinie dadurch bestimmt, dass alle drei Eigenschaften OuterLineWidth, InnerLineWidth und LineDistance einen Wert größer als null haben. Ist nur OuterLineWidth größer als null, wird eine Einzellinie gezogen.

In LO sind Linienstile seit der Version 3.4 als numerische Werte in einer eigenen Konstantengruppe definiert. Das Struct BorderLine2 (s. Tabelle 175) bietet die notwendigen zusätzlichen Eigenschaften zu denen des ererbten Structs BorderLine. Die Zelleigenschaften TopBorder2 usw. für die Zellrahmen erwarten das Struct BorderLine2.

Tabelle 175. *Eigenschaften des Structs com.sun.star.table.BorderLine2 [nur LO].*

Eigenschaft	Beschreibung
LineStyle	Linienstil als Wert der Konstantengruppe com.sun.star.table.BorderLineStyle: NONE = 0x7FFF: Keine Linie. SOLID = 0: Durchgehende Linie. DOTTED = 1: Gepunktete Linie. DASHED = 2: Gestrichelte Linie. DOUBLE = 3: Doppelte durchgehende Linie. THINTHICK_SMALLGAP = 4: Doppellinie, außen dünn, innen dick, schmaler Zwischenraum. THINTHICK_MEDIUMGAP = 5: Doppellinie, außen dünn, innen dick, mittlerer Zwischenraum. THINTHICK_LARGEGAP = 6: Doppellinie, außen dünn, innen dick, breiter Zwischenraum. THICKTHIN_SMALLGAP = 7: Doppellinie, außen dick, innen, dünn schmaler Zwischenraum. THICKTHIN_MEDIUMGAP = 8: Doppellinie, außen dick, innen dünn, mittlerer Zwischenraum. THICKTHIN_LARGEGAP = 9: Doppellinie, außen dick, innen dünn, breiter Zwischenraum. EMBOSSED = 10: 3D-Linie, Prägeeffekt. ENGRAVED = 11: 3D-Linie, Gravureffekt. OUTSET = 12: Erhaben. INSET = 13: Eingedrückt. FINE_DASHED = 14: Fein gestrichelte Linie. DOUBLE_THIN = 15: Dünne Doppellinie, feste Breite, variabler Zwischenraum. DASH_DOT =16: Strich, Punkt im Wechsel. DASH_DOT_DOT = 17: Strich, Punkt, Punkt im Wechsel. BORDER_LINE_STYLE_MAX = 17: Der momentan höchste Konstantenwert.
LineWidth	Linienbreite als vorzeichenloser Long Integer, aus der die einzelnen Linienstile berechnet werden. Nur wenn LineWidth den Wert null hat, werden die BorderLine-Werte für OuterLineWidth, InnerLineWidth und LineDistance herangezogen. Da deren Auswirkungen auf die einzelnen Linienstile unterschiedlich sind, empfiehlt es sich, vor der Anwendung Tests durchzuführen.

Im folgenden Makro werden alle vier Umrandungen der Zellen eines Zellbereichs formatiert.

Listing 423. *Zellumrandungen setzen.*

```
Sub SetCalcBorder
  Dim oDoc     'Neues Tabellendokument
  Dim oSheet   'Ein Tabellenblatt
  Dim oCells   'Ein Zellbereich
  Dim oBorder  'Eine Randliniendefinition
  Dim bLO      'Läuft das Makro unter LO (True) oder AOO (False)?

  oDoc = CreateNewCalcDoc()
  oSheet = oDoc.Sheets(0)
  oCells = oSheet.getCellRangeByName("B2:C6")
  bLO = True   'Vermuten wir mal.

  'Die erweiterten Linienstile in LO werden über das Struct BorderLine2
  'gesteuert. Es erbt alle Eigenschaften vom Struct BorderLine.
```

```
'Im neuen Struct sind alle Werte auf 0 initialisiert.
oBorder = CreateUnoStruct("com.sun.star.table.BorderLine2")
If IsNull(oBorder) Then
  'AOO kennt nur das Struct BorderLine. oBorder ist daher ein Null-Objekt.
  bLO = False
  oBorder = CreateUnoStruct("com.sun.star.table.BorderLine")
End If

'Setzt die Farbe der Linie auf rot.
oBorder.Color = RGB(199, 50, 0)
'Der äußere Teil einer Doppellinie mit der Breite 0,4 mm.
oBorder.OuterLineWidth = 40
'Da InnerLineWidth und LineDistance den Wert 0 haben, wird in AOO
'eine Einzellinie gezeichnet.
'In LO wird die Einzellinie über eine BorderLineStyle-Konstante gesetzt.
'Dem linken Rand der Zellen wird die rote Einzellinie zugewiesen.
If bLO Then
  oBorder.LineStyle = com.sun.star.table.BorderLineStyle.SOLID
  oBorder.LineWidth = 40
  oCells.LeftBorder2 = oBorder
Else
  oCells.LeftBorder = oBorder
End If

'Setzt die Farbe auf blau.
oBorder.Color = RGB(0, 102, 199)
'Der innere Teil einer Doppellinie mit der Breite 0,25 mm.
oBorder.InnerLineWidth = 25
'Da LineDistance immer noch den Wert 0 hat, wird in AOO
'eine Einzellinie gezeichnet.
'In LO wird die Doppellinie über eine BorderLineStyle-Konstante gesetzt.
'Dem oberen Rand der Zellen wird die blaue Linie zugewiesen. In LO erhält
'der Zwischraum einen Standardwert.
If bLO Then
  oBorder.LineStyle = com.sun.star.table.BorderLineStyle.DOUBLE
  oCells.TopBorder2 = oBorder
Else
  oCells.TopBorder = oBorder
End If

'Der Abstand zwischen der inneren und der äußeren Linie: 0,35 mm.
oBorder.LineDistance = 35
'Sowohl der rechte als auch der untere Rand erhalten nun in Lo und AOO
'eine Doppellinie mit breiterem Zwischenraum.
If bLO Then
  oCells.RightBorder2 = oBorder
  oCells.BottomBorder2 = oBorder
Else
  oCells.RightBorder = oBorder
  oCells.BottomBorder = oBorder
End If

'Kopiert die linke Rahmenlinie einer Zelle.
'Da in LO die Eigenschaft LeftBorder2 die Eigenschaft LeftBorder erbt,
'funktioniert die folgende Zuordnung sowohl in AOO als auch in LO.
oBorder = oSheet.getCellRangeByName("C2").LeftBorder
```

```
    MsgBox "Rotanteil des linken Randes der Umrandung = " & Red(oBorder.Color), _
           0, "Randlinienfarbe"
End Sub
```

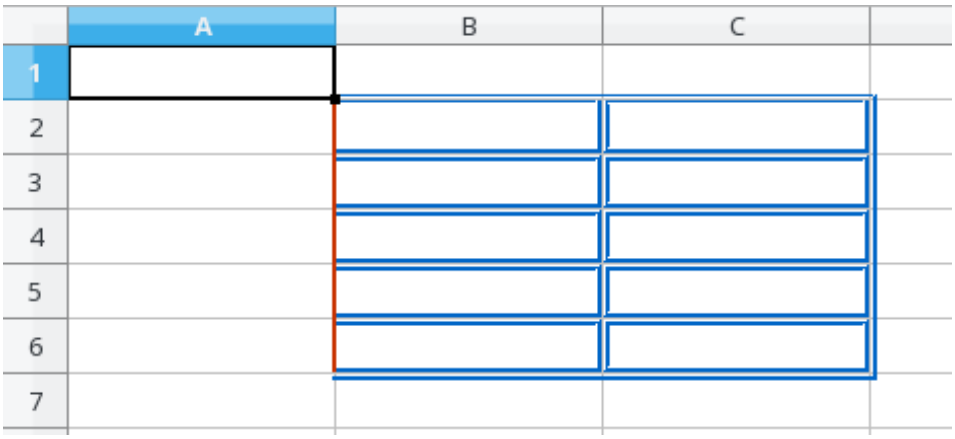


Bild 104. *Zellen mit Umrandungen (LO).*

Beachten Sie Folgendes:

- Das Struct BorderLine wird mit seinem Wert kopiert, nicht als Referenz. Daher muss man das Struct zur Modifizierung erst in eine Variable kopieren, danach die Variable modifizieren und schließlich das Struct zurück zur linken, rechten, oberen und unteren Umrandung kopieren.
- Man kann Zellumrandungen auch für alle Zellen eines Zellbereichs gleichzeitig setzen.
- Obwohl der linke Rand beider Zellen B2 und C2 aus einer roten Linie besteht, wird sie in der Zelle C2 nicht angezeigt (s. Bild 104). Wie entscheidet OOo nun, welcher Rand links, rechts, oben oder unten dargestellt wird? Obwohl das Verhalten nicht dokumentiert ist, kann man folgende Rangfolge beobachten:
  - Eine Umrandung mit Linienabstand ist wichtiger als eine ohne Linienabstand.
  - Obere und linke Umrandungen sind wichtiger als untere oder rechte.

Rahmen für komplette Zellbereiche werden in AOO über die Zelleigenschaft TableBorder gesetzt, die das entsprechende Struct TableBorder erwartet, s. Tabelle 176. Die Einstellungen für die einzelnen Linien ergeben sich aus dem Struct BorderLine (Tabelle 174).

Für LO gilt jedoch die Zelleigenschaft TableBorder2, das seinerseits das Struct TableBorder2 erwartet, in dem die Linienbeschreibungen aus dem erweiterten Struct BorderLine2 (Tabelle 175) stammen.

Die Eigenschaft TableBorder[2] kann auch für Einzelzellen eingesetzt werden.

Tabelle 176. *Eigenschaften des Structs com.sun.star.table.TableBorder [LO: TableBorder2].*

Eigenschaft	Beschreibung
TopLine	Linienstil des oberen Randes (AOO: s. Tabelle 174, LO: s. Tabelle 175).
IsTopLineValid	Falls True, wird TopLine verwendet.
BottomLine	Linienstil des unteren Randes (AOO: s. Tabelle 174, LO: s. Tabelle 175).
IsBottomLineValid	Falls True, wird BottomLine verwendet.
LeftLine	Linienstil des linken Randes (AOO: s. Tabelle 174, LO: s. Tabelle 175).
IsLeftLineValid	Falls True, wird LeftLine verwendet.
RightLine	Linienstil des rechten Randes (AOO: s. Tabelle 174, LO: s. Tabelle 175).
IsRightLineValid	Falls True, wird RightLine verwendet.
HorizontalLine	Linienstil für horizontale Linien zwischen den Zellen (AOO: s. Tabelle 174, LO: s. Tabelle 175).

Eigenschaft	Beschreibung
IsHorizontalLineValid	Falls True, wird HorizontalLine verwendet.
VerticalLine	Linienstil für vertikale Linien zwischen den Zellen (AOO: s. Tabelle 174, LO: s. Tabelle 175).
IsVerticalLineValid	Falls True, wird VerticalLine verwendet.
Distance	Abstand zwischen den Linien und anderem Inhalt als Short Integer.
IsDistanceValid	Falls True, wird Distance verwendet.

Wenn man in einem TableBorder-Struct Werte einstellt, werden nicht immer alle Werte benötigt. Wird zum Beispiel die Umrandung einer Zelle konfiguriert, werden die einzelnen Werte nur genutzt, wenn auch die entsprechende „Is...Valid“-Eigenschaft auf True gesetzt ist. So kann man eine einzelne Linie einstellen und die Werte für die anderen unangetastet lassen. Wenn andererseits ein TableBorder-Struct über eine Abfrage geholt wird (wenn also die Werte gelesen werden), zeigen die „Is...Valid“-Eigenschaften an, dass nicht alle Linien gleich aussehen.

Das folgende Makro Listing 424demonstriert, wie mit TableBorder ein Rahmen aus Doppellinien um einen Zellbereich herum und Einzellinien zwischen den Zellen gezogen werden.

Listing 424. *Tabellenumrandungen setzen.*

```
Sub SetCalcTableBorder
  Dim oDoc          'Neues Tabellendokument
  Dim oSheet        'Ein Tabellenblatt
  Dim oCells        'Ein Zellbereich
  Dim oBorder       'Eine Randliniendefinition
  Dim oTableBorder  'Eine Tabellenranddefinition
  Dim bLO           'Läuft das Makro unter LO (True) oder AOO (False)?

  oDoc = CreateNewCalcDoc()
  oSheet = oDoc.Sheets(0)
  oCells = oSheet.getCellRangeByName("B2:C6")
  bLO = True   'Vermuten wir mal.
  oDoc.CurrentController.Select(oCells) 'Selektiert den Zellbereich.

  'Die erweiterten Linienstile in LO werden über das Struct BorderLine2
  'gesteuert. Es erbt alle Eigenschaften vom Struct BorderLine.
  'Im neuen Struct sind alle Werte auf 0 initialisiert.
  oBorder = CreateUnoStruct("com.sun.star.table.BorderLine2")
  If IsNull(oBorder) Then
    'AOO kennt nur das Struct BorderLine. oBorder ist daher ein Null-Objekt.
    bLO = False
    oBorder = CreateUnoStruct("com.sun.star.table.BorderLine")
    oTableBorder = CreateUnoStruct("com.sun.star.table.TableBorder")
  Else
    oTableBorder = CreateUnoStruct("com.sun.star.table.TableBorder2")
    'Für LO: Eine durchgehende Einzellinie mit der Breite 0,3 mm.
    oBorder.LineStyle = com.sun.star.table.BorderLineStyle.SOLID
    oBorder.LineWidth = 30
  End If
  'Für AOO: Eine Einzellinie mit der Breite 0,3 mm.
  'Für LO gilt die Information auch, ist aber uninteressant, weil
  'LineWidth größer als 0 ist.
  oBorder.OuterLineWidth = 30
```

```
  With oTableBorder
    'Die horizontalen und vertikalen Linien zwischen den Zellen
    'sind Einzellinien.
    .HorizontalLine = oBorder
    .VerticalLine = oBorder

    'Definition einer Doppellinie.
    'Zuerst LO:
    If bLO Then
      oBorder.LineStyle = com.sun.star.table.BorderLineStyle.DOUBLE
      oBorder.LineWidth = 40
    End If
    'Dann AOO (auch LO):
    oBorder.OuterLineWidth = 40
    oBorder.InnerLineWidth = 25
    oBorder.LineDistance = 35

    'Die Außenlinien des Zellbereichs sind Doppellinien.
    .LeftLine = oBorder
    .TopLine = oBorder
    .RightLine = oBorder
    .BottomLine = oBorder

    'Die Außenlinien werden aktiviert.
    .IsLeftLineValid = True
    .IsTopLineValid = True
    .IsRightLineValid = True
    .IsBottomLineValid = True

    If bLO Then
      oCells.TableBorder2 = oTableBorder
    Else
      oCells.TableBorder = oTableBorder
    End If

    MsgBox "Tabellenrahmen ohne Zwischenlinien", 0, "Tabellenrahmen"

    'Nun werden zusätzlich auch die Innenlinien aktiviert.
    .IsHorizontalLineValid = True
    .IsVerticalLineValid = True

    If bLO Then
      oCells.TableBorder2 = oTableBorder
    Else
      oCells.TableBorder = oTableBorder
    End If

    MsgBox "Tabellenrahmen mit Zwischenlinien", 0, "Tabellenrahmen"
  End With
End Sub
```

Im allgemeinen ist das Setzen der Zelleigenschaften ein simpler Prozess. Das Makro im Listing 425 zeigt einige weitere Eigenschaften aus der Tabelle 173. In die Zelle B1 wird der Text „Hallo“ geschrieben, der dann zentriert und gegen den Uhrzeigersinn um 30 Grad gedreht wird.

Listing 425. *„Hallo“ wird zentriert und um 30 Grad gedreht.*

```
Sub RotateCellText
  Dim oCell
```

```
    Dim oDoc

    oDoc = CreateNewCalcDoc()
    oCell = oDoc.Sheets(0).getCellByPosition(1, 0) 'Zelle B1
    oCell.setString("Hallo")
    oCell.HoriJustify = com.sun.star.table.CellHoriJustify.CENTER
    oCell.RotateAngle = 3000  '30 Grad
End Sub
```

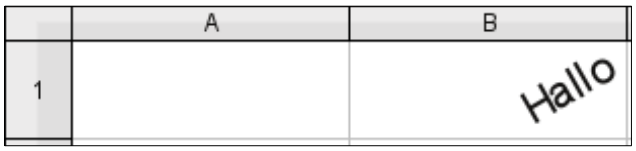


Bild 105. *Rotation eines Zellinhalts um 30 Grad.*

Die Zelleigenschaften aus der Tabelle 173 gelten allgemein für die meisten Zelltypen inklusive Zellen in Texttabellen. Für Zellen in Tabellenblättern gibt es zusätzliche Eigenschaften, s. Tabelle 177.

Tabelle 177. *Eigenschaften im Service com.sun.star.sheet.SheetCell.*

Eigenschaft	Beschreibung
Position	Die Position der Zelle im Tabellenblatt (in 1/100 mm) als Struct com.sun.star.awt.Point. Es ist die absolute Position im gesamten Blatt, nicht die Position im sichtbaren Bereich. • X – Die x-Koordinate als Long Integer. • Y – Die y-Koordinate als Long Integer.
Size	Die Größe der Zelle (in 1/100 mm) als Struct com.sun.star.awt.Size. • Width – Die Breite als Long Integer. • Height – Die Höhe als Long Integer.
FormulaLocal	Optionaler String der Formel mit Funktionsnamen und Zeichensetzung gemäß dem aktuellen Gebietsschema.
FormulaResultType	Formelergebnistyp als Wert der Konstantengruppe com.sun.star.sheet.FormulaResult: • VALUE = 1 – Fließkommazahl mit Genauigkeit Double. • STRING = 2 – String. • ERROR = 4 – Fehlermeldung.
ConditionalFormat	Die bedingten Formatierungen für diese Zelle als Interface com.sun.star.sheet.XSheetConditionalEntries. Wenn eine Formatbedingung geändert wird, muss sie in die Property-Liste neu eingefügt werden.
ConditionalFormatLocal	Optionale Eigenschaft, verwendet wie ConditionalFormat, aber mit Funktionsnamen gemäß dem aktuellen Gebietsschema.
Validation	Die Datengültigkeitsregeln für diese Zelle als com.sun.star.beans.XPropertySet.
ValidationLocal	Optionale Eigenschaft als Duplikat von Validation, aber mit Funktionsnamen gemäß dem aktuellen Gebietsschema.

Das Makro im Listing 426 zeigt, wie man Position und Größe einer Zelle im Tabellenblatt ausliest.

Listing 426. *Ausgabe der Zelldimensionen.*

```
Sub CellDimensions
  Dim oCell
  Dim s As String
  oCell = ThisComponent.Sheets(0).getCellByPosition(1, 0)            'Zelle B1
  s = s & CStr(oCell.Position.X \ 100)  & " mm von links" & Chr$(10)
  s = s & CStr(oCell.Position.Y \ 100)  & " mm von oben" & Chr$(10)
```

```
    s = s & CStr(oCell.Size.Width \ 100)  & " mm breit" & Chr$(10)
    s = s & CStr(oCell.Size.Height \ 100) & " mm hoch" & Chr$(10)
    MsgBox s, 0, "Zellgröße und -position " & oCell.AbsoluteName
End Sub
```

15.2.4. Zellkommentare

Jede Zelle kann einen Kommentar erhalten, der aus einfachem, unformatiertem Text besteht. Die Zellmethode getAnnotation() gibt ein Objekt zurück, das den Service com.sun.star.sheet.CellAnnotation unterstützt. Zellkommentare unterstützen das auch in Writer-Dokumenten verwendete Interface XSimpleText sowie die spezifischeren Methoden aus der Tabelle 178.

Tabelle 178. *Über den Service com.sun.star.sheet.CellAnnotation verfügbare Methoden.*

Methode	Beschreibung
getParent()	Zugriff auf das Objekt, zu dem dieser Kommentar gehört – die Zelle.
setParent(oCell)	Legt die Zelle fest, zu der dieser Kommentar gehört.
getPosition()	Die Adresse (com.sun.star.table.CellAddress) der Zelle, zu der dieser Kommentar gehört (s. Tabelle 172).
getAuthor()	Der Benutzer, der als letzter den Kommentarstring geändert hat.
getDate()	Das Datum (als formatierter String) der letzten Änderung des Kommentarstrings.
getIsVisible()	Gibt True zurück, wenn der Kommentar sichtbar ist.
setIsVisible(boolean)	Legt fest, ob der Kommentar sichtbar ist.

Das Tabellenblatt-Objekt unterstützt die Methode getAnnotations(), die alle Kommentare im Tabellenblatt zurückgibt. Den Zugriff auf einzelne Kommentare erhält man über den Index oder die Enumeration. Das von getAnnotations() zurückgegebene Objekt verfügt auch über die Methoden removeByIndex(n) und insertNew(CellAddress, String) zum Löschen und Einfügen eines Kommentars.

Listing 427. *Manipulation von Zellkommentaren.*

```
Sub ManipulateAnnotations
  Dim oDoc              'Tabellendokument
  Dim oSheet            'Tabellenblatt
  Dim oCell             'Eine Zelle
  Dim oAnnotations      'Alle Kommentare
  Dim oMyAnnotation     'Ein Kommentar

  oDoc = ThisComponent
  oSheet = oDoc.Sheets.getByIndex(0)    'Erste Tabelle

  'Fügt einen neuen Kommentar zur Zelle A1 ein.
  oCell = oSheet.getCellByPosition(0, 0)
  oAnnotations = oSheet.Annotations
  oAnnotations.insertNew(oCell.CellAddress, "Textinhalt")

  'Ändert einen vorhandenen Kommentar.
  'Die Pseudo-Eigenschaft "Annotation" der Zelle kann nur gelesen werden.
  oMyAnnotation = GetAnnotationByCell(oCell)
  oMyAnnotation.IsVisible = True  'Identisch mit oMyAnnotation.setIsVisible(True)
  oMyAnnotation.AnnotationShape.FillColor = RGB(148, 0, 107)    'Farbe
  oMyAnnotation.AnnotationShape.CharFontName = "Tahoma"         'Schriftart
  oMyAnnotation.AnnotationShape.CharHeight = 12                 'Höhe
End Sub

Function GetAnnotationByCell(oCell)
  Dim oResult                   'Suchergebnis
```

```
  Dim oAnnotations               'Alle Kommentare
  Dim lAnnotationIndex As Long   'Indexzählung der Kommentare
  Dim oAnAnnotation              'Ein Kommentar
  Dim bIsFound As Boolean        'Gefunden oder nicht?

  REM Die CellRange-Methode getSpreadsheet() gibt das Tabellenblattobjekt zurück.
  oAnnotations = oCell.SpreadSheet.Annotations
  'Suche nach einem Kommentar zur angegebenen Zelle. Alle Kommentare werden einzeln
  'über ihre Indexwerte aufgerufen, und ihre Positionswerte werden mit den Adresswerten
  'der Zelle verglichen. Wenn die Adressen übereinstimmen, ist die Suche erfolgreich.
  While ((lAnnotationIndex < oAnnotations.Count) And (Not bIsFound))
          oAnAnnotation = oAnnotations.getByIndex(lAnnotationIndex)
          bIsFound = ((oAnAnnotation.Position.Row = oCell.CellAddress.Row) And _
                      (oAnAnnotation.Position.Column = oCell.CellAddress.Column))
    If bIsFound Then
      oResult = oAnAnnotation
    End If
    lAnnotationIndex = lAnnotationIndex + 1
  Wend
  GetAnnotationByCell = oResult
End Function
```

15.3. Nicht übersetzte XML-Attribute

OOo speichert Dokumente im XML-Format. Wenn ein OOo-Dokument gelesen wird, wird es von einem XML-Parser zergliedert. Benutzerdefinierte Attribute werden vom Parser nicht übersetzt, sie werden einfach nur gelesen, gespeichert und dann geschrieben. Somit kann ein Parser Attribute speichern, die er selbst beim Lesen einer XML-Datei nicht verarbeiten kann. Wenn die Datei wieder gespeichert wird, können die unbekannten Attribute ohne Verlust zurückgeschrieben werden.

Nicht übersetzte Attribute bieten Ihnen die Möglichkeit, eigene Eigenschaften hinzuzufügen, die mit dem Dokument gespeichert werden. Benutzerdefinierte Attribute haben die Eigenschaften Namespace (XML-Namensraum als String), Type (XML-Typ als String) und Value (String).

Tabelle 179. *Methoden zur Manipulation benutzerdefinierter Attribute.*

Methode	Beschreibung
getByName(name)	Zugriff auf das genannte Attribut.
getElementNames	Stringarray mit den Namen der vorhandenen Attribute.
hasByName(name)	Falls True, ist das genannte Attribut vorhanden.
hasElements	Falls True, ist mindestens ein Attribut vorhanden.
insertByName(name, attr)	Fügt ein neues Attribut ein.
removeByName(name)	Entfernt ein vorhandenes Attribut.
replaceByName(name, attr)	Ersetzt ein vorhandenes Attribut.

Listing 428. *Benutzerdefinierte Attribute werden manipuliert, indem sie kopiert und wieder neu zugewiesen werden.*

```
Sub UserDefinedAttributeToCell
  Dim oCell       'Die Zelle, die das Attribut erhalten wird.
  Dim oUserData   'Kopie der UserDefinedAttributes.
  Dim oMyAttributeData As New com.sun.star.xml.AttributeData
  Dim oAttribute  'Ein Attribut
  Dim oDoc        'Ein neues Tabellendokument
```

```
    oDoc = CreateNewCalcDoc()
    REM Zuerst die Zelle, die das Attribut erhalten wird.
    oCell = oDoc.Sheets(0).getCellByPosition(1, 0) 'Zelle B1

    REM Nun werden dem Attribut die Daten gegeben.
    REM Namespace (URL des XML-Namensraums) wird normalerweise leer gelassen,
    REM aber Sie dürfen auch etwas eintragen.
    REM oMyAttribute.Namespace = "http://what/ever/you/want"

    REM Beachten Sie, dass der Typ CDATA ist und nicht so etwas wie "String".
    oMyAttributeData.Type = "CDATA"
    oMyAttributeData.Value = "Andrew Pitonyak"

    REM An dieser Stelle läuft häufig etwas falsch mit einer solchen Zeile:
    REM oCell.UserDefinedAttributes.insertByName("Andy", oMyAttributeData)
    REM Das funktioniert nie. Machen Sie stattdessen erst eine Kopie
    REM und bearbeiten Sie die Kopie.
    oUserData = oCell.UserDefinedAttributes
    If Not oUserData.hasByName("Andy") Then
      oUserData.insertByName("Andy", oMyAttributeData)
      oCell.UserDefinedAttributes = oUserData
    End If
    oAttribute = oCell.UserDefinedAttributes.getByName("Andy")
    Print oAttribute.Value
End Sub
```

Tipp	Alle Services, die UserDefinedAttributes unterstützen, funktionieren auf dieselbe Weise.

15.4. Zellbereiche in einem Tabellenblatt

In Writer-Dokumenten kann zusammenhängender Text als Textrange gruppiert werden. In einem Tabellenblatt können Zellen als SheetCellRange in rechteckige Bereiche gruppiert werden. Auf diese Weise können mehrere Zellen gleichzeitig bearbeitet werden. Der Service SheetCellRange unterstützt viele der auch von SheetCell unterstützten Interfaces und Eigenschaften.

Tip	Jedes Tabellenblatt in einem Tabellendokument ist auch ein SheetRange.

So wie jede Zelle eine Zelladresse hat (s. Tabelle 172), die ihre Position im Tabellendokument beschreibt, so wird auch die Position eines jeden Zellbereichs durch eine analoge Struktur beschrieben (s. Tabelle 180).

Tabelle 180. *Eigenschaften des Structs com.sun.star.table.CellRangeAddress.*

Eigenschaft	Beschreibung
Sheet	Short Integer. Index des Tabellenblatts, in dem die Zelle ist.
StartColumn	Long Integer. Index der Spalte des linken Randes.
StartRow	Long Integer. Index der Zeile des oberen Randes.
EndColumn	Long Integer. Index der Spalte des rechten Randes.
EndRow	Long Integer. Index der Zeile des unteren Randes.

Ein einzelner Zellbereich ist auf ein einzelnes Tabellenblatt beschränkt und enthält eine rechteckige Region. Multiple Bereiche werden vom Service SheetCellRanges gekapselt, der die meisten derselben Eigenschaften und Services unterstützt wie ein SheetCellRange. Die Ähnlichkeit der Funktionalität vereinfacht die Lernkurve und verdoppelt die ansonsten komplizierten Einsatzmöglichkeiten.

Der Service SheetCellRanges bietet den Zugriff auf jeden Einzelbereich über die Interfaces XElementAccess und XIndexAccess. Die enthaltenen Zellbereiche können auch über die Methoden in der Tabelle 181 angesprochen werden.

Tabelle 181. *Methoden im Interface com.sun.star.table.XSheetCellRanges.*

Methode	Beschreibung
getCells()	Gibt die Kollektion der enthaltenen Zellen als XEnumerationAccess zurück.
getRangeAddressesAsString()	Gibt einen String mit den Adressen aller enthaltenen Bereiche zurück, in der Form „Tabelle1.B2:D6;Tabelle3.C4:D5“.
getRangeAddresses()	Gibt ein Array von Services des Typs CellRangeAddress zurück (s. Tabelle 180).

15.4.1. Eigenschaften von Zellbereichen

Sowohl Zellen als auch Zellbereiche unterstützen die Eigenschaften Position, Size, ConditionalFormat, ConditionalFormatLocal, Validation und ValidationLocal (s. Tabelle 177). Die Eigenschaft Position enthält die Position der linken oberen Zelle – genauso, als wenn die Position-Eigenschaft dieser Zelle direkt abgefragt würde. Die Eigenschaft Size gibt für eine Zelle die Größe der Einzelzelle, für einen Zellbereich die Größe aller Zellen in diesem Bereich an.

Gültigkeitsregeln

Sowohl Zellen als auch Zellbereiche sind in der Lage, die Gültigkeit der enthaltenen Daten zu prüfen, um zu verhindern, dass unzulässige Daten in die Zellen gelangen. Sie können einen Dialog aufrufen, wenn eine unzulässige Eingabe erfolgt (s. Tabelle 183). Der Service com.sun.star.sheet.TableValidation kontrolliert den Gültigkeitsprüfungsprozess.

Befor ich zeige, wie die Gültigkeitsprüfung vonstatten geht, muss ich ein paar enumerierte Werte, Eigenschaften und Methoden vorstellen. Welche Gültigkeitsprüfung erfolgen soll, bestimmen Sie mit den Werten aus der Tabelle 182.

Tabelle 182. *Typen der Gültigkeitsprüfung: die Enumeration com.sun.star.sheet.ValidationType.*

Wert	Beschreibung
ANY	Jeder Inhalt ist gültig, keine Bedingungen.
WHOLE	Vergleicht eine Ganzzahl (Integer) mit der angegebenen Bedingung.
DECIMAL	Vergleicht jede Zahl mit der angegebenen Bedingung.
DATE	Vergleicht einen Datumswert mit der angegebenen Bedingung.
TIME	Vergleicht einen Uhrzeitwert mit der angegebenen Bedingung.
TEXT_LEN	Vergleicht eine Stringlänge mit der angegebenen Bedingung.
LIST	Erlaubt nur Strings aus der angegebenen Liste.
CUSTOM	Die Gültigkeit der Inhalte wird über eine angegebene Formel geprüft.

Wenn eine Zelle mit unzulässigen Daten gefunden wird, bestimmt die Enumeration ValidationAlertStyle, wie die ungültigen Daten behandelt werden sollen (s. Tabelle 183).

Tabelle 183. *Ungültigkeitswarnungen: die Enumeration com.sun.star.sheet.ValidationAlertStyle.*

Wert	Beschreibung
STOP	Ausgabe einer Fehlermeldung und Zurückweisung der Zelländerung.
WARNING	Ausgabe einer Warnmeldung mit der Frage an den Benutzer, ob die Zelländerung akzeptiert wird. Die Standardantwort ist Nein.

Wert	Beschreibung
INFO	Ausgabe einer Informationsmeldung mit der Frage an den Benutzer, ob die Zelländerung akzeptiert wird. Die Standardantwort ist Ja.
MACRO	Aufruf eines spezifizierten Makros.

Tabelle 184 listet die verfügbaren Vergleichsoperatoren auf.

Tabelle 184. *Vergleichsoperatoren: die Enumeration com.sun.star.sheet.ConditionOperator.*

Wert	Beschreibung
NONE	Keine Bedingung.
EQUAL	Der Wert muss gleich dem angegebenen Wert sein.
NOT_EQUAL	Der Wert muss ungleich dem angegebenen Wert sein.
GREATER	Der Wert muss größer als der angegebene Wert sein.
GREATER_EQUAL	Der Wert muss größer als oder gleich groß wie der angegebene Wert sein.
LESS	Der Wert muss kleiner als der angegebene Wert sein.
LESS_EQUAL	Der Wert muss kleiner als oder gleich groß wie der angegebene Wert sein.
BETWEEN	Der Wert muss im Bereich zwischen zwei angegebenen Werten liegen.
NOT_BETWEEN	Der Wert muss außerhalb des Bereichs zweier angegebener Werte liegen.
FORMULA	Die angegebene Formel muss einen Wert ungleich null ergeben.

Das Validation-Objekt definiert über die Objekteigenschaften den Überprüfungstyp und die Modalitäten der Überprüfung, s. Tabelle 185.

Tabelle 185. *Eigenschaften im Service com.sun.star.sheet.TableValidation.*

Eigenschaft	Beschreibung
Type	Der Überprüfungstyp, s. Tabelle 182.
ShowInputMessage	Falls True, erscheint ein Eingabedialog, wenn der Cursor in einer Zelle mit ungültigem Wert ist.
InputTitle	Titel (String) des Dialogs mit der Eingabemeldung.
InputMessage	Text (String) der Eingabemeldung.
ShowErrorMessage	Falls True, erscheint eine Fehlermeldung, wenn ein ungültiger Wert eingegeben wird.
ErrorTitle	Titel (String) des Dialogs mit der Fehlermeldung.
ErrorMessage	Text (String) der Fehlermeldung.
IgnoreBlankCells	Falls True, sind leere Zellen erlaubt.
ErrorAlertStyle	Die Aktion beim Auftreten eines Fehlers, s. Tabelle 183.

Schließlich wird der durchzuführende Vergleich mit den vom Service TableValidation eingesetzten Methoden spezifiziert, s. Tabelle 186.

Tabelle 186. *Über den Service com.sun.star.sheet.TableValidation verfügbare Methoden.*

Methode	Beschreibung
getOperator()	Liest den im Vergleich benutzten Operator aus (s. Tabelle 184).
setOperator(condition)	Legt den im Vergleich benutzten Operator fest (s. Tabelle 184).
getFormula1()	Liest den im Vergleich benutzten Wert aus (String) oder den ersten Wert, wenn zwei nötig sind.
setFormula1(String)	Legt den im Vergleich benutzten Wert fest (String) oder den ersten Wert, wenn zwei nötig sind.

Methode	Beschreibung
getFormula2()	Liest den zweiten Wert aus (String), wenn zwei nötig sind.
setFormula2(String)	Legt den zweiten Wert fest (String), wenn zwei nötig sind.
getSourcePosition()	Liest die Zelladresse (CellAdress) aus, die als Basis für relative Verknüpfungen in den Formeln dient (s. Tabelle 172).
setSourcePosition(CellAddress)	Legt die Zelladresse (CellAdress) fest, die als Basis für relative Verknüpfungen in den Formeln dient (s. Tabelle 172).

Das Makro im Listing 429 legt im ersten Tabellenblatt – das heißt, im Blatt mit dem numerischen Index 0 – einen Bereich zur Gültigkeitsprüfung fest. Den Zellen von B2 bis D6 sind nur Werte erlaubt, die zwischen 1 und 10 liegen. Das Makro selbst ist erstaunlich einfach.

Listing 429. *Gültigkeitsprüfung in Calc.*

```
Sub SetValidationRange
  Dim oRange        'Bereich, der eine Gültigkeitsprüfung erhält
  Dim oValidation   'Das Validation-Objekt
  Dim oDoc          'Neues Tabellendokument

  oDoc = CreateNewCalcDoc()

  REM Tabellenblätter bieten die Möglichkeit, einen Zellbereich
  REM mit den GUI-Namen festzulegen.
  oRange = oDoc.Sheets(0).getCellRangeByName("B2:D6")

  REM Zugriff auf das Validation-Objekt.
  oValidation = oRange.Validation

  REM Die Gültigkeitsprüfung wird konfiguriert.
  oValidation.Type = com.sun.star.sheet.ValidationType.DECIMAL
  oValidation.ErrorMessage = "Bitte geben Sie eine Zahl zwischen 1 und 10 ein."
  oValidation.ShowErrorMessage = True
  oValidation.ErrorAlertStyle = com.sun.star.sheet.ValidationAlertStyle.STOP
  oValidation.setOperator(com.sun.star.sheet.ConditionOperator.BETWEEN)

  oValidation.setFormula1(1.0)
  oValidation.setFormula2(10.0)

  REM Nun wird die Validation-Eigenschaft gespeichert.
  oRange.Validation = oValidation
  Print "Versuchen Sie, in den Zellen B2:D6 eine Zahl einzugeben, " & _
        "die nicht zwischen 1 und 10 liegt."
End Sub
```

Bedingte Formatierung

Bedingte Formatierung lässt Ihnen die vom Zellinhalt abhängige Wahl der Zellvorlage. Sowohl Zellen als auch Zellbereiche unterstützen die Eigenschaft ConditionalFormat, die ihrerseits das Interface XSheetConditionalEntries unterstützt. Sie können auf die bedingten Formatierungseinträge über den Element-Zugang, den indexierten Zugang oder die Methoden der Tabelle 187 zugreifen.

Tabelle 187. *Methoden im Interface com.sun.star.sheet.XSheetConditionalEntries.*

Methode	Beschreibung
addNew(Array von PropertyValues)	Fügt der Zelle oder dem Zellbereich ein bedingtes Format als PropertyValues hinzu: Operator – s. Tabelle 184 Formula1 – Wert oder Formel, als String Formula2 – Wert oder Formel, als String. Wird nur verwendet mit den Operatoren BETWEEN und NOT_BETWEEN. SourcePosition – Die Zelladresse (CellAdress), die als Basis für relative Verknüpfungen in den Formeln dient (s. Tabelle 172). StyleName – Der Name der Zellformatvorlage. Die Vorlage muss schon existieren.
clear()	Löscht alle bedingten Formate.
removeByIndex(index)	Entfernt ein bestimmtes bedingtes Format.

Sie können durchaus mehrere bedingte Formatierungseinträge auf eine Zelle anwenden. Die Vorlage der ersten zutreffenden Bedingung wird angewendet. Jeder Formatierungseintrag wird durch ein Array von Property-Structs definiert. Bedingte Formatierung hat große Ähnlichkeit mit der Gültigkeitsprüfung.

Die Gültigkeitsprüfung ist eine relativ simple auf ein Datenelement angewendete Bedingung und erzwingt Typen- und Formatbeschränkungen. Bedingte Formatierung verwendet eine Gültigkeitsprüfung und einen erweiterten Satz von Kontrollen, einschließlich Attribute oder Metadaten zu Elementen oder Elementlisten – die Namen sind gleich, aber die tatsächlichen Operationen, Einsatzbereiche und Implikationen sind ungleich vielfältiger. Es ist viel schwieriger zu erklären als zu demonstrieren. Das Makro im Listing 430 richtet einen Zellbereich so ein, dass die Zellvorlage „Überschrift1“ verwendet wird, wenn die Zelle eine negative Zahl enthält.

Listing 430. *Ein bedingtes Format in Calc setzen.*

```
Sub SetConditionalStyle
  Dim oRange        'Zu nutzender Zellbereich.
  Dim oConFormat    'Objekt des bedingten Formats.
  Dim oCondition(2) As New com.sun.star.beans.PropertyValue 'Bedingung

  REM Tabellenblätter bieten die Möglichkeit, einen Zellbereich
  REM mit den GUI-Namen festzulegen.
  oRange = ThisComponent.Sheets(0).getCellRangeByName("B2:D6")

  REM Die bedingte Formatierung wird konfiguriert.
  oConFormat = oRange.ConditionalFormat

  oCondition(0).Name = "Operator"
  oCondition(0).Value = com.sun.star.sheet.ConditionOperator.LESS
  oCondition(1).Name = "Formula1"
  oCondition(1).Value = "0"        'Der Wert muss als String angegeben sein!
  oCondition(2).Name = "StyleName"
  oCondition(2).Value = "Überschrift1"
  oConFormat.addNew(oCondition())
  oRange.ConditionalFormat = oConFormat
End Sub
```

Das Beispiel oben arbeitet mit einem mathematischen Operator und einem festen Wert. Formula1 könnte aber auch eine Formel enthalten, zum Beispiel:

```
oCondition(1).Name = "Formula1"
oCondition(1).Value = "SUM($G$2:$G$3)"  'für ConditionalFormat
```

Diese Formel müssen Sie in der originalen englischen Form angeben. Wenn Ihnen die deutsche Form verständlicherweise aber geläufiger ist und Sie sicher sind, dass das Makro nur mit einem deutsch-

sprachigen Gebietsschema genutzt wird, so verwenden Sie statt ConditionalFormat die Zelleigenschaft ConditionalFormatLocal. Die entsprechenden Zeilen lauten dann:

```
oCondition(1).Name = "Formula1"
oCondition(1).Value = "SUMME($G$2:$G$3)"  'für ConditionalFormatLocal
```

Das folgende Listing 431 mit dem Operator FORMULA und deutschsprachiger Formel färbt die Zellen im Bereich A1:D4 unter der Bedingung, dass deren Werte restlos durch 3 teilbar sind. Diese Bedingung ist wahr, wenn die Gleichung der Formel wahr ist, wenn also nach der Division durch 3 als Rest 0 bleibt.

Listing 431. *Ein bedingtes Format mit einer Formel in Deutsch.*

```
Sub SetConditionalStyleWithGermanFormula
  Dim oRange      'zu formatierender Zellbereich
  Dim oConFormat  'Objekt des bedingten Formats
  Dim oCondition(2) as New com.sun.star.beans.PropertyValue 'Bedingung

  oRange = ThisComponent.Sheets(0).getCellRangeByName("A1:D4")
  oConFormat = oRange.ConditionalFormatLocal
  oCondition(0).Name = "Operator"
  oCondition(0).Value = com.sun.star.sheet.ConditionOperator.FORMULA
  oCondition(1).Name = "Formula1"
  oCondition(1).Value = "REST(INDIREKT(ADRESSE(ZEILE();SPALTE()));3) = 0"
  oCondition(2).Name = "StyleName"
  oCondition(2).Value = "Achtung"   'Muss vorher angelegt worden sein.
  oConFormat.addNew(oCondition())
  oRange.ConditionalFormatLocal = oConFormat
End Sub
```

	A	B	C	D
1	1	2	3	4
2	5	6	7	8
3	9	10	11	12
4	13	14	15	16

Bild 106. *Anwendung des bedingten Formats in Listing 431.*

Gelegentlich werden Sie den Wunsch haben, bedingte Formatierungen wieder zu löschen. Dabei ist zu beachten, das Format nach der Änderung wieder in die Properties der Zelle oder des Zellbereichs zurückzuschreiben. Das Löschen aller bedingten Formatierungen funktioniert also folgendermaßen:

Listing 432. *Bedingte Formate in Calc löschen.*

```
Sub ClearConditionalStyle
  Dim oRange       'Zu nutzender Zellbereich.
  Dim oConFormat   'Das ConditionalFormat-Objekt

  oRange = ThisComponent.Sheets(0).getCellRangeByName("B2:D6")
  oConFormat = oRange.ConditionalFormat
  oConFormat.clear()
  oRange.ConditionalFormat = oConFormat  'Wichtig: neu festlegen!
End Sub
```

In bestimmten Versionen der 4-Reihe von LO scheint ein Bug zu sein, durch den die Formatierung nicht zurückgesetzt wird. Für diesen Fall hat Roland Scheuerich den Tipp bereitgestellt, die Formate mit clearContents() zu löschen:

```
oRange.clearContents(64)    '64 = com.sun.star.sheet.CellFlags.STYLES
```

15.4.2. Services für Zellbereiche

Zellen und Zellbereiche haben zahlreiche Services gemeinsam – zum Beispiel CellProperties (s. Tabelle 173), CharacterProperties, CharacterPropertiesAsian, CharacterPropertiesComplex, ParagraphProperties und SheetRangesQuery.

Zugriff auf Zellen und Zellbereiche

Ein SheetCellRange unterstützt den Service CellRange, der seinerseits CellProperties (s. Tabelle 173) unterstützt. Der Service CellRange bietet zusätzliche Funktionalitäten, die sich für Zellbereiche, aber nicht für einzelne Zellen eignen – zum Beispiel auf Zellen und Zellbereiche zuzugreifen. Wenn man mit den Methoden aus der Tabelle 188 einen Zellbereich auswählt, werden die Zellen relativ zur oberen linken Ecke des Bereichs indexiert. Wenn der Bereich ein ganzes Tabellenblatt umfasst, bezieht sich die Position (0, 0) auf die Zelle „A1“. Wenn der Bereich jedoch die Zellen „B2:D6“ umfasst, bezieht sich die Position (0, 0) auf die Zelle „B2“. Die Makros in den Listings 422 bis 426 verwenden alle die Methode getCellByPosition().

Tabelle 188. *Methoden im Interface com.sun.star.table.XCellRange.*

Methode	Beschreibung
getCellByPosition(links, oben)	Zugriff auf eine Zelle innerhalb des Bereichs.
getCellRangeByPosition(links, oben, rechts, unten)	Zugriff auf einen Zellbereich innerhalb des Bereichs.
getCellRangeByName(name)	Namensbasierter Zugriff auf einen Zellbereich innerhalb des Bereichs. Der String enthält das Standardformat für den direkten Bezug auf die Zellen – etwa „B2:D5“ oder „$B$2“ - oder festgelegte Namen für Zellbereiche.

Tipp Die Methoden getCellByPosition(), getCellRangeByPosition() und getCellRangeByName() können nur Objekte zurückgeben, die sich im Bereich befinden (s. Listing 460).

Zellabfrage

Zellbereiche und Einzelzellen bieten gleichermaßen die Voraussetzungen, Zellen mit bestimmten Eigenschaften zu suchen. Dazu gehört auch ein Mechanismus, mit dem alle Zellen gefunden werden, die von der Formel der aktuellen Zelle referenziert werden (Vorgänger), sowie alle Zellen, die die aktuelle Zelle referenzieren (Nachfolger). Bei einer Abfrage auf der Basis des Zellinhalts wird der Zellinhaltstyp mit den in der Tabelle 189 aufgeführten Signalen (CellFlags) bestimmt.

Tabelle 189. *Die Konstantengruppe com.sun.star.sheet.CellFlags.*

Wert	Flag	Beschreibung
1	com.sun.star.sheet.CellFlags.VALUE	Zahlen, die nicht als Datum oder Uhrzeit formatiert sind.
2	com.sun.star.sheet.CellFlags.DATETIME	Zahlen, die als Datum oder Uhrzeit formatiert sind.
4	com.sun.star.sheet.CellFlags.STRING	Strings.
8	com.sun.star.sheet.CellFlags.ANNOTATION	Zellkommentare.
16	com.sun.star.sheet.CellFlags.FORMULA	Formeln.
32	com.sun.star.sheet.CellFlags.HARDATTR	Hart, nicht über Formatvorlagen formatierte Zellen.
64	com.sun.star.sheet.CellFlags.STYLES	Formatvorlagen.
128	com.sun.star.sheet.CellFlags.OBJECTS	Zeichnungsobjekte wie Schaltflächen und Grafiken.
256	com.sun.star.sheet.CellFlags.EDITATTR	Formatierungen innerhalb des Zellinhalts.

Jede der Methoden zur Abfrage eines Zellbereichs (s. Tabelle 190) gibt wiederum einen Zellbereich zurück (s. Tabelle 181).

Tabelle 190. *Methoden für Zellbereichsabfragen.*

Methode	Beschreibung
queryDependents(boolean)	Gibt alle Zellen zurück, die Zellen in diesem Bereich referenzieren (Nachfolger). Falls True, ist die Suche rekursiv.
queryPrecedents(boolean)	Gibt alle Zellen zurück, die von Zellen in diesem Bereich referenziert werden (Vorgänger). Falls True, ist die Suche rekursiv.
queryVisibleCells()	Gibt alle sichtbaren Zellen zurück.
queryEmptyCells()	Gibt alle leeren Zellen zurück.
queryContentCells(CellFlags)	Gibt alle Zellen des angegebenen Inhaltstyps zurück (s. Tabelle 189).
queryFormulaCells(FormulaResult)	Gibt alle Zellen zurück, die eine Formel mit dem angegebenen Ergebnistyp enthalten (s. FormulaResultType in der Tabelle 177).
queryColumnDifferences(CellAddress)	Vergleicht jede Zelle, die sich in der Zeile der angegebenen Adresse befindet, mit den Zellen in der jeweiligen Spalte und gibt die Zellen zurück, die sich von ihr unterscheiden (s. Tabelle 172).
queryRowDifferences(CellAddress)	Vergleicht jede Zelle, die sich in der Spalte der angegebenen Adresse befindet, mit den Zellen in der jeweiligen Zeile und gibt die Zellen zurück, die sich von ihr unterscheiden (s. Tabelle 172).
queryIntersection(CellRangeAddress)	Gibt den Zellbereich zurück, der sich sowohl im aktuellen Bereich als auch im angegebenen Bereich befindet (s. Tabelle 180).

Suche nach nicht-leeren Zellen in einem Bereich

Mit der Methode queryContentCells(CellFlags) werden alle Zellen in einem Bereich zurückgegeben, die nicht leer sind. Das CellFlags-Argument legt fest, ob es Zellen sind, die einen Wert, einen String, eine Formel oder Datum/Uhrzeit enthalten. Listing 433 ist nicht deshalb so interessant, weil darin nicht-leere Zellen durch eine Abfrage gefunden werden, sondern dadurch, dass gezeigt wird, wie man die Zellen aus einem Abfrageergebnis extrahiert und alle zurückgegebenen Zellen enumeriert: wie man also auf alle nicht-leeren Zellen eines bestimmten Bereichs zugreift.

Listing 433. *Suche in einem Bereich nach Zellen, die nicht leer sind.*

```
Function NonEmptyCellsInRange(oRange, sep$) As String
  Dim oCell          'Die zu verwendende Zelle
  Dim oRanges        'Die nach der Zellabfrage zurückgegebenen Bereiche
  Dim oAddrs()       'CellRangeAddress-Array der Adressen der Zellbereiche
  Dim oAddr          'Eine einzelne Bereichsadresse
  Dim oSheet         'Tabellenblatt, das den Zellbereich enthält
  Dim i As Long      'Indexvariable
  Dim nRow As Long   'Zeilennummer
  Dim nCol As Long   'Spaltennummer
  Dim s As String

  REM Zuerst werden die Zellen im Bereich gesucht, die nicht leer sind.
  REM Dabei wird eine Zelle als nicht leer betrachtet, die entweder
  REM einen Wert, einen String, eine Formel oder Datum/Uhrzeit enthält.
  oRanges = oRange.queryContentCells(_
    com.sun.star.sheet.CellFlags.VALUE Or _
    com.sun.star.sheet.CellFlags.DATETIME Or _
    com.sun.star.sheet.CellFlags.STRING Or _
    com.sun.star.sheet.CellFlags.FORMULA)

  oAddrs() = oRanges.getRangeAddresses()
```

```
    For i = 0 To UBound(oAddrs())
      REM Eine spezifische Bereichsadresse.
      oAddr = oAddrs(i)
      For nRow = oAddr.StartRow To oAddr.EndRow
        For nCol = oAddr.StartColumn To oAddr.EndColumn
          REM Die CellRange-Methode getSpreadsheet() gibt das
          REM Tabellenblattobjekt zurück.
          REM oRange.getCellByPosition(nCol, nRow) wäre ein relativer
          REM Bezug innerhalb des Zellbereichs.
          oCell = oRange.Spreadsheet.getCellByPosition(nCol, nRow)
          s = s & oCell.AbsoluteName & sep$
        Next
      Next
    Next

    NonEmptyCellsInRange = s
End Function
```

Komplexe Zellabfragen

Obwohl die Abfragemethoden einfach zu benutzen sind, so sind manche doch konzeptionell kompliziert. All diese Methoden geben ein SheetCellRanges-Objekt zurück. Das Makro im Listing 434 zeigt, wie man mit der Methode queryPrecedents() die von einer Formel referenzierten Zellen findet, wie man mit der Methode queryDependents() die Zellen findet, die eine bestimmte Zelle referenzieren, und wie man Zeilen- und Spaltenunterschiede mit den Methoden queryRowDifferences() und queryColumnDifferences() findet.

Das Makro im Listing 434 trägt in das erste Tabellenblatt Werte und Formeln ein und führt dann Abfragen durch. Der Code für die Demonstrationsabfragen ist einfach und kurz. Der Code allerdings, der die abzufragenden Daten erzeugt, ist komplizierter und lehrreich. Ganz speziell zeigt das Makro auch, wie man mit der Methode clearContents() einen Zellbereich leert. In Bild 107 sehen Sie den vom Listing 434 produzierten Dialog.

Listing 434. *Abfrage eines Zellbereichs, um referenzierte, abhängige und abweichende Zellen zu finden.*

```
Sub PopulateSheetForQuery(oDoc)
  Dim oCell         'Temporäre Zelle
  Dim oCellAddress  'Zelladresse
  Dim oRange        'Der zugrunde liegende Zellbereich
  Dim oSheet        'Das erste Tabellenblatt
  Dim i As Integer

  oSheet = oDoc.Sheets(0)

  REM Der zu nutzende Bereich.
  oRange = oSheet.getCellRangeByName("A1:F8")

  REM Über die CellFlags werden nun alle Attribute und Werte gelöscht.
  REM Beachten Sie, dass alle Flags mit Or kombiniert sind.
  oRange.clearContents(_
    com.sun.star.sheet.CellFlags.VALUE Or _
    com.sun.star.sheet.CellFlags.DATETIME Or _
    com.sun.star.sheet.CellFlags.STRING Or _
    com.sun.star.sheet.CellFlags.ANNOTATION Or _
    com.sun.star.sheet.CellFlags.FORMULA Or _
    com.sun.star.sheet.CellFlags.HARDATTR Or _
    com.sun.star.sheet.CellFlags.STYLES Or _
    com.sun.star.sheet.CellFlags.OBJECTS Or _
```

```
    com.sun.star.sheet.CellFlags.EDITATTR)

  For i = 1 To 5
    oCell = oSheet.getCellByPosition(1, i)     'Zellen B2 bis B6
    oCell.setValue(i)
    oCell = oSheet.getCellByPosition(2, i)        'Zellen C2 bis C6
    oCell.setFormula("=B" & CStr(i + 1) & " + 1") '=B2+1, =B3+1, ...
    oCell = oSheet.getCellByPosition(3, i)        'Zellen D2 bis D6
    oCell.setFormula("=C" & CStr(i + 1) & " - 1") '=C2-1, =C3-1, ...
  Next
  oCell = oSheet.getCellByPosition(1, 6)     'B7
  oCell.setFormula("=SUM(B2:B5)")            'B6 wird ignoriert.
  oCell = oSheet.getCellByPosition(2, 6)     'C7
  oCell.setFormula("=SUM(C2:C6)")            'C6 ist eingeschlossen.

  oCell = oSheet.getCellByPosition(2, 0)     'C1
  oCell.setFormula("=B1 - 1")
  oCell = oSheet.getCellByPosition(1, 0)     'B1
  oCell.setValue(2)
  oCell = oSheet.getCellByPosition(1, 7)     'B8
  oCell.setValue(2)
  oCell = oSheet.getCellByPosition(4, 2)     'E3
  oCell.setValue(2)
  oCell = oSheet.getCellByPosition(4, 7)     'E8
  oCell.setValue(2)
End Sub

Sub QueryCellRange
  Dim oCell          'Temporäre Zelle
  Dim oCellAddress   'Zelladresse
  Dim oRange         'Der zugrunde liegende Zellbereich
  Dim oSheet         'Das erste Tabellenblatt
  Dim i As Integer   'Indexvariable
  Dim s As String    'Temporärer String
  Dim oDoc           : oDoc = CreateNewCalcDoc()

  PopulateSheetForQuery(oDoc)

  oSheet = oDoc.Sheets(0)

  REM Der zu nutzende Bereich.
  oRange = oSheet.getCellRangeByName("A1:F8")
  s = "C7"

  REM Die Abfrage findet die Zellen "C2:C7".
  REM Beachten Sie, dass die Zelle selbst dazugehört.
  oCell = oSheet.getCellByPosition(2, 6)
  s = s & "=SUMME(C2:C6) referenziert direkt: " &_
     oCell.queryPrecedents(False).getRangeAddressesAsString() & Chr$(10)

  REM Die Abfrage findet die Zellen "B2:B6;C2:C7".
  s = s & "=SUMME(C2:C6) referenziert direkt und indirekt: " &_
     oCell.queryPrecedents(True).getRangeAddressesAsString() & Chr$(10)

  REM Findet direkte und indirekte Referenzierungen auf die Zelle B3.
  oCell = oSheet.getCellByPosition(1, 2)     'B3
```

```
    s = s & "Zellen, die B3 referenzieren: " &_
       oCell.queryDependents(True).getRangeAddressesAsString() & Chr$(10)

    oCellAddress = oCell.CellAddress
    s = s & "Spaltenunterschiede für B3: " &_
       oRange.queryColumnDifferences(oCellAddress).getRangeAddressesAsString()
    s = s & Chr$(10)

    s = s & " Zeilenunterschiede for B3: " &_
       oRange.queryRowDifferences(oCellAddress).getRangeAddressesAsString()
    s = s & Chr$(10)

    MsgBox s, 0, "Abfrage eines Zellbereichs"
End Sub
```

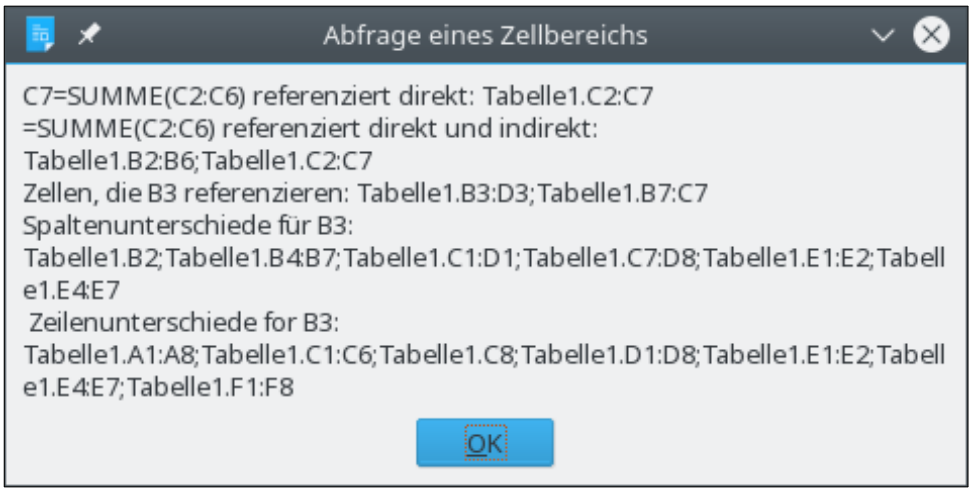


Bild 107. *Zellbereichsabfrage, um referenzierte, abhängige und abweichende Zellen zu finden.*

Tabelle 191 zeigt die Formeln und Werte, die das Makro im Listing 434 erzeugt hat, zum besseren Verständnis der Ausgabe im Bild 107.

Tabelle 191. *Die vom Listing 434 erzeugten Formeln und Werte.*

	B	C	D	E
1	2	=B1-1		
2	1	=B2+1	=C2-1	
3	2	=B3+1	=C3-1	2
4	3	=B4+1	=C4-1	
5	4	=B5+1	=C5-1	
6	5	=B6+1	=C6-1	
7	=SUMME(B2:B5)	=SUMME(C2:C6)		
8	2			2

Vorgänger und Nachfolger suchen

Die erste Zeile im Bild 107 zeigt das Resultat von queryPrecedents(False) auf einen Bereich, der nur aus der einzigen Zelle C7 besteht. Wie man in der Tabelle 191 sehen kann, referenziert C7 direkt die Zellen C2:C6 und sich selbst. Mit dem Aufruf von queryPrecedents(True) – der zweiten Zeile im Bild 107 – werden auch B2:B6 hinzugefügt, denn die Zellen in der Spalte C referenzieren die Zellen in der Spalte B.

Die Methode queryDependents(True) liefert die dritte Zeile im Bild 107 und gibt alle Zellen aus, die die Zelle B3 referenzieren, direkt oder indirekt. Tabelle 191 zeigt, dass die Zelle B3 von den Zellen B3, B7 und C3 direkt und von den Zellen C7 und D3 indirekt referenziert wird.

Spaltenunterschiede suchen

Die vierte Zeile im Bild 107 listet die „Spaltenunterschiede“ basierend auf Zelle B3 auf. Bei der Berechnung der Spaltenunterschiede zählt nur die Zeile. Das heißt, dass dasselbe Ergebnis herauskommt, wenn anstelle von B3 eine der Zellen A3, C3, D3 oder E3 genommen würde. Bild 108 zeigt das Tabellenblatt mit den als unterschiedlich gewerteten Zellen vor schwarzem Hintergrund. Diese Darstellung soll illustrieren, was und was nicht als Unterschied zu betrachten ist.

Mit der Wahl der Zelle B3 werden alle Zellen in jeder Spalte (innerhalb des Bereichs) mit der Zelle verglichen, die sich in der jeweiligen Spalte in der dritten Reihe befindet. In der ersten Spalte, A, sind keine Zellen hervorgehoben, denn alle sind sie leer, daher haben sie alle denselben Wert. In der Spalte B hat die Zelle B3 den Wert 2, genau wie die Zellen B1 und B8. Also werden diese beiden Zellen nicht als unterschiedlich gewertet. Spalte C ist sehr interessant, denn sie enthält keine Konstanten, sondern ähnlich strukturierte Formeln. Die Zellen C2, C4, C5 und C6 ähneln der Zelle C3. Die Formeln sind tatsächlich gleich (s. Tabelle 191) in dem Sinne, dass sie alle den Wert 1 zum Wert der Zelle links daneben addieren. Die Formel in C1 ist der in C3 nicht ähnlich, also wird sie in die Ausgabeliste mit aufgenommen. In Spalte D ist es wie in Spalte C, und in Spalte E ist es wie in Spalte B. Als Übung können Sie einmal die Zeilenunterschiede nach dem Muster von Bild 108 erklären. Erstaunlich ist, dass LO (5.3) im Gegensatz zu AOO die Zelle C7 nicht als unterschiedlich erkennt.

	A	B	C	D	E
1		2	1		
2		1	2	1	
3		2	3	2	2
4		3	4	3	
5		4	5	4	
6		5	6	5	
7		10	20		
8		2			2

Bild 108. *Ausgabe vom Listing 434 mit hervorgehobenen „Spaltenunterschieden“.*

15.4.3. Suchen und ersetzen

Der für mich interessanteste Aspekt der Suche in einem Tabellendokument ist, dass die Suche nicht vom Dokument-Objekt unterstützt wird, wohl aber von Zellobjekten und Zellbereichsobjekten. Jedes Tabellenblatt ist auch ein Zellbereich, somit kann ein gesamtes Tabellenblatt durchsucht werden. Ein gesamtes Dokument kann man jedoch nicht in einem Vorgang durchsuchen. Man muss Tabellenblatt für Tabellenblatt vorgehen. Listing 435 zeigt das Suchen und Ersetzen in einem einzelnen Tabellenblatt.

Listing 435. *Einfaches Suchen und Ersetzen in Calc.*

```
Sub SearchSheet
  Dim oSheet    'Tabellenblatt, in dem das Ersetzen stattfindet.
  Dim oReplace  'Ersetzen-Deskriptor.
  Dim nCount    'Anzahl der Ersetzungen.

  oSheet = ThisComponent.Sheets(0)
  oReplace = oSheet.createReplaceDescriptor()
  oReplace.setSearchString("Xyzzy")                'Der Suchstring
  oReplace.setReplaceString("Irgendwas anderes")   'Der Ersetzungsstring
  oReplace.SearchWords = False                     'Nicht nur ganze Zellen
  nCount = oSheet.replaceAll(oReplace)             'Alle Treffer auf einmal ersetzen
```

```
    MsgBox nCount & " Ersetzungen"
End Sub
```

Ein Tabellenblatt in einem Calc-Dokument zu durchsuchen, ist fast identisch mit der Textsuche in einem Writer-Dokument. In einem Calc-Dokument bewirkt True für die Eigenschaft SearchWords, dass die gesamte Zelle aus dem Suchtext bestehen muss. In einem Writer-Dokument heißt es, dass der Text nicht Teil eines weiteren Wortes sein darf.

Tipp In einem Calc-Dokument bewirkt True für die Eigenschaft SearchWords, dass die gesamte Zelle aus dem Suchtext bestehen muss.

15.4.4. Zellen verbinden

Mit der Methode merge(True) wird ein Zellbereich verbunden und mit merge(False) wieder geteilt. Wenn der Bereich B2:D7 verbunden wird, erscheint die Zelle B2 in dem Gebiet, das vorher vom gesamten Bereich eingenommen wurde. Was das Bild 109 nicht zeigt, ist, dass die nicht sichtbaren Zellen immer noch existieren und erreichbar sind, aber einfach nicht dargestellt werden. Mit der Methode getIsMerged() stellen Sie fest, ob alle Zellen in einem Bereich verbunden sind.

Listing 436. *Einen Zellbereich verbinden.*

```
Sub MergeExperiment
  Dim oCell          'Temporäres Zellobjekt
  Dim oRange         'Der primäre Bereich
  Dim oSheet         'Das erste Tabellenblatt

  REM Verbindet einen Zellbereich.
  oSheet = ThisComponent.Sheets(0)
  oRange = oSheet.getCellRangeByName("B2:D7")
  oRange.merge(True)

  REM Nun greifen wir auf eine verbundene Zelle zu.
  REM Es geht!
  oCell = oSheet.getCellByPosition(2, 3)      'C4
  Print oCell.getValue()
End Sub
```

	A	B	C	D	E
1		2	1		
2					
3					2
4					
5					
6					
7				1	
8		2			2

Bild 109. *Zellen verbinden: die Zelle links oben nutzt den gesamten verbundenen Zellbereich.*

15.4.5. Spalten und Zeilen: Zugriff, Einfügen und Löschen

Der Zugriff auf die Spalten und Zeilen, die von Zellen und Zellbereichen belegt sind, geschieht über die Methoden getColumns() und getRows(). Nachdem Sie die Spalten eines Bereichs oder einer Zelle abgerufen haben, können Sie auf die einzelnen Spalten über die Interfaces XElementAccess oder XIndexAccess zugreifen. Über zwei weitere Methoden – insertByIndex(index, anzahl) und removeByIndex(index, anzahl) – können Sie Spalten einfügen und entfernen. Nachdem Sie eine einzelne Spalte referenziert haben, können Sie den Spaltennamen mit getName() abrufen, Eigenschaften für die gesamte Spalte festlegen und Zellen mit den CellRange-Methoden der Tabelle 188 extrahieren.

Alle die für Spalten beschriebenen Manipulationen gelten auch – mit Ausnahme von getName() – für die Zeilen, auf die man mit der Methode getRows() zugreift. Sie unterscheiden sich in den Eigenschaften der einzelnen Spalten und Zeilen, s. Tabelle 192.

Tabelle 192. *Die Eigenschaften einzelner Zeilen und Spalten.*

Typ	Eigenschaft	Beschreibung
Spalte	Width	Breite der Spalte (in 1/100 mm) als Long Integer.
Zeile	Height	Höhe der Zeile (in 1/100 mm) als Long Integer.
Spalte	OptimalWidth	Falls True, behält die Spalte immer ihre optimale Breite.
Zeile	OptimalHeight	Falls True, behält die Zeile immer ihre optimale Höhe.
Beide	IsVisible	Falls True, ist die Zeile oder die Spalte sichtbar.
Beide	IsStartOfNewPage	Falls True, wird der Spalte (Zeile) ein horizontaler (vertikaler) Seitenumbruch beigefügt.

Das Makro im Listing 437 holt sich den Bereich „B6:C9“ und iteriert über die nicht-leeren Zellen. Die eigentliche Arbeit wird von der Routine im Listing 433 erledigt. Listing 437 zeigt jedoch, wie man aus einem Bereich die einzelnen Zeilen extrahiert. Normalerweise kennen Sie beim Schreiben eines Makros die Datenpositionen, so dass Ihr Code einfach die Daten direkt durchläuft. Bild 110 zeigt die Ausgabe des Listing 437, wenn das Makro mit den Daten aus Bild 108 läuft.

Listing 437. *Enumeriert Zeilen in einem Bereich und sucht nicht-leere Zellen.*

```
Sub TraverseRows
  Dim oRange        'Der primäre Bereich
  Dim oSheet        'Das erste Tabellenblatt
  Dim oRows         'Das Zeilen-Objekt
  Dim oRow          'Eine einzelne Zeile
  Dim oRowEnum      'Enumerator für die Zeilen
  Dim s As String   'Ausgabestring

  oSheet = ThisComponent.Sheets(0)
  oRange = oSheet.getCellRangeByName("B6:C9")

  REM Ich will nun ALLE nicht-leeren Zellen in den Zeilen finden,
  REM die den Bereich durchlaufen. Beachten Sie, dass ich die Suche
  REM nicht auf den Bereich begrenzen will, aber ich interessiere mich für die Zeilen.
  oRows  = oRange.getRows()
  REM Sicher, ich könnte über den Index zugreifen,
  REM aber das haben Sie sicher auch erwartet!
  oRowEnum = oRows.createEnumeration()
  Do While oRowEnum.hasMoreElements()
    oRow = oRowEnum.nextElement()
    s = s & NonEmptyCellsInRange(oRow, "  ") & Chr$(10)
  Loop
  MsgBox s, 0, "Nicht-leere Zellen in Zeilen"
End Sub
```

Bild 110. *Die nicht-leeren Zellen werden aus den Zeilen 6 bis 8 angezeigt.*

15.4.6. Daten als Array lesen und schreiben

Man kann schnell und einfach alle Daten aus einem Bereich mit der Methode getDataArray() als Array von Arrays auslesen. Darin repräsentiert das äußere Array die Zeilen und das innere Array die Spalten. Der Inhalt einer jeden Zelle wird entweder als Zahl oder als String zurückgegeben. Sie können gleichermaßen die Daten in die Zellen eines Bereichs mit der Methode setDataArray() eingeben. Sie müssen nur darauf achten, dass die Dimensionierung des Arrays der des Zellbereichs entspricht (s. Listing 438).

Listing 438. *Daten in einem Calc-Tabellenblatt lesen und schreiben.*

```
Sub GetAndSetData
  Dim oRange         'Der primäre Bereich
  Dim oSheet         'Das erste Tabellenblatt
  Dim oAllData       'Array mit den Daten
  Dim s As String    'Allgemeine Stringvariable
  Dim i As Integer   'Allgemeine Indexvariable

  oSheet = ThisComponent.Sheets(0)
  oRange = oSheet.getCellRangeByName("B6:E8")

  REM Liest die Daten aus den Zellen im Bereich aus,
  REM einschließlich der leeren Zellen.
  oAllData = oRange.getDataArray()
  For i = 0 To UBound(oAllData)
    REM oAllData(i) ist ein Array, so kann man die Daten einfach mit Join
    REM für eine schnelle Ausgabe verketten.
    s = s & "(" & Join(oAllData(i), ") (") & ")" & Chr$(10)
  Next
  MsgBox s, 0, "Daten im Zellbereich"

  REM Nun werden auf die Schnelle ein paar Daten geschrieben.
  oRange = oSheet.getCellRangeByName("F1:G2")
  oRange.setDataArray(Array(Array(1, "Eins"), Array(2, "Zwei")))
End Sub
```

Zellbereiche bieten auch die Möglichkeit, Formeln in einem Schwung über Arrays einzugeben. Mit den Methoden getFormulaArray() und setFormulaArray() können Sie die Formeln zeitsparend aus Zellbereichen in Arrays lesen oder aus Arrays in Zellbereiche schreiben.

Wenn Sie einen Zellbereich inspizieren, werden sie neben der Methode getDataArray() auch getData() finden. Beide Methoden geben ein Array von Arrays zurück: getDataArray liefert Zahlen und Strings, wohingegen getData – vom Interface XChartData definiert – nur Zahlen liefert und Strings ignoriert.

15.4.7. Funktionsberechnungen auf einen Zellbereich anwenden

Es ist möglich, auf der Basis eines bestimmten Bereichs einen Wert zu errechnen (s. Listing 433 und Listing 437), aber es ist mühsam. Alternativ kann man mit der Methode getDataArray (s. Listing 438) schnell alle Daten auslesen und sie dann in den verschachtelten Arrays verarbeiten. Sie können aber auch ganz leicht eine einfache Funktion auf den gesamten Bereich anwenden: mit der Objektmethode computeFunction(GeneralFunction). Tabelle 193 zeigt Ihnen die von der Enumeration GeneralFunction bereitgestellten Funktionen.

Tabelle 193. *Die Enumeration com.sun.star.sheet.GeneralFunction.*

Wert	Beschreibung
NONE	Keine Berechnung.
AUTO	Verwendet SUM, wenn alle Werte numerisch sind, ansonsten COUNT.
SUM	Summe (Addition) aller numerischen Werte.
COUNT	Anzahl aller Werte.
AVERAGE	Durchschnitt aller numerischen Werte.
MAX	Der größte numerische Wert.
MIN	Der kleinste numerische Wert.
PRODUCT	Produkt (Multiplikation) aller numerischen Werte.
COUNTNUMS	Anzahl der numerischen Werte.
STDEV	Standardabweichung basierend auf einer Stichprobe.
STDEVP	Standardabweichung basierend auf der Grundgesamtheit.
VAR	Varianz basierend auf einer Stichprobe.
VARP	Varianz basierend auf der Grundgesamtheit.

Das Makro im Listing 439 zeigt den Einsatz der Berechnungsfunktion. In dem Tabellenblatt, das im Bild 108 gezeigt wird, findet das Makro heraus, dass im Bereich A5:C9 sieben nicht-leere Zellen sind.

Listing 439. *Anwendung einer Funktion auf einen Zellbereich.*

```
Sub UseCompute
  Dim oRange         'Der primäre Bereich
  Dim oSheet         'Das erste Tabellenblatt
  Dim d As Double    'Rückgabewert

  oSheet = ThisComponent.Sheets(0)
  oRange = oSheet.getCellRangeByName("A5:C9")
  d = oRange.computeFunction(com.sun.star.sheet.GeneralFunction.COUNT)
  MsgBox "Nicht-leere Zellen im Bereich A5:C9 = " & d, 0, "ComputeFunction()"
End Sub
```

15.4.8. Zellen und Zellbereiche leeren

In einem Calc-Dokument klickt man auf eine Zelle und drückt die Entf-Taste (in manchen Versionen die Backspace-Taste), um einen Dialog zu öffnen mit einer Liste von Dingen, die gelöscht werden können. Diese Möglichkeit, jede Kombination unterschiedlicher Inhalts- und Formatierungsaspekte zum Löschen auszuwählen, ist erstaunlich mächtig und flexibel. Die löschbaren Elemente sind in den CellFlags gekapselt (s. Tabelle 189). Diese CellFlags können mit dem Operator Or kombiniert werden und so der Methode clearContents(CellFlags) übergeben werden, s. Listing 434.

Tip Die Methode clearContents() wird von Zellen, Zellbereichen und sogar Zeilen und Spalten unterstützt.

15.4.9. Zellen automatisch mit Daten füllen

Listing 434 schreibt mühselig Daten nacheinander in Zellen. Zellbereiche bieten eine bessere Methode, einen Bereich automatisch mit Daten auszufüllen: die Methode fillAuto(FillDirection, nCount). Die Werte der Enumeration FillDirection (s. Tabelle 194) steuern die Abfolge der Daten.

Tabelle 194. *Die Enumeration com.sun.star.sheet.FillDirection.*

Wert	Beschreibung
TO_BOTTOM	Die Zeilen werden von oben nach unten ausgefüllt.
TO_RIGHT	Die Spalten werden von links nach rechts ausgefüllt.
TO_TOP	Die Zeilen werden von unten nach oben ausgefüllt.
TO_LEFT	Die Spalten werden von rechts nach links ausgefüllt.

Mit der Methode fillAuto(FillDirection, nCount) wird automatisch ein Zellbereich ausgefüllt. Als erstes wählen Sie den auszufüllenden Bereich aus. Als zweites setzen Sie den Initialwert, der von fillAuto() inkrementiert werden soll. Die Position der Initialwerte hängt von der Füllrichtung ab. Bei der Richtung TO_LEFT zum Beispiel müssen die Zellen ganz rechts im Bereich einen Initialwert haben, damit nach links ausgefüllt werden kann.

Beim Ausfüllen mit neuen Werten vergrößert die Methode fillAuto() die Zahl um 1 in der Richtung nach rechts oder nach unten, verringert die Zahl hingegen um 1 in der Richtung nach links oder nach oben. Sollte der Initialwert als Uhrzeit oder Datum formatiert sein, wird der Wert jeweils um 1 Tag vergrößert oder verringert.

Tipp Wenn eine Uhrzeit mit fillAuto() inkrementiert wird, wird sie um 1 Tag vergrößert. Wenn die Zelle so formatiert ist, dass nur die Uhrzeit angezeigt wird, scheint sich der Wert nicht zu verändern – tut er aber doch. Wenn Sie darüber nachdenken, wird es Ihnen einleuchten.

Das letzte Argument zu fillAuto(), nCount, bestimmt, um wie viele Zellen der Cursor vor dem Eintragen eines Wertes weiterbewegt wird. Der Wert 1 bewirkt daher, dass jede Zelle gefüllt wird. Die Methode fillAuto() verlässt niemals den Bereich, in dem sie aufgerufen wurde. Der Codeschnipsel im Listing 440 geht davon aus, dass im Zellbereich E11:E20 der Tabelle1 numerische Werte stehen.

Listing 440. *Bereich E11:N20 automatisch mit fillAuto ausfüllen.*

```
oSheet = ThisComponent.Sheets(0)
oRange = oSheet.getCellRangeByName("E11:N20")
oRange.fillAuto(com.sun.star.sheet.FillDirection.TO_RIGHT, 1)
```

OOo unterstützt auch kompliziertere Füllmethoden. Die Enumeration FillMode (s. Tabelle 195) steuert die von der Methode fillSeries() gebotene Funktionalität. Die Methode fillAuto() nutzt immer den LINEAR-Modus zum Vergrößern des Wertes um 1, wohingegen die Methode fillSeries() beliebige Füllmodi akzeptiert.

Tabelle 195. *Die Enumeration com.sun.star.sheet.FillMode.*

Wert	Beschreibung
SIMPLE	Die Werte bleiben immer gleich (konstante Reihe).
LINEAR	Die Werte ändern sich durch konstante Addition (arithmetische Reihe).
GROWTH	Die Werte ändern sich durch konstante Multiplikation (geometrische Reihe).
DATE	Arithmetische Reihe für Datumswerte. Hierbei werden alle Zahlen als Datumsangaben gewertet, unabhängig von der Formatierung.

Wert	Beschreibung
AUTO	Die Zellen werden aus einer benutzerdefinierten Liste ausgefüllt.

Die Methode fillSeries() erkennt Datumswerte und Uhrzeiten an dem für die Darstellung verwendeten Zahlenformat. Im FillMode DATE werden jedoch alle Zahlen als Datumsangaben angesehen und nicht nur Zahlen, die als Datum formatiert sind. Beim Ausfüllen eines Datums kann je nach dem gewählten Wert der Enumeration FillDateMode entweder der Tag oder der Monat oder das Jahr angepasst werden (s.Tabelle 196).

Tabelle 196. *Die Enumeration com.sun.star.sheet.FillDateMode.*

Wert	Beschreibung
FILL_DATE_DAY	Plus 1 Tag.
FILL_DATE_WEEKDAY	Plus 1 Tag. Samstag und Sonntag werden übersprungen.
FILL_DATE_MONTH	Plus 1 Monat. Der Tag bleibt unverändert.
FILL_DATE_YEAR	Plus 1 Jahr. Tag und Monat bleiben unverändert.

Mit der Methode fillSeries(FillDirection, FillMode, FillDateMode, nStep, nEndValue) gewinnen Sie die größte Flexibilität für die auszufüllenden Werte. Das Argument nStep bestimmt, wie der Wert von Zelle zu Zelle modifiziert wird. Mit dem letzten Argument geben Sie einen Schlusswert an, der beim Ausfüllen nicht überschritten werden darf. Die Methode fillSeries() ändert keinen Wert außerhalb des definierten Bereichs und stoppt die Ausführung, wenn der Schlusswert erreicht ist. Wenn Sie einen Schlusswert angeben, bedenken Sie, dass ein Datum als reguläre Zahl ziemlich groß ist: 42000 ist der 27. Dezember 2014. Listing 441 füllt einen Zellbereich mit der Methode fillSeries() aus.

Tipp Die Methode fillAuto() vergrößert oder verringert den Wert in Abhängigkeit von der Ausfüllrichtung. Die Methode fillSeries() hingegen verwendet immer den Wert von nStep, ohne Rücksicht auf die Richtung.

Listing 441. *Bereich E11:N20 mit fillSeries() ausfüllen.*

```
oSheet = ThisComponent.Sheets(0)
oRange = oSheet.getCellRangeByName("E11:N20")
oRange.fillSeries(com.sun.star.sheet.FillDirection.TO_LEFT,_
  com.sun.star.sheet.FillMode.LINEAR,_
  com.sun.star.sheet.FillDateMode.FILL_DATE_DAY, 2, 42000)
```

Wird als Initialwert ein Text gefunden, der eine Zahl enthält, wird der Text kopiert, und die am weitesten rechts stehende Zahl wird aufgerechnet. Wenn zum Beispiel der Text „Text 1“ in der Zelle steht, dann werden die ausgefüllten Zellen „Text 3“, „Text 5“ usw. enthalten.

15.4.10. Matrixformeln

Die einfachste Anwendung einer Matrixformel, die ich kenne, besteht darin, eine Matrixformel in eine Zelle einzugeben und diese Formel in mehreren Zellen zu verwenden. Nachdem ich nun genug Einzelheiten ausgebreitet habe, um für ordentliche Verwirrung zu sorgen, betrachten Sie das einfache Beispiel in der Tabelle 197.

Tabelle 197. *Eine einfache Formel in Spalte I.*

	F	G	H	I	J	K
3		1	3	=G3+H3		
4		2	4	=G4+H4		

	F	G	H	I	J	K
5		3	5	=G5+H5		
6		4	6	=G6+H6		
7		5	7	=G7+H7		
8		6	8	=G8+H8		

Spalte I enthält die Formel, mit der Spalte G zu Spalte H addiert wird. Das kann man auch über eine Matrixformel erreichen, indem man nur eine Formel in eine einzige Zelle schreibt. Um die Formel der Spalte I als Matrixformel einzugeben, setzen Sie zuerst den Cursor in die Zelle J3. Schreiben Sie die Formel „=G3:G8+H3:H8“ und drücken Sie die Tastenkombination Strg-Umsch-Enter. Die Zellen J3 bis J8 enthalten nun dieselbe Formel: „{=G3:G8+H3:H8}“. Die Werte in der Spalte J sollten gleich denen in der Spalte I sein. Die Spalte I enthält sechs Formeln, die in keiner direkten Beziehung zueinander stehen, aber die Zellen in der Spalte J nutzen nur eine einzige Formel. Das Makro im Listing 442 baut eine Tabelle auf, die wie die Tabelle 197 aussieht, und trägt dann in die Spalte J eine Matrixformel ein, die dieselben Werte berechnet wie die Zellen der Spalte I.

Listing 442. *Demonstration einer Matrixformel.*

```
Sub ArrayFormula
  Dim oRange         'Der primäre Bereich
  Dim oSheet         'Das erste Tabellenblatt
  Dim oCell          'Eine temporäre Zelle
  Dim i As Integer   'Allgemeine Indexvariable
  Dim oDoc           'Referenz auf das neu erzeugte Calc-Dokument

  oDoc = StarDesktop.loadComponentFromURL("private:factory/scalc", _
         "_default", 0, Array())
  oSheet = oDoc.Sheets(0)

  REM Eingabe in die beiden oberen Zellen im Bereich G3:H8.
  oCell = oSheet.getCellByPosition(6, 2)      'Zelle G3
  oCell.setValue(1)

  oCell = oSheet.getCellByPosition(7, 2)      ' Zelle H3
  oCell.setValue(3)

  REM Ausfüllen der Werte nach unten.
  oRange = oSheet.getCellRangeByName("G3:H8")
  oRange.fillAuto(com.sun.star.sheet.FillDirection.TO_BOTTOM, 1)

  REM Demonstration, wie jede Zelle einzeln belegt wird.
  For i = 3 To 8
    oCell = oSheet.getCellByPosition(8, i - 1)      ' Zellen I3 - I8
    oCell.setFormula("=G" & i & "+H" & i)
  Next

  REM In diesem Fall ist es viel leichter, eine einzige Matrixformel einzutragen.
  oRange = oSheet.getCellRangeByName("J3:J8")
  oRange.setArrayFormula("=G3:G8+H3:H8")

  REM Fügt ein paar Überschriften hinzu.
  oRange = oSheet.getCellRangeByName("G2:J2")
  oRange.setDataArray(Array(Array("G", "H", "Formel", "Matrixformel")))
End Sub
```

15.4.11. Mehrfachoperationen in einem Zellbereich

OOo erlaubt den Einsatz von Formelreihen, in denen der Wert einer Variablen aus einer Reihe von Werten ersetzt wird. In der Regel sieht das so aus, dass direkt neben einer einzelnen Spalte (oder Zeile) mit Zahlen weitere Spalten (oder Zeilen) mit Formeln angeordnet sind, die diese Zahlenwerte nutzen. Die enumerierten Werte in der Tabelle 198 geben an, ob Spalten oder Zeilen verwendet werden.

Tabelle 198. *Die Enumeration com.sun.star.sheet.TableOperationMode.*

Wert	Beschreibung
COLUMN	Anwendung der Operation auf die Spaltenwerte.
ROW	Anwendung der Operation auf die Zeilenwerte.
BOTH	Anwendung der Operation auf die Spalten- und Zeilenwerte.

Die Methode setTableOperation() bietet die Möglichkeit, in mehreren Funktionen den Wert einer Variablen aus einem Satz von Daten zu nehmen und so eine Ergebnistabelle zu erzeugen. Die Methode benötigt die folgenden vier Argumente:

1. CellRangeAddress – Adresse des Zellbereichs mit den anzuwendenden Funktionen.
2. TableOperationMode – Angabe, ob die Daten in Zeilen oder Spalten vorliegen (s. Tabelle 198).
3. CellAddress – Zelladresse der Variablen, deren Wert aus einer Spalte genommen wird (nicht verwendet im Zeilenmodus).
4. CellAddress – Zelladresse der Variablen, deren Wert aus einer Zeile genommen wird (nicht verwendet im Spaltenmodus).

Das Makro im Listing 443 erzeugt eine Reihe von Zahlen von 0 bis 6,4. Die Funktionen Sin() und Cos() werden dann auf die Spalte angewendet.

Listing 443. *Mehrfachoperation an einer Spalte mit setTableOperation.*

```
Sub MultipleOpsColumns
  Dim oRange        'Der primäre Bereich.
  Dim oSheet        'Das erste Tabellenblatt.
  Dim oCell         'Eine temporäre Zelle.
  Dim oBlockAddress 'Adresse des auszufüllenden Blocks.
  Dim oCellAddress  'Adresse der Zeilen- oder Spaltenzelle.
  Dim oDoc          'Referenz auf das neu erzeugte Calc-Dokument.

  oDoc = StarDesktop.loadComponentFromURL("private:factory/scalc", _
         "_default", 0, Array())
  oSheet = oDoc.Sheets(0)

  REM Der oberste Wert: 0!
  oCell = oSheet.getCellByPosition(0, 9)     'Zelle A10
  oCell.setValue(0)

  REM Füllt die Zellen mit Werten von 0 bis (willkürlich) 6,4.
  oRange = oSheet.getCellRangeByName("A10:A74")
  oRange.fillSeries(com.sun.star.sheet.FillDirection.TO_BOTTOM,_
    com.sun.star.sheet.FillMode.LINEAR,_
    com.sun.star.sheet.FillDateMode.FILL_DATE_DAY,_
    0.1, 6.4)

  REM Schreibt die Spaltentitel Sin() und Cos().
```

```
  oCell = oSheet.getCellByPosition(1, 8)      ' Zelle B9
  oCell.setString("Sin()")
  oCell = oSheet.getCellByPosition(2, 8)      ' Zelle C9
  oCell.setString("Cos()")

  REM Schreibt die Formeln Sin() und Cos() in die Zellen.
  oCell = oSheet.getCellByPosition(1, 9)      ' Zelle B10
  oCell.setFormula("=Sin(A10)")
  oCell = oSheet.getCellByPosition(2, 9)      ' Zelle C10
  oCell.setFormula("=Cos(A10)")

  REM Der gesamte Block, in dem die Operation ausgeführt wird.
  oRange = oSheet.getCellRangeByName("A11:C74")

  REM Die Adresse des Zellbereichs mit den zu kopierenden Formeln.
  oBlockAddress = oSheet.getCellRangeByName("B10:C10").getRangeAddress()

  REM Die Zelladresse der Variablen, die durch die Spaltendaten ersetzt wird.
  oCellAddress = oSheet.getCellByPosition(0, 9).getCellAddress()

  REM Ich brauche zwar nur den Spaltenwert, weil der Zeilenwert nicht benutzt wird,
  REM aber es müssen beide Argumente vorhanden sein.
  oRange.setTableOperation(oBlockAddress,_
    com.sun.star.sheet.TableOperationMode.COLUMN,_
    oCellAddress, oCellAddress)
End Sub
```

Wenn der Mehrfachoperationsmodus auf BOTH statt einfach nur ROW oder COLUMN gesetzt ist, dann werden zwei Variablen einer einzigen Funktion ersetzt. Das Makro im Listing 444 erzeugt eine Multiplikationstabelle, die ich in der dritten Klasse auswendig lernen musste.

Listing 444. *Eine Multiplikationstabelle, erzeugt durch eine Mehrfachoperation.*

```
Sub CreateMultiplicationTableWrapper()
  CreateMultiplicationTable()
End Sub

Function CreateMultiplicationTable()
  Dim oRowCell      'Die Zeilenzelle
  Dim oColCell      'Die Spaltenzelle
  Dim oRange        'Der primäre Bereich
  Dim oSheet        'Das erste Tabellenblatt
  Dim oCell         'Eine temporäre Zelle
  Dim oBlockAddress 'Adresse des auszufüllenden Blocks
  Dim oCellAddress  'Zeilen- oder Spaltenzelle
  Dim oDoc          'Referenz auf das neu erzeugte Calc-Dokument

  oDoc = StarDesktop.loadComponentFromURL("private:factory/scalc", _
         "_default", 0, Array())
  oSheet = oDoc.Sheets(0)

  REM Die Zeile mit den festen Werten.
  oRowCell = oSheet.getCellByPosition(1, 9)    'Zelle B10
  oRowCell.setValue(1)
  oRange = oSheet.getCellRangeByName("B10:K10")
  oRange.fillAuto(com.sun.star.sheet.FillDirection.TO_RIGHT, 1)
  oRange.CharWeight = com.sun.star.awt.FontWeight.BOLD
  oRange.CharHeight = 14

  REM Die Spalte mit den festen Werten.
```

```
  oColCell = oSheet.getCellByPosition(0, 10)    ' Zelle A11
  oColCell.setValue(1)
  oRange = oSheet.getCellRangeByName("A11:A20")
  oRange.fillAuto(com.sun.star.sheet.FillDirection.TO_BOTTOM, 1)
  oRange.CharWeight = com.sun.star.awt.FontWeight.BOLD
  oRange.CharHeight = 14

  REM Die zu benutzende Formel. Sie verwendet die ersten Werte!
  oCell = oSheet.getCellByPosition(0, 9)     ' Zelle A10
  oCell.setFormula("=A11*B10")

  REM Der gesamte Zellbereich
  oRange = oSheet.getCellRangeByName("A10:K20")

  REM Füllt die Multiplikationstabelle für die Werte 1x1 bis 10x10 aus.
  oRange.setTableOperation(oRange.getRangeAddress(),_
    com.sun.star.sheet.TableOperationMode.BOTH,_
    oColCell.getCellAddress(),_
    oRowCell.getCellAddress())
  CreateMultiplicationTable() = oDoc
End Function
```

15.4.12. Einheitlich formatierte Zellen

Zellbereiche bieten zwei Methoden, Zellgruppen zu erreichen, die einheitlich formatiert sind. Die Methode getCellFormatRanges() gibt ein Objekt zurück, das sowohl indexierten als auch enumerierten Zugriff ermöglicht. Die einheitlich formatierten Zellen sind in mehrfache rechteckige Bereiche aufgesplittet. Die enumerierten Bereiche werden als SheetCellRange-Objekte bereitgestellt.

Die Objektmethode getUniqueCellFormatRanges() ist der Methode getCellFormatRanges() sehr ähnlich, außer dass die zurückgegebenen Objekte vom Typ SheetCellRanges sind. Der Hauptunterschied liegt darin, dass alle gleich formatierten Objekte in einem Container zusammengefasst sind. Das Makro im Listing 445 gibt die einheitlich formatierten Bereiche mit Hilfe der beiden verschiedenen Methoden aus.

Listing 445. *Ausgabe einheitlich formatierter Zellgruppen mit zwei verschiedenen Methoden.*

```
Sub DisplaySimilarRanges
  Dim oSheetCellRange  'Ein einzelner Zellbereich
  Dim oSheetCellRanges 'Zellbereichscontainer
  Dim oAddr            'Die Adresse eines Zellbereichs
  Dim s$               'Stringvariable
  Dim x                'Objekt der zurückgegebenen Bereichsobjekte
  Dim i%               'Indexvariable
  Dim oSheet           'Das erste Tabellenblatt
  Dim oDoc             'Referenz auf das neu erzeugte Calc-Dokument

  oDoc = CreateMultiplicationTable()
  oSheet = oDoc.Sheets(0)

  REM Der primäre Zellbereich umfasst das gesamte Tabellenblatt!
  x = oSheet.getCellFormatRanges()
  s = "**** getCellFormatRanges()" & Chr$(10)
  For i = 0 To x.getcount() - 1
    oSheetCellRange = x.getByIndex(i)
    oAddr = oSheetCellRange.getRangeAddress()
    s = s & PrettyRangeAddressName(oAddr) & Chr$(10)
```

```
    Next

    REM SheetCellRanges
    x = oSheet.getUniqueCellFormatRanges()
    s = s & Chr$(10) & "**** getUniqueCellFormatRanges()" & Chr$(10)
    For i = 0 To x.getcount() - 1
      oSheetCellRanges = x.getByIndex(i)
      s = s & i & " = "& oSheetCellRanges.getRangeAddressesAsString() & Chr$(10)
    Next
    MsgBox s, 0, "Einheitlich formatierte Bereiche"
End Sub
```

Bild 111 demonstriert sehr deutlich den Unterschied zwischen den beiden Methoden. Die Methode getCellFormatRanges() listet kontinuierlich die rechteckigen Bereiche auf, die einheitlich formatiert sind: bei jedem Formatwechsel beginnt ein neuer Bereich. Die Methode getUniqueCellFormatRanges() fasst die gefundenen Bereiche in Gruppen gleichen Formats zusammen. Somit zeigt sich, dass von den sieben Bereichen fünf (Index 0) beziehungsweise zwei (Index 1) die gleichen Formate haben.

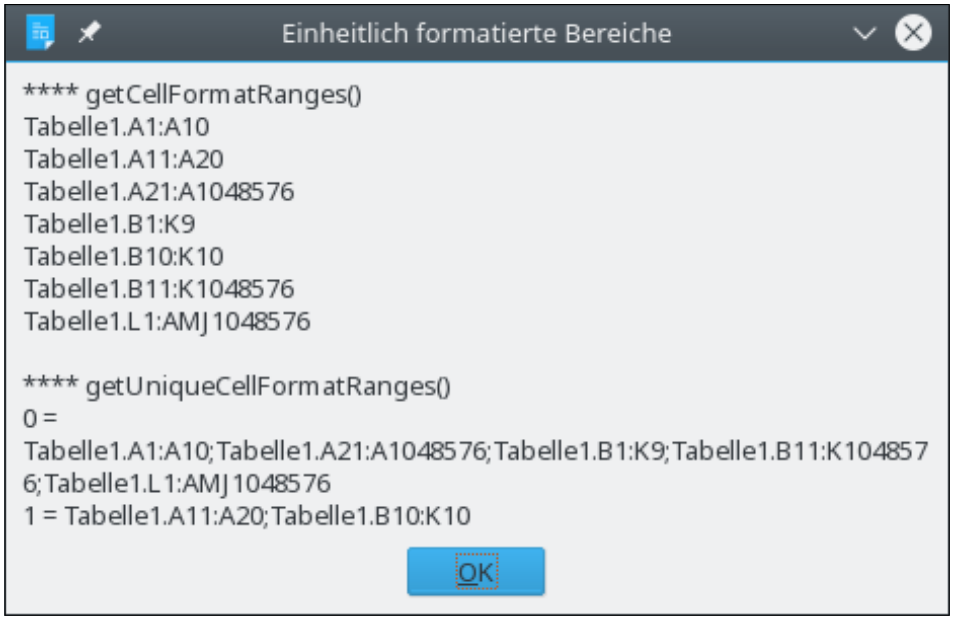


Bild 111. *Die beiden Methoden getCellFormatRanges() und getUniqueCellFormatRanges() gruppieren die Daten auf unterschiedliche Weise.*

15.4.13. Sortieren

Wenn Sie eine Sortierung anfordern, legen Sie mit einem Array von TableSortField-Structs fest, welche Spalten oder Zeilen in welcher Sortierfolge und nach welchem Typ sortiert werden soll (s. Tabelle 200). Im allgemeinen kennt OOo den Datentyp einer Zelle. Wegen dieser Kenntnis braucht man OOo nicht notwendigerweise über den zu sortierenden Datentyp zu informieren, dennoch ist es mit einem Wert der Enumeration TableSortFieldType möglich (s. Tabelle 199).

Tabelle 199. *Die Enumeration com.sun.star.table.TableSortFieldType.*

Wert	Beschreibung
AUTOMATIC	Automatische Bestimmung des Datentyps.
NUMERIC	Sortierung der Daten als Zahlen.
ALPHANUMERIC	Sortierung der Daten als Text.

Tabelle 200. *Eigenschaften des Structs com.sun.star.table.TableSortField.*

Eigenschaft	Beschreibung
Field	Nullbasierter Index der zu sortierenden Zeile oder Spalte. Der Index ist relativ und bezieht sich auf den Start des Sortierbereichs.
IsAscending	Falls True, werden die Daten aufsteigend sortiert.
IsCaseSensitive	Falls True, wird bei der Sortierung Groß- und Kleinschreibung berücksichtigt.
FieldType	Der Datentyp als TableSortFieldType (s. Tabelle 199).
CollatorLocale	Das bei der Textsortierung zu nutzende Locale-Objekt (Gebietsschema).
CollatorAlgorithm	Der Sortieralgorithmus, den der Collator bei der Textsortierung verwenden soll. Über das Interface com.sun.star.i18n.XCollator können Sie herausfinden, welche Algorithmen für Ihr Gebietsschema zur Verfügung stehen. Ich bin mit dem Standardwert immer gut gefahren.

Bei der Anforderung einer Sortierung wird der Sortierroutine ein Sortierdeskriptor als Property-Array übergeben. Die Properties bestimmen, was und auf welche Weise sortiert wird. Eine der unterstützten Properties ist SortFields (s. Tabelle 201), das als Wert ein Array des Structs TableSortField (Tabelle 200) erhält, in dem die Sortierregeln für die Zeilen und Spalten stehen.

Tabelle 201. *Der alte Sortierdeskriptor vom Service com.sun.star.table.TableSortDescriptor.*

Property	Beschreibung
IsCaseSensitive	Falls True, wird bei der Sortierung Groß- und Kleinschreibung berücksichtigt.
SortAscending	Falls True, werden die Daten aufsteigend sortiert. Diese Eigenschaft wird normalerweise nicht genutzt, denn TableSortField legt IsAscending für die einzelnen Felder fest.
SortColumns	Falls True, werden Spalten sortiert, andernfalls Zeilen.
CollatorLocale	Das bei der Textsortierung zu nutzende Locale-Objekt – das Gebietsschema (wird normalerweise im TableSortField festgelegt).
CollatorAlgorithm	Der zu verwendende Sortieralgorithmus (wird normalerweise im TableSortField festgelegt).
SortFields	TableSortField-Array (s. Tabelle 200), das steuert, was sortiert wird.
MaxFieldCount	Long Integer. Die Höchstanzahl an Sortierfeldern, die der Deskriptor aufnehmen kann. Dieser Wert kann nicht gesetzt, sondern nur gelesen werden.
ContainsHeader	Falls True, wird die erste Zeile oder Spalte als Überschrift betrachtet und nicht sortiert.
Orientation	Diese Eigenschaft ist veraltet und sollte nicht mehr benutzt werden!

Die ersten fünf Properties sind vom Service com.sun.star.util.SortDescriptor geerbt. Die Tabellen 200 und 201 lassen eine Menge Redundanz erkennen. Sie können zum Beispiel die Beachtung der Groß- und Kleinschreibung entweder global (Tabelle 201) oder für jedes einzelne Feld (Tabelle 200) festlegen.

Weil die redundanten Felder der Tabelle 201 nicht benötigt und daher normalerweise auch nicht verwendet werden, wurde ein neuer Satz von Sortierdeskriptoren eingeführt (s. Tabelle 202). Sie können zwar die Properties aus der einen wie auch aus der anderen Tabelle nutzen, aber Sie dürfen sie nicht vermengen. Das OOo-Entwicklerteam hat allerdings den Gebrauch der Property Orientation (Tabelle 201) ausdrücklich als veraltet gekennzeichnet. Meine Empfehlung ist, den neuen Property-Satz aus der Tabelle 202 zu wählen.

Tabelle 202. *Der neue Sortierdeskriptor vom Service com.sun.star.sheet.SheetSortDescriptor2.*

Property	Beschreibung
SortFields	TableSortField-Array (s. Tabelle 200), das steuert, was sortiert wird.

Property	Beschreibung
MaxSortFieldsCount	Long Integer. Die Höchstanzahl an Sortierfeldern, die der Deskriptor aufnehmen kann. Dieser Wert kann nicht gesetzt, sondern nur gelesen werden.
IsSortColumns	Falls True, werden Spalten sortiert, andernfalls Zeilen.
BindFormatsToContent	Falls True, werden Zellformate bei der Sortierung mit den Inhalten mitgeführt. Wirkt nur, wenn unterschiedliche Zellen im Sortierbereich unterschiedlich formatiert sind.
IsUserListEnabled	Falls True, wird eine benutzerdefinierte Sortierliste aus den GlobalSheetSettings verwendet (s. auch Extras | Optionen | LibreOffice Calc | Sortierlisten).
UserListIndex	Long Integer. Die konkret zu verwendende benutzerdefinierte Sortierliste.
CopyOutputData	Falls True, werden die sortierten Daten an eine andere Stelle im Dokument kopiert.
OutputPosition	Zieladresse der Kopie der sortierten Daten (falls CopyOutputData True ist).
ContainsHeader	Falls True, wird die erste Zeile oder Spalte als Überschrift betrachtet und nicht sortiert.

Die ersten drei Properties sind vom Service com.sun.star.table.TableSortDescriptor2 geerbt.

Der erste Schritt zur Sortierung eines Bereichs besteht darin, über ein TableSortField-Array die Felder zu definieren, nach denen sortiert werden soll. Dann kommen die Properties aus der Tabelle 202 an die Reihe, die bei der Sortierung zu beachten sind. Und schließlich wird die Methode sort() des zu sortierenden Bereichs aufgerufen. Das Makro im Listing 446 führt eine absteigende Sortierung über die erste Spalte durch.

Listing 446. *Eine Spalte in einem Calc-Tabellenblatt sortieren.*

```
Sub SortColZero
  Dim oSheet         'Das erste Tabellenblatt
  Dim oRange         'Der zu sortierende Bereich
  Dim oSortFields(0) As New com.sun.star.table.TableSortField
  Dim oSortDesc(0) As New com.sun.star.beans.PropertyValue

  oSheet = ThisComponent.Sheets(0)
  REM Der zu sortierende Bereich.
  oRange = oSheet.getCellRangeByName("B28:D33")

  REM Sortiert wird absteigend über das erste Feld im Bereich.
  oSortFields(0).Field = 0
  oSortFields(0).IsAscending = False

  REM Die zu nutzenden Sortierfelder.
  oSortDesc(0).Name = "SortFields"
  oSortDesc(0).Value = oSortFields()

  REM Nun wird der Bereich sortiert!
  oRange.sort(oSortDesc())
End Sub
```

Über zwei Spalten zu sortieren, bedeutet einfach, ein zweites Sortierfeld hinzuzufügen. Es wirkt bei identischen Werten der ersten Sortierspalte. Listing 447 sortiert über die zweite und die dritte Spalte.

Listing 447. *Zwei Spalten in einem Calc-Tabellenblatt sortieren.*

```
Sub SortColOne
  Dim oSheet         'Das erste Tabellenblatt
  Dim oRange         'Der zu sortierende Bereich
  Dim oSortFields(1) As New com.sun.star.table.TableSortField
  Dim oSortDesc(0) As New com.sun.star.beans.PropertyValue

  oSheet = ThisComponent.Sheets(0)
```

```
  REM Der zu sortierende Bereich.
  oRange = oSheet.getCellRangeByName("B28:D33")

  REM Sortiert wird aufsteigend über das zweite Feld im Bereich.
  oSortFields(0).Field = 1
  oSortFields(0).IsAscending = True
  oSortFields(0).FieldType = com.sun.star.util.SortFieldType.NUMERIC

  REM Sortiert wird auch über das dritte Feld im Bereich.
  oSortFields(1).Field = 2
  oSortFields(1).IsAscending = True
  oSortFields(1).FieldType = com.sun.star.util.SortFieldType.ALPHANUMERIC

  REM Die zu nutzenden Sortierfelder.
  oSortDesc(0).Name = "SortFields"
  oSortDesc(0).Value = oSortFields()

  REM Nun wird der Bereich sortiert!
  oRange.sort(oSortDesc())
End Sub
```

Die Methode createSortDescriptor() gibt ein Array von Property-Werten zurück, die definieren, wie eine Sortierung durchgeführt werden soll. Eine Inspizierung dieses erzeugten Sortierdeskriptors zeigt, dass man zum Sortieren maximal drei Felder heranziehen kann (s. MaxSortFieldsCount in der Tabelle 202). Das Makro im Listing 448 erzeugt einen Sortierdeskriptor und gibt die enthaltenen Properties aus (s. Bild 112). Welche Properties es sind, hängt von der Implementierung der verwendeten AOO/LO-Programmversion ab.

Listing 448. *Ausgabe der Sortierdeskriptor-Properties in Calc.*

```
Sub DisplaySortDescriptor
  On Error Resume Next
  Dim oSheet          'Das erste Tabellenblatt
  Dim oRange          'Der Sortierbereich
  Dim oSortDescript   'Der Sortierdeskriptor
  Dim i%
  Dim s$
  Dim oDoc            'Referenz auf das neu erzeugte Calc-Dokument.

  oDoc = StarDesktop.loadComponentFromURL("private:factory/scalc", _
         "_default", 0, Array())
  oSheet = oDoc.Sheets(0)
  oRange = oSheet.getCellRangeByName("B28:D33")
  'Der Sortierdeskriptor wird vom Bereich erzeugt.
  oSortDescript = oRange.createSortDescriptor()
  For i = LBound(oSortDescript) To UBound(oSortDescript)
    s = s & oSortDescript(i).Name & " = "
    s = s & oSortDescript(i).Value
    s = s & Chr$(10)
  Next
  MsgBox s, 0, "Sortierdeskriptor"
End Sub
```

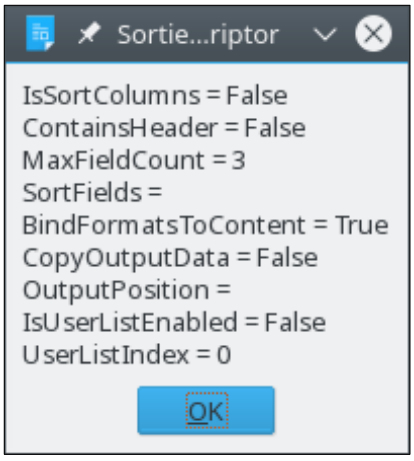


Bild 112. *Attribute des Sortierdeskriptors.*

15.5. Tabellenblätter

Der Großteil der Funktionalitäten eines Calc-Dokuments gehört zu den einzelnen Tabellenblättern und nicht zum gesamten Dokument. Die vom Tabellendokument eingebundenen Interfaces (s. Tabelle 170) betreffen primär das Dokument als Ganzes, nicht die einzelnen Tabellenblätter.

Die Tabellenblätter in einem Calc-Dokument binden den Service SheetCellRange ein, der eine beträchtliche Fülle an Funktionalitäten bietet, und das nicht nur für jeden Zellbereich eines Tabellenblatts, sondern sogar über Tabellenblätter hinaus: jeder Bereich hat Zugang zu den vom Service SheetCellRange eingesetzten Methoden. Die Tabellenblätter selbst unterstützen noch weitere Interfaces, die nicht direkt mit Zellbereichen zu tun haben (s. Tabelle 203).

Tabelle 203. *Vom Service com.sun.star.sheet.Spreadsheet eingebundene Interfaces.*

Interfaces	Beschreibung
com.sun.star.sheet.XSpreadsheet	Methoden zur Erzeugung eines Zellcursors.
com.sun.star.container.XNamed	Zugriff auf den Namen des Tabellenblatts.
com.sun.star.util.XProtectable	Methoden zum Setzen und Entfernen des Tabellenblattschutzes.
com.sun.star.sheet.XDataPilotTablesSupplier	Zugriff auf die Datenpilot-Tabellen über die Methode getDataPilotTables(). Im englischsprachigen LibreOffice ist der Data Pilot in Pivot Tables umbenannt worden.
com.sun.star.sheet.XScenariosSupplier	Zugriff auf die Szenarien über die Methode getScenarios().
com.sun.star.sheet.XSheetAnnotationsSupplier	Zugriff auf die Kommentare über die Methode getAnnotations().
com.sun.star.drawing.XDrawPageSupplier	Zugriff auf die Folie des Tabellenblatts über die Methode getDrawPage().
com.sun.star.table.XTableChartsSupplier	Zugriff auf die Diagrammobjekte des Dokuments mit der Methode getCharts().
com.sun.star.sheet.XCellRangeMovement	Verschieben von Zellbereichen innerhalb des Tabellenblatts oder in andere Tabellenblätter im selben Dokument.
com.sun.star.sheet.XPrintAreas	Zugriff auf die Druckbereichseinstellungen dieses Tabellenblatts.
com.sun.star.sheet.XSheetPageBreak	Zugriff auf die Seitenumbrüche (und Modifizierung) in diesem Tabellenblatt.
com.sun.star.sheet.XScenario	Methoden für ein Szenarienblatt.
com.sun.star.sheet.XSheetOutline	Zugriff auf die Einstellungen der Zeilen- und Spaltengliederung für das Tabellenblatt.
com.sun.star.sheet.XSheetAuditing	Suche nach verknüpften Zellen (Detektiv).
com.sun.star.sheet.XSheetLinkable	Methoden zur Verknüpfung mit in anderen Dokumenten enthaltenen Tabellenblättern.

15.5.1. Verknüpfung mit einem externen Tabellendokument

Ein einzelnes Tabellenblatt kann mit einem Tabellenblatt aus einem anderen Tabellendokument verknüpft werden. Das bedeutet, dass die „Verknüpfungstabelle“ als Container für die „verknüpfte Tabelle“ dient. Obwohl Sie nach der Verknüpfung die verknüpfte Tabelle im Container bearbeiten können, werden die Änderungen aber nicht in das Originaldokument übertragen. Wenn die verknüpfte Tabelle im Originaldokument manipuliert wurde, ist die Änderung in der Verknüpfungstabelle erst dann zu sehen, wenn die Verknüpfung aktualisiert wurde. Die Dokumentverknüpfung wird durch einen der enumerierten Werte der Tabelle 204 gesteuert.

Tabelle 204. *Die Enumeration com.sun.star.sheet.SheetLinkMode.*

Wert	Beschreibung
NONE	Das Tabellenblatt ist nicht verknüpft.
NORMAL	Kopiert den gesamten Inhalt mit Werten und Formeln.
VALUE	Kopiert den Wert des Inhalts. Das Formelresultat wird statt der Formel selbst kopiert.

Mit der Methode link() wird die Verknüpfung mit einem Tabellenblatt in einem anderen Dokument hergestellt. Die Tabelle 205 listet die für Verknüpfungen vorgesehenen Methoden eines Tabellenblatts auf.

Tabelle 205. *Methoden im Interface com.sun.star.sheet.XSheetLinkable.*

Methode	Beschreibung
getLinkMode()	Liest den Verknüpfungsmodus (s. Tabelle 204).
setLinkMode(SheetLinkMode)	Setzt den Verknüpfungsmodus (s. Tabelle 204).
getLinkUrl()	Liest den URL der Verknüpfung.
setLinkUrl(url)	Setzt den URL der Verknüpfung.
getLinkSheetName()	Liest den Namen der verknüpften Tabelle.
setLinkSheetName(name)	Setzt den Namen der verknüpften Tabelle.
link(url, sheetName, filterName, filterOptions, SheetLinkMode)	Verknüpft die Tabelle mit einer anderen Tabelle in einem anderen Dokument.

Das Makro im Listing 449 erzeugt eine Tabelle namens „Verknüpft“ und stellt dann eine Verknüpfung zu einem externen Dokument her. Wenn das Tabellenblatt „Verknüpft“ schon existiert, wird die Verknüpfung vom Tabellendokument geholt und aktualisiert. Damit werden die Daten aus dem verknüpften in das aktuelle Tabellenblatt neu eingelesen.

Listing 449. *Verknüpfung mit einem externen Tabellenblatt.*

```
Sub LinkASheet
  Dim oSheets          'Das Sheets-Objekt enthält alle Einzeltabellenblätter.
  Dim oSheet           'Ein einzelnes Tabellenblatt
  Dim oSheetEnum       'Für den enumerierten Zugriff
  Dim s As String      'Stringvariable für temporäre Daten
  Dim i As Integer     'Indexvariable
  Dim sURL As String   'URL des zu importierenden Objekts
  Dim oLink            'Das Verknüpfungsobjekt

  sURL = "file://C:/Meine%20Dokumente/Kap15/test.ods"
  oSheets = ThisComponent.Sheets

  If oSheets.hasByName("Verknüpft") Then
    REM Die Verknüpfungen sind im Dokument-Objekt
```

```
      REM unter dem Namen des verwendeten URL abrufbar.
      oLink = ThisComponent.SheetLinks.getByName(sURL)
      oLink.refresh()
      MsgBox "Die Tabelle Verknüpft wurde aktualisiert."
      Exit Sub
    End If

    REM Ein neues Tabellenblatt wird am Ende eingefügt.
    oSheets.insertNewByName ("Verknüpft", oSheets.getCount())
    oSheet = oSheets.getByName("Verknüpft")

    oSheet.link(sURL, "Tabelle1", "", "", com.sun.star.sheet.SheetLinkMode.NORMAL)
End Sub
```

Die erste Anwendung, die ich für verknüpfte Tabellen gesehen habe, führte eine Reihe verschiedener Kapitalanlagen zusammen, deren Verlauf in jeweils eigenen Calc-Dokumenten verfolgt wurde. Jedes dieser Dokumente hatte eine zusammenfassende Tabelle. In einem gesonderten Überblicksdokument gab es Verknüpfungen zu all den zusammenfassenden Tabellen der anderen Anlagendokumente.

Obwohl verknüpfte Tabellen ganz nett sind, so sind sie doch manchmal zu viel des Guten. Wenn Sie nicht unbedingt eine Referenz auf ein komplettes Tabellenblatt aus einem anderen Dokument benötigen, können Sie Verknüpfungen zu einzelnen Zellen direkt in den Formeln setzen: s. Listing 450.

Listing 450. *Verknüpfung in der Zelle A1 mit der Zelle K89 in einem anderen Dokument.*

```
oCell = ThisComponent.Sheets(0).getCellByPosition(0,0) ' A1
oCell.setFormula("=" & "'file:///home/USER/CalcFile2.odt'#$Tabelle2.K89")
```

15.5.2. Abhängigkeiten suchen mit Detektiv-Funktionen

Die in der Tabelle 190 gezeigten Methoden queryDependents() und queryPrecedents() geben eine Liste von Zellen zurück, die einen Zellbereich referenzieren (Nachfolger) oder von ihm referenziert werden (Vorgänger). Diese Abfragemethoden sind nützlich für Makros, die auf jede verknüpfte Zelle zugreifen sollen. Die vom Interface XSheetAuditing bereitgestellte Detektiv-Funktionalität bietet Methoden, die Zellenabhängigkeit zu visualisieren (s. Tabelle 206).

Tabelle 206. *Methoden im Interface com.sun.star.sheet.XSheetAuditing.*

Methode	Beschreibung
hideDependents(CellAddress)	Entfernt Pfeile für eine Nachfolgerebene. Gibt True zurück, wenn die Aktion durchgeführt werden konnte.
hidePrecedents(CellAddress)	Entfernt Pfeile für eine Vorgängerebene. Gibt True zurück, wenn die Aktion durchgeführt werden konnte.
showDependents(CellAddress)	Zeichnet Pfeile von der Zelladresse (s. Tabelle 172) zu seinen Nachfolgern. Gibt True zurück, wenn die Aktion durchgeführt werden konnte.
showPrecedents(CellAddress)	Zeichnet Pfeile zur Zelladresse (s. Tabelle 172) von seinen Vorgängern. Gibt True zurück, wenn die Aktion durchgeführt werden konnte.
showErrors(CellAddress)	Zeichnet Pfeile von der Zelladresse (s. Tabelle 172), die eine Fehlermeldung enthält, zu den Zellen, die den Fehler ausgelöst haben. Gibt True zurück, wenn die Aktion durchgeführt werden konnte.
showInvalid()	Zeigt alle Zellen mit ungültigen Werten an. Gibt True zurück, wenn die Aktion durchgeführt werden konnte.
clearArrows()	Entfernt alle durch den Detektiv eingefügten Zellen aus dem Tabellenblatt.

Bei jedem Aufruf der Methode showPrecedents() wird eine weitere Vorgängerebene mit Pfeilen markiert. Nach dem ersten Aufruf werden Pfeile von allen Zellen gezogen, die von der angegebenen Zelle direkt referenziert werden. Das Makro QueryRange im Listing 434 listet die Abhängigkeiten auf

(s. Bild 107). Listing 451 jedoch visualisiert die Vorgänger im Tabellenblatt. Bild 113 zeigt eine Ebene.

Die Zelle B7 enthält die Formel „=SUMME(B1:B6)“. Wie im Bild 113 zu sehen ist, referenziert die Zelle B7 alle summierten Zellen. Nach einem zweiten Aufruf von showPrecedents() sehen Sie die von B1:B6 referenzierten Zellen. Die Methode showPrecedents() gibt True zurück, solange weitere Vorgänger mit Pfeilen markiert werden. Bild 114 zeigt die zweite Vorgängerebene.

Listing 451. *Visualisierung der Vorgänger.*

```
Function SimpleCalcDocAddition()
  Dim oDoc          'Referenz auf das neu erzeugte Calc-Dokument
  Dim oSheet        'Erstes Tabellenblatt
  Dim oRange        'Zellbereich

  oDoc = StarDesktop.loadComponentFromURL("private:factory/scalc", _
         "_default", 0, Array())
  oSheet = oDoc.Sheets(0)
  oSheet.getCellByPosition(0, 0).setValue(1)
  oRange = oSheet.getCellRangeByName("A1:A6")
  oRange.fillAuto(com.sun.star.sheet.FillDirection.TO_BOTTOM, 1)
  oRange = oSheet.getCellRangeByName("B1:B6")
  oRange.setArrayFormula("=A1:A6+1")     'Matrixformel
  oSheet.getCellByPosition(1, 6).setFormula("=Sum(B1:B6)")
  SimpleCalcDocAddition = oDoc
End Function

Sub ShowCellPrecedence()
  Dim oDoc
  Dim oSheet
  Dim oAddr

  oDoc = SimpleCalcDocAddition()
  oSheet = oDoc.Sheets(0)
  oAddr = oSheet.getCellByPosition(1, 6).CellAddress
  oSheet.showPrecedents(oAddr)
  Print "Eine Vorgängerebene für die Zelle B7"
  oSheet.showPrecedents(oAddr)
  Print "Zwei Vorgängerebenen für die Zelle B7"
End Sub
```

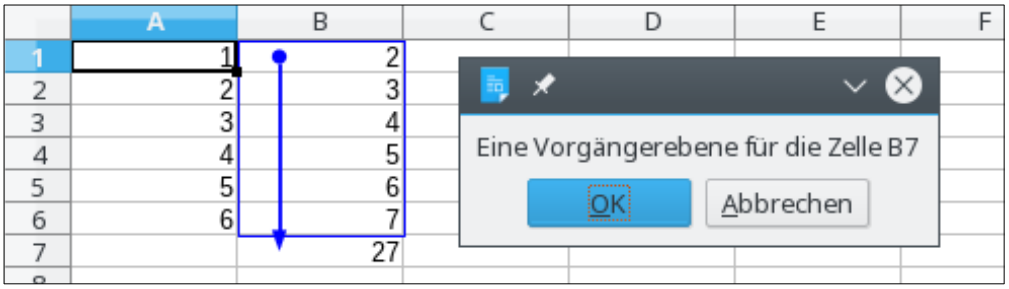


Bild 113. *Eine Formel-Vorgängerebene*

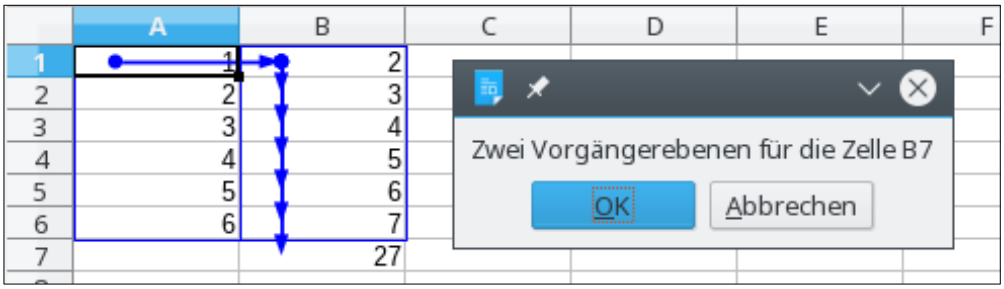


Bild 114. *Zwei Formel-Vorgängerebenen mit einer Matrixformel in B1:B6.*

Die Pfeile im Bild 114 zeigen zwei Vorgängerebenen mit einer Matrixformel. Wird statt der Matrixformel in jeder Zelle eine einfache Formel verwendet, ändert sich die Darstellung erheblich. Überlegen Sie einmal zur Übung, warum das so ist.

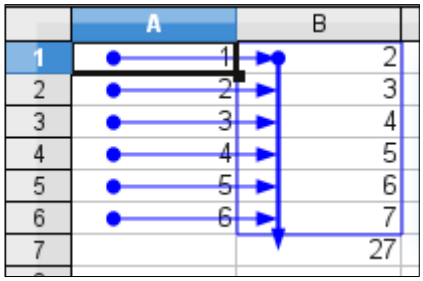


Bild 115. *Zwei Formel-Vorgängerebenen mit Formeln in B1:B6.*

15.5.3. Gliederungen

In einem Calc-Dokument können Zeilen und Spalten zu Gliederungen gruppiert werden, so dass man die Gruppen mit einfachem Mausklick ein- oder ausklappen kann. Wenn Sie eine Gliederung erzeugen, müssen Sie mit der Enumeration TableOrientation (s. Tabelle 207) festlegen, ob sie sich auf Zeilen oder Spalten bezieht. Die in der Tabelle 208 aufgeführten Methoden verhalten sich wie ihre Gegenstücke im OOo-GUI „Gruppierung und Gliederung“.

Tabelle 207. *Die Enumeration com.sun.star.table.TableOrientation.*

Wert	Beschreibung
ROWS	Es werden Zeilen verwendet.
COLUMNS	Es werden Spalten verwendet.

Tabelle 208. *Methoden im Interface com.sun.star.sheet.XSheetOutline.*

Methode	Beschreibung
group(CellRangeAddress, TableOrientation)	Fasst die Zellen im Zellbereich zu einer Gruppe zusammen.
ungroup(CellRangeAddress, TableOrientation)	Entfernt die untersten Ebenen aus der Gruppe.
autoOutline(CellRangeAddress)	Erzeugt Gliederungsgruppen auf der Basis von Formelverknüpfungen.
clearOutline()	Entfernt alle Gliederungsgruppen aus dem Tabellenblatt.
hideDetail(CellRangeAddress)	Klappt eine Gliederungsgruppe ein (verbirgt die untere Ebene).
showDetail(CellRangeAddress)	Klappt eine Gliederungsgruppe aus (zeigt die untere Ebene).
showLevel(n, CellRangeAddress)	Zeigt Gliederungsgruppen von der Ebene 1 bis n.

15.5.4. Zellen kopieren, verschieben und einfügen

In einem Writer-Dokument wird Textinhalt hauptsächlich über die Zwischenablage verschoben oder kopiert. Der Service Spreadsheet bietet jedoch Methoden, Zellen direkt zu verschieben und einzufü-

gen. Wenn neue Zellen einzufügen sind, legen Sie mit der Enumeration CellInsertMode fest, wie die vorhandenen Zellen dem Neuankömmling Platz machen (s. Tabelle 209).

Tabelle 209. *Die Enumeration com.sun.star.sheet.CellInsertMode.*

Wert	Beschreibung
NONE	Es werden keine Zellen verschoben.
DOWN	Die Zellen werden nach unten verschoben.
RIGHT	Die Zellen werden nach rechts verschoben.
ROWS	Die gesamte Zeile wird nach unten verschoben.
COLUMNS	Die gesamte Spalte wird nach rechts verschoben.

Mit der Methode insertCells(CellRangeAddress, CellInsertMode) schaffen Sie Platz in der Größe der Zellbereichsadresse. Wenn der Einfügemodus COLUMNS ist, wird die gesamte Spalte um die Bereichsbreite nach rechts verschoben, beginnend mit der Spalte ganz links im Bereich. Wenn der Einfügemodus RIGHT ist, wird nicht die gesamte Spalte nach rechts verschoben, sondern nur die Zeilen im Bereich. Die Einfügemodi ROWS und DOWN verhalten sich vergleichsweise wie die Modi COLUMNS und RIGHT. Der Modus NONE bewirkt, dass keine Zellen verschoben werden, mit anderen Worten, es passiert gar nichts. Listing 452 verschiebt Zellen nach unten.

Listing 452. *Verschiebt den Bereich L4:M5 nach unten.*

```
Dim oSheet          'Das vierte Tabellenblatt
Dim oRangeAddress   'Der zu verschiebende Bereich
oSheet = ThisComponent.Sheets(3)
oRangeAddress = oSheet.getCellRangeByName("L4:M5").getRangeAddress()
oSheet.insertCells(oRangeAddress, com.sun.star.sheet.CellInsertMode.DOWN)
```

Tipp Die Methoden insertCells() und removeRange() werden ohne Aktion und ohne Hinweis abgebrochen, wenn beim Einfügen eine Matrixformel auseinandergerissen würde.

Die Methode removeRange(CellRangeAddress, CellInsertMode) ist im Prinzip ein „Rückgängig“-Befehl für die Methode insertCells().

Mit der Methode copyRange(CellAddress, CellRangeAddress) wird ein Zellbereich an die von der Zelladresse angegebene Position kopiert. Die Zelle links oben in der Zellbereichsadresse wird beim Kopieren auf die Zelladresse positioniert. Im Endeffekt es dasselbe, als wenn Sie einen Zellbereich in die Zwischenablage kopieren, den Cursor auf die Zielzelle setzen und dann aus der Zwischenablage an diese Stelle kopieren (s. Listing 453).

Listing 453. *Kopiert den Bereich L4:M5 nach N8.*

```
Dim oSheet        'Das vierte Tabellenblatt
Dim oRangeAddress 'Der zu kopierende Bereich
Dim oCellAddress  'Zieladresse
oSheet = ThisComponent.Sheets(3)
oRangeAddress = oSheet.getCellRangeByName("L4:M5").getRangeAddress()
oCellAddress  = oSheet.getCellByPosition(13, 7).getCellAddress()  'N8
oSheet.copyRange(oCellAddress, oRangeAddress)
```

Mit der Methode moveRange(CellAddress, CellRangeAddress) wird ein Zellbereich verschoben. Die Methode verhält sich ähnlich wie die Methode copyRange(), mit dem Unterschied, dass die Zellen verschoben und nicht kopiert werden. Die Zellen im ursprünglichen Bereich bleiben leer.

15.5.5. Daten zwischen Dokumenten kopieren

Die Methode copyRange kann nur einen Zellbereich im selben Dokument kopieren. Um Daten zwischen verschiedenen Dokumenten zu kopieren, muss man andere Methoden anwenden.

Datenfunktionen

Mit getData() und setData() werden Zahlen kopiert. Mit getDataArray() und setDataArray() werden Zahlen und Strings kopiert. Die Datenfunktionen sind einfach und schnell, aber sie kopieren nur Daten, keine Formate.

Zwischenablage

Zum Kopieren von Daten in die Zwischenablage sind Dispatch-Befehle nützlich. Die Zwischenablage bietet beim Kopieren viele Optionen. Das Hauptproblem der Zwischenablage ist die ihr eigene Unzuverlässigkeit, denn andere Anwendungen könnten sie zur selben Zeit verwenden. Trotz dieser Unsicherheit bietet „Inhalte einfügen“ eine Reihe von Optionen, wenn Daten einzufügen sind – Optionen, die andere Methoden nicht bieten.

Listing 454. *Daten über die Zwischenablage zwischen Dokumenten kopieren.*

```
oDispatcher = CreateUnoService("com.sun.star.frame.DispatchHelper")
oFrame1 = oDoc1.CurrentController.Frame
'Die Zellen A1:B2 werden über den Controller ausgewählt.
oSheet = oDoc1.Sheets(0)
oRng = oSheet.getCellRangeByName("A1:B2")
oDoc1.CurrentController.select(oRng)
'Mit einem Dispatch-Befehl wird in die Zwischenablage kopiert.
oDispatcher.executeDispatch(oFrame1, ".uno:Copy", "", 0, Array())
'Auswahl der oberen linken Ecke zum Einfügen der Daten.
oRng = oDoc2.Sheets(0).getCellRangeByName("A1")
'Erst wird der Viewcursor dort platziert, dann wird aus der Zwischenablage eingefügt.
oDoc2.CurrentController.select(oRng)
oFrame2 = oDoc2.CurrentController.Frame
oDispatcher.executeDispatch(oFrame2, ".uno:Paste", "", 0, Array())
```

Zeichnen Sie ein Makro mit „Inhalte einfügen“ auf, um zu sehen, wie die Argumente gesetzt werden müssen. Ein Beispiel folgt:

Listing 455. *Argumente für „Inhalte einfügen“.*

```
Dim args1(5) As New com.sun.star.beans.PropertyValue
args1(0).Name = "Flags"
args1(0).Value = "SVDNT"
args1(1).Name = "FormulaCommand"
args1(1).Value = 0
args1(2).Name = "SkipEmptyCells"
args1(2).Value = False
args1(3).Name = "Transpose"
args1(3).Value = False
args1(4).Name = "AsLink"
args1(4).Value = False
args1(5).Name = "MoveMode"
args1(5).Value = 4
oDispatcher.executeDispatch(oFrame2, ".uno:InsertContents", "", 0, args1())
```

Übertragbarer Inhalt

Übertragbarer Inhalt, das Neueste in Sachen Inhalt kopieren, gibt eine Kopie der ausgewählten Daten zurück, als wenn die Kopie in die Zwischenablage gegangen wäre, doch die Zwischenablage wird nicht genutzt. Übertragbarer Inhalt ist auch in Writer-Dokumenten verfügbar. Anders als bei der

Zwischenablage sind keine Probleme zu erwarten, aber man hat leider auch nicht die Flexibilität von „Inhalte einfügen“. Der übertragbare Inhalt wird mit der Controller-Methode getTransferable() (Interface com.sun.star.datatransfer.XTransferableSupplier) aus der aktuellen Selektion geholt und mit insertTransferable() auf die neue Selektion kopiert.

Listing 456. *Daten über übertragbaren Inhalt zwischen Dokumenten kopieren.*

```
Dim o         'Die übertragbaren Daten
Dim oSheet    'Tabellenblatt
Dim oRange    'Zellbereich
Dim oDoc      'Dokument

oRange = oDoc1.Sheets(0).getCellRangeByName("B2:C3")
oDoc1.CurrentController.select(oRange)
o = oDoc1.CurrentController.getTransferable()

oRange = oDoc2.Sheets(0).getCellRangeByName("F1")
oDoc2.CurrentController.select(oRange)
oDoc2.CurrentController.insertTransferable(o)
```

15.5.6. Datenpilot und Pivot-Tabellen

OOo bietet eine Funktionalität namens Datenpilot, den LibreOffice in Pivot-Tabelle umbenannt hat. Dieser Abschnitt geht davon aus, dass Sie die Benutzung des Datenpiloten schon kennen, und stellt nur einige seiner Funktionen innerhalb von Makros vor.

Tipp Ich könnte einen umfangreichen Abschnitt über die zahllosen Einsatzbereiche des Datenpiloten (Pivot-Tabelle) schreiben. Um ein Gefühl für die Möglichkeiten zu bekommen, schauen Sie sich die Eingabetabelle in der Tabelle 210 an und vergleichen sie dann mit der Ergebnistabelle im Bild 116, wie sie vom Datenpiloten automatisch generiert wird.

Der Datenpilot ist ein mächtiger Mechanismus, der die Kombination, den Vergleich und die Analyse großer Datenmengen ermöglicht. Der Datenpilot manipuliert Teile der Daten aus der Quelltabelle und gibt die Ergebnisse an anderer Stelle aus. Leider ist eine Vielzahl von Einzelheiten bei der Erzeugung und Manipulation von Datenpilot-Tabellen zu beachten, doch das liegt an der enormen Flexibilität.

Diese Vielfalt der Einzelheiten zum Erstellen und Benutzen des Datenpiloten ist zwar logisch und geradlinig aufgebaut, aber durch ihre schiere Menge kann man sich leicht in den Feinheiten verlieren. Daher ist es hilfreich, ein einfaches Beispiel nachzuvollziehen, das die einzelnen Schritte verdeutlicht. Die spezifischen Typen und Enumerationen werde ich nach dem Beispiel vorstellen. Sie können sie dann bei Bedarf nachlesen.

Ein Beispiel für den Datenpiloten

Für dieses Beispiel habe ich eine Firma erfunden, die Bücher, Süßigkeiten und Schreibwaren verkauft. Die Firma hat Büros in drei Bundesländern und Verkaufspersonal in jedem Land. Ich habe ein Tabellenblatt erstellt, das die Verkaufszahlen für jedes Produkt pro Verkäufer und Jahr zeigt. Das Ziel dieses Beispiels ist die Erstellung einer Datenpilot-Tabelle mit den Verkaufszahlen für jedes Produkt, nach Typ und Land gelistet. Die Ausgangsdaten für dieses Beispiel finden Sie in der Tabelle 210.

Tabelle 210. *Die Datenwerte für die Datenpilot-Beispiele.*

Artikel	Bundesland	Team	2002	2003	2004
Bücher	Bayern	Hanna	14.788,00 €	30.222,00 €	23.490,00 €

Artikel	Bundesland	Team	2002	2003	2004
Süßigkeiten	Bayern	Hanna	26.388,00 €	15.641,00 €	32.849,00 €
Schreibwaren	Bayern	Hanna	16.569,00 €	32.675,00 €	25.396,00 €
Bücher	Bayern	Volker	21.961,00 €	21.242,00 €	29.009,00 €
Süßigkeiten	Bayern	Volker	26.142,00 €	22.407,00 €	32.841,00 €
Schreibwaren	Bayern	Volker	29.149,00 €	18.320,00 €	34.429,00 €
Bücher	Hessen	Sigrid	21.845,00 €	33.503,00 €	32.200,00 €
Süßigkeiten	Hessen	Sigrid	23.799,00 €	23.597,00 €	23.020,00 €
Schreibwaren	Hessen	Sigrid	28.328,00 €	17.930,00 €	23.303,00 €
Bücher	Hessen	Katrin	22.797,00 €	31.386,00 €	39.490,00 €
Süßigkeiten	Hessen	Katrin	13.613,00 €	16.174,00 €	35.163,00 €
Schreibwaren	Hessen	Katrin	17.103,00 €	32.563,00 €	35.804,00 €
Bücher	Berlin	Michael	29.952,00 €	19.133,00 €	33.480,00 €
Süßigkeiten	Berlin	Michael	15.348,00 €	31.094,00 €	39.722,00 €
Schreibwaren	Berlin	Michael	24.358,00 €	27.236,00 €	27.129,00 €
Bücher	Berlin	Andy	17.199,00 €	26.386,00 €	30.450,00 €
Süßigkeiten	Berlin	Andy	11.628,00 €	18.232,00 €	28.953,00 €
Schreibwaren	Berlin	Andy	23.828,00 €	32.031,00 €	37.551,00 €

Der Aufbau der Daten

Die Daten der Tabelle 210 werden von dem Makro im Listing 457 aufgebaut, das auch die Formatierung besorgt. Achten Sie auf folgende Techniken:

- Die Generierung von Zufallszahlen.
- Das Einfügen aller Daten auf einmal mit der Methode setDataArray().
- Das Zentrieren der Überschriften und die Einstellung der Hintergrundfarbe.
- Die Formatierung einer Zelle als Währung.

Listing 457. *Erzeugung des Dokuments für die Datenpilot-Beispiele.*

```
Function CreatePivotTableDoc()
  Dim oDoc          'Referenz auf das neu erzeugte Calc-Dokument.
  Dim oSheet        'Das erste Tabellenblatt
  Dim oRange        'Zellbereich
  Dim oData         'Datenarray
  Dim oFormats      'Zahlenformate

  oData = Array(Array("Artikel", "Bundesland", "Team", "2002", "2003", "2004"), _
            Array("Bücher", "Bayern", "Hanna", 14788, 30222, 23490), _
            Array("Süßigkeiten", "Bayern", "Hanna", 26388, 15641, 32849), _
            Array("Schreibwaren", "Bayern", "Hanna", 16569, 32675, 25396), _
            Array("Bücher", "Bayern", "Volker", 21961, 21242, 29009), _
            Array("Süßigkeiten", "Bayern", "Volker", 26142, 22407, 32841), _
            Array("Schreibwaren", "Bayern", "Volker", 29149, 18320, 34429), _
            Array("Bücher", "Hessen", "Sigrid", 21845, 33503, 32200), _
            Array("Süßigkeiten", "Hessen", "Sigrid", 23799, 23597, 23020), _
            Array("Schreibwaren", "Hessen", "Sigrid", 28328, 17930, 23303), _
```

```
            Array("Bücher", "Hessen", "Katrin", 22797, 31386, 39490), _
            Array("Süßigkeiten", "Hessen", "Katrin", 13613, 16174, 35163), _
            Array("Schreibwaren", "Hessen", "Katrin", 17103, 32563, 35804), _
            Array("Bücher", "Berlin", "Michael", 29952, 19133, 33480), _
            Array("Süßigkeiten", "Berlin", "Michael", 15348, 31094, 39722), _
            Array("Schreibwaren", "Berlin", "Michael", 24358, 27236, 27129), _
            Array("Bücher", "Berlin", "Andy", 17199, 26386, 30450), _
            Array("Süßigkeiten", "Berlin", "Andy", 11628, 18232, 28953), _
            Array("Schreibwaren", "Berlin", "Andy", 23828, 32031, 37551))

  'Das Einfügen der Daten
  oDoc = StarDesktop.loadComponentFromURL(_
                    "private:factory/scalc", "_default", 0, Array())
  oSheet = oDoc.Sheets(0)
  oSheet.getCellRangeByName("A1:F19").setDataArray(oData)

  'Der Titelbereich
  oRange = oSheet.getCellRangeByName("A1:F1")
  oRange.HoriJustify = com.sun.star.table.CellHoriJustify.CENTER
  oRange.CellBackColor = RGB(225, 225, 225)

  'Zahlen als Währung formatiert
  oFormats = oDoc.NumberFormats
  Dim aLocale As New com.sun.star.lang.Locale
  oRange = oSheet.getCellRangeByName("D2:F19")
  oRange.NumberFormat = oFormats.getStandardFormat(_
      com.sun.star.util.NumberFormat.CURRENCY, aLocale)
  CreatePivotTableDoc = oDoc
End Function
```

Erzeugung der Datenpilot-Tabelle

Das Makro im Listing 458 erzeugt eine Datenpilot-Tabelle und fügt sie ein:

1. Der Datenpilot-Deskriptor wird mit der Methode createDataPilotDescriptor() erzeugt.
2. Der zu nutzende Datenquellbereich wird festgelegt.
3. Es wird konfiguriert, welche Spalte zu welchem Zweck genutzt wird.
4. Der Datenpilot-Deskriptor wird in die Reihe der Datenpilot-Tabellen eingefügt.

Listing 458. *Erzeugt eine Datenpilot-Tabelle.*

```
Sub CreateDataPilotTable()
  Dim oSheet         'Tabellenblatt, das den Datenpiloten enthält
  Dim oRange         'Quellbereich für den Datenpiloten
  Dim oRangeAddress 'Die Adresse des Objekts oRange
  Dim oTables        'Kollektion der Datenpilot-Tabellen
  Dim oTDescriptor  'Ein einzelner Datenpilot-Deskriptor
  Dim oFields        'Kollektion aller Felder
  Dim oField         'Ein einzelnes Feld
  Dim oCellAddress As New com.sun.star.table.CellAddress
  Dim oDoc           'Das neue Datendokument
  Dim oColumns       'Die Tabellenspalten
  Dim i%             'Zählindex

  oDoc = CreatePivotTableDoc()
  oSheet = oDoc.Sheets.getByIndex(0)
```

```
    oRange = oSheet.getCellRangeByName("A1:F19")

    REM Sicherlich könnte ich die Adresse einfach angeben, aber so macht es mehr Spaß!
    REM Legt die Zieladresse zwei Zeilen unterhalb der Daten fest.
    oRangeAddress = oRange.getRangeAddress()
    oCellAddress.Sheet  = oRangeAddress.Sheet
    oCellAddress.Column = oRangeAddress.StartColumn
    oCellAddress.Row    = oRangeAddress.EndRow + 2

    oTables = oSheet.getDataPilotTables()

    REM Schritt 1, Erzeugung des Deskriptors
    oTDescriptor = oTables.createDataPilotDescriptor()

    REM Schritt 2, Festlegung des Quellbereichs
    oTDescriptor.setSourceRange(oRangeAddress)

    REM Schritt 3, Festlegung der Felder
    oFields = oTDescriptor.getDataPilotFields()

    REM Spalte 0 in der Quelle ist Artikel, es soll ein Zeilenelement sein.
    oField = oFields.getByIndex(0)
    oField.Orientation = com.sun.star.sheet.DataPilotFieldOrientation.ROW

    REM Spalte 1 in der Quelle ist Bundesland, es soll ein Spaltenelement sein.
    oField = oFields.getByIndex(1)
    oField.Orientation = com.sun.star.sheet.DataPilotFieldOrientation.COLUMN

    REM Spalte 3 in der Quelle ist 2002. Dafür wird die Summe gebildet!
    oField = oFields.getByIndex(3)
    oField.Orientation = com.sun.star.sheet.DataPilotFieldOrientation.DATA
    oField.Function = com.sun.star.sheet.GeneralFunction.SUM

    oTables.insertNewByName("Mein erster Datenpilot", oCellAddress, oTDescriptor)

    oColumns = oSheet.getColumns
    For i = 0 To 5
      oColumns(i).OptimalWidth = True    'Spaltenbreite: optimale Breite
    Next
End Sub
```

21	Filter				
22					
23	Summe - 2002	Bundeslan			
24	Artikel	Bayern	Berlin	Hessen	**Gesamt Ergebnis**
25	Bücher	36,749.00 €	47,151.00 €	44,642.00 €	**128,542.00 €**
26	Schreibwaren	45,718.00 €	48,186.00 €	45,431.00 €	**139,335.00 €**
27	Süßigkeiten	52,530.00 €	26,976.00 €	37,412.00 €	**116,918.00 €**
28	**Gesamt Ergebnis**	**134,997.00 €**	**122,313.00 €**	**127,485.00 €**	**384,795.00 €**

Bild 116. *Das Makro im Listing 458 fügt die Datenpilot-Tabelle unterhalb der Quelldaten ein.*

Eingriff in die Kollektion der Datenpilot-Tabellen

Die von jedem Tabellenblatt unterstützte Methode getDataPilotTables() gibt ein Objekt zurück, das den Service com.sun.star.sheet.DataPilotTables unterstützt. Dieser Service bietet den Zugang zu den Datenpilot-Tabellen des Tabellenblatts über sowohl indexierten als auch enumerierten Zugriff. Das Objekt DataPilotTables unterstützt zusätzlich die Methoden der Tabelle 211.

Tabelle 211. *Methoden im Interface com.sun.star.sheet.XDataPilotTables.*

Methode	Beschreibung
createDataPilotDescriptor()	Erzeugt einen neuen Datenpilot-Deskriptor.
insertNewByName(name, CellAddress, DataPilotDescriptor)	Fügt der Kollektion eine neue Datenpilot-Tabelle hinzu, in der die angegebene Zelladresse (s. Tabelle 172) die linke oberen Ecke der Tabelle ist.
removeByName(name)	Löscht eine Datenpilot-Tabelle aus der Kollektion.

Datenpilot-Felder

Jedes „Feld“ in der erzeugten Datenpilot-Tabelle entspricht einer Spalte in der Datenpilot-Quelle (Zellbereich) und erhält seinen Namen aus der obersten Zelle der Spalte in dem Bereich. Der Feldname ist über die Methoden getName() und setName(String) erreichbar.

Jedes Feld enthält die Eigenschaft Orientation, die vom Typ DataPilotFieldOrientation ist und die Ausrichtung des Feldes in der Ausgabetabelle festlegt (s. Tabelle 212). Die Eigenschaft Function bestimmt als Wert der Enumeration GeneralFunction die Funktion zur Berechnung des Resultats für dieses Feld (s. Tabelle 193).

Tabelle 212. *Die Enumeration com.sun.star.sheet.DataPilotFieldOrientation.*

Wert	Beschreibung
HIDDEN	Das Feld wird nicht genutzt.
COLUMN	Das Feld wird als Spaltenfeld genutzt.
ROW	Das Feld wird als Zeilenfeld genutzt.
PAGE	Das Feld wird als Seitenfeld genutzt.
DATA	Das Feld wird als Datenfeld genutzt.

Datenpilot-Tabellen

Jede Datenpilot-Tabelle basiert auf einem Zellbereich eines Tabellenblatts. Jede Datenpilot-Tabelle unterstützt die Objektmethode getOutputRange(), die eine Adresse als CellRangeAddress zurückgibt (s. Tabelle 180). Die Methode refresh() baut die Tabelle auf der Grundlage der aktuellen Daten im Quellbereich neu auf. Jede Datenpilot-Tabelle unterstützt auch den Service DataPilotDescriptor, der die Methoden in der Tabelle 213 definiert.

Tabelle 213. *Methoden im Interface com.sun.star.sheet.XDataPilotDescriptor.*

Methode	Beschreibung
getTag()	Liest den in der Datenpilot-Tabelle gespeicherten zusätzlichen String.
setTag(String)	Setzt den in der Datenpilot-Tabelle gespeicherten zusätzlichen String.
getSourceRange()	Gibt die Adresse des Zellbereichs (CellRangeAddress s. Tabelle 180) zurück, aus dem die Daten für die Datenpilot-Tabelle stammen.
setSourceRange (CellRangeAddress)	Setzt den Zellbereich mit den Quelldaten für die Datenpilot-Tabelle.
getFilterDescriptor()	Zugriff auf den SheetFilterDescriptor (s. Tabelle 219), der festlegt, welche Daten aus dem Quellbereich für die Datenpilot-Tabelle verwendet werden.
getDataPilotFields()	Zugriff auf die Datenpilotfelder, als Objekt, das indexierten Zugriff erlaubt.
getColumnFields()	Zugriff auf die als Spaltenfelder genutzten Datenpilotfelder, als Objekt, das indexierten Zugriff erlaubt.
getRowFields()	Zugriff auf die als Zeilenfelder genutzten Datenpilotfelder, als Objekt, das indexierten Zugriff erlaubt.

Methode	Beschreibung
getPageFields()	Zugriff auf die als Seitenfelder genutzten Datenpilotfelder, als Objekt, das indexierten Zugriff erlaubt.
getDataFields()	Zugriff auf die als Datenfelder genutzten Datenpilotfelder, als Objekt, das indexierten Zugriff erlaubt.
getHiddenFields()	Zugriff auf die Datenpilotfelder, die nicht als Spalten-, Zeilen-, Seiten- oder Datenfelder genutzt werden, als Objekt, das indexierten Zugriff erlaubt.

Datenpilot-Felder filtern

Eine Datenpilot-Tabelle kann über die Methode getFilterDescriptor() die Anzeige der Felder in der erzeugten Tabelle unter definierten Bedingungen filtern. Beachten Sie, dass der zurückgegebene Filterdeskriptor das Interface com.sun.star.sheet.XSheetFilterDescriptor ist und nicht com.sun.star.sheet.XSheetFilterDescriptor2. Eine Beschreibung der Unterschiede und der Filtereinstellungen finden Sie im Abschnitt 15.6.3. Filter.

15.5.7. Tabellenblattcursors

In einem Calc-Dokument ist ein Cursor ein Zellbereich, der Methoden anbietet, sich durch die enthaltenen Zellen zu bewegen. Cursors werden in Calc-Dokumenten nicht so häufig verwendet wie in Writer-Dokumenten, weil der größte Teil des Inhalts im Gegensatz zu Textdokumenten direkt über den Index oder den Namen erreichbar ist. Tabellenblattcursors sind wie Zellbereiche auf ein Tabellenblatt beschränkt. Der von Calc-Dokumenten verwendete Service SheetCellCursor ist mit den Zellcursors von Texttabellen vergleichbar (s. Tabelle 214).

Tabelle 214. *Die Hauptkomponenten des Service com.sun.star.sheet.SheetCellCursor.*

Komponente	Beschreibung
com.sun.star.table.CellCursor	Methoden zur Kontrolle der Position eines Zellcursors.
com.sun.star.table.CellRange	Methoden zum Zugriff auf Zellen oder Unterbereiche eines Zellbereichs (s. Tabelle 188).
com.sun.star.sheet.XSheetCellCursor	Erweiterte Methoden zur Kontrolle der Position des Cursors.
com.sun.star.sheet.SheetCellRange	Ein rechteckiger Zellbereich in einem Tabellendokument. Dies ist eine Erweiterung des Service CellRange zum Gebrauch in einem Tabellendokument.
com.sun.star.sheet.XUsedAreaCursor	Methoden zur Suche nach Bereichen mit Inhalt in einem Tabellenblatt.

Die wichtigsten vom Service SheetCellCursor unterstützten Methoden finden Sie in der Tabelle 215.

Tabelle 215. *Die Hauptmethoden des Service com.sun.star.sheet.SheetCellCursor.*

Interface	Methode	Beschreibung
XCellCursor	gotoStart()	Verschiebt den Cursor zur ersten Zelle mit Inhalt am Anfang einer zusammenhängenden Reihe von Zellen mit Inhalt. Diese Zelle kann außerhalb des Cursorbereichs liegen.
XCellCursor	gotoEnd()	Verschiebt den Cursor zur letzten Zelle mit Inhalt am Ende einer zusammenhängenden Reihe von Zellen mit Inhalt. Diese Zelle kann außerhalb des Cursorbereichs liegen.
XCellCursor	gotoNext()	Verschiebt den Cursor zur nächsten (nach rechts) ungeschützten Zelle.
XCellCursor	gotoPrevious()	Verschiebt den Cursor zur vorherigen (nach links) ungeschützten Zelle.
XCellCursor	gotoOffset(nSpalte, nZeile)	Versetzt den Cursorbereich relativ zur aktuellen Position. Negative Zahlen bewirken einen Versatz nach links und oben, positive Zahlen nach rechts und unten.

Interface	Methode	Beschreibung
XCellRange	getCellByPosition(links, oben)	Zugriff auf eine Zelle innerhalb des Bereichs.
XCellRange	getCellRangeByPosition (links, oben, rechts, unten)	Zugriff auf einen Zellbereich innerhalb des Bereichs.
XCellRange	getCellRangeByName(name)	Zugriff auf einen Zellbereich innerhalb des Bereichs über den Namen. Der String referenziert die Zellen direkt im Standardformat – etwa „B2:D5“ oder „$B$2“ – oder über definierte Zellbereichsnamen.
XSheetCellCursor	collapseToCurrentRegion()	Dehnt den Bereich auf alle nicht-leeren zusammenhängenden Zellen aus.
XSheetCellCursor	collapseToCurrentArray()	Dehnt den Bereich auf die aktuelle Matrixformel aus.
XSheetCellCursor	collapseToMergedArea()	Dehnt den Bereich auf die zusammenhängenden Zellen aus, die den Bereich schneiden.
XSheetCellCursor	expandToEntireColumns()	Dehnt den Bereich auf alle Spalten aus, die den Bereich schneiden.
XSheetCellCursor	expandToEntireRows()	Dehnt den Bereich auf alle Zeilen aus, die den Bereich schneiden.
XSheetCellCursor	collapseToSize (nSpalten, nZeilen)	Setzt die Größe des Cursorbereichs, ohne die linke obere Ecke zu verändern.
XSheetCellRange	getSpreadsheet()	Zugriff auf das Tabellenblattobjekt, in dem der Zellbereich liegt.
XUsedAreaCursor	gotoStartOfUsedArea()	Setzt den Cursor an den Anfang des verwendeten Bereichs.
XUsedAreaCursor	gotoEndOfUsedArea()	Setzt den Cursor an das Ende des verwendeten Bereichs.

Zellbereiche, somit auch Zellcursors, sind rechteckige Gebiete. Die Verwendung rechteckiger Gebiete mag jetzt, wo ich es sage, einleuchtend sein, aber als ich die Methoden gotoStart() und gotoEnd() der Tabelle 215 getestet habe, war es doch eine Überraschung für mich. Als ich den Code im Listing 459 schrieb, startete ich mit der im Bild 108 zu sehenden Konfiguration.

Listing 459. *Einfache Befehle zur Cursorverschiebung in zusammenhängenden Blöcken.*

```
oCurs = oSheet.createCursorByRange(oSheet.getCellRangeByName("C3"))
oCurs.gotoStart()  REM Verschiebt den Cursor zur Zelle B1
oCurs.gotoEnd()    REM Verschiebt den Cursor zur Zelle E8
oCurs.gotoStart()  REM Verschiebt den Cursor zur Zelle E8
```

Die erste Zeile im Listing 459 positioniert den Cursor auf der Zelle C3, mitten in einem Block mit Werten. Im Bild 108 ist B die am weitesten links stehende zusammenhängende Spalte und 1 ist die oberste zusammenhängende Zeile. Die Methode gotoStart() verschiebt den Cursor daher in die obere linke Ecke zur Position B1. Nun beginnen die Dinge, sich etwas unerwartet zu entwickeln. Die am weitesten rechts stehende zusammenhängende Spalte ist E und die unterste zusammenhängende Zeile ist 8. Die Methode gotoEnd() verschiebt daher den Cursor zur Position E8. Wie man im Bild 108 sehen kann, ist die Zelle E8 komplett von der zusammenhängenden Zellgruppe abgeschnitten. Der Cursor würde auch dann auf die Zelle E8 gesetzt, wenn sie gar keinen Wert enthielte. Nun hat der Cursor keinen Bezug mehr zum ursprünglichen Zellblock. Also verschiebt die Methode gotoStart() den Cursor auch nicht zurück zur Zelle B1.

Um das Verhalten des Listing 459 zu verstehen, ist es wichtig zu erkennen, wie OOo zusammenhängende Zellen ermittelt. An keiner mir bekannten Stelle ist das dokumentiert. Ich habe experimentell festgestellt, dass der Satz zusammenhängender nicht-leerer Zellen als der kleinste Bereich (rechteckiger Zellblock) definiert ist, der von leeren Zellen umschlossen wird. Wenn Zelle E9 einen Wert enthielte, dann würden die Zellen E8 und E9, obwohl nicht direkt über nicht-leere Zellen mit dem ur-

sprünglichen Zellblock verbunden, beide als Teil des Blocks zusammenhängender nicht-leerer Zellen gewertet.

Die Methode collapseToCurrentRegion() bewirkt, dass der Cursor den Block zusammenhängender Zellen enthält. Allerdings ist zu beachten, dass nach der Anweisung der ursprüngliche Bereich immer Teil des neuen Bereichs ist, auch wenn dadurch überzählige leere Zellen eingeschlossen sind. Ähnlich ist es bei der Methode collapseToCurrentArray(), außer dass sie einen Bereich zurückgibt, der eine Matrixformel umfasst. Die obere linke Ecke des Gebiets muss zum Funktionieren der Methode collapseToCurrentArray() in einer Matrixformel sein.

Der Codeschnipsel im Listing 460 erzeugt einen Cursor über einem Bereich und zeigt dann, dass sich getCellByPosition() und getCellRangeByPosition() relativ zur oberen linken Ecke des Bereichs verhalten. Die Methode getCellRangeByName() erzeugt einen Laufzeitfehler, wenn eine Zelle außerhalb des Bereichs angefordert wird.

Listing 460. *Manche Befehle arbeiten nur relativ zum Bereich.*

```
oCurs  = oSheet.createCursorByRange(oSheet.getCellRangeByName("C3:F12"))
oCell  = oCurs.getCellByPosition(0, 0)            REM Zelle C3
oRange = oCurs.getCellRangeByPosition(1, 0, 3, 2) REM D3:F5
oRange = oCurs.getCellRangeByName("C4:D6")        REM C4:D6
oRange = oCurs.getCellRangeByName("C2:D6")        REM Error: C2 nicht im Bereich!
```

Tipp Die Methoden getCellByPosition(), getCellRangeByPosition() und getCellRangeByName() können keinen Wert zurückgeben, der nicht im Bereich liegt.

15.6. Calc-Dokumente

Viele der Methoden und Eigenschaften auf Dokumentebene betreffen das gesamte Dokument – zum Beispiel, um ein Dokument zu speichern und zu drucken. Andere Methoden und Eigenschaften sind einfach nur praktisch, die Informationen sind auch auf der Ebene der Tabellenblätter vorhanden. Zum Beispiel fungiert das Calc-Dokument als Folienbehälter zum Zugriff auf alle Folien, auch wenn jede einzelne von ihnen vom zugehörigen Tabellenblatt aus erreichbar ist.

15.6.1. Bereichsname

Die offizielle Definition eines Bereichsnamens ist ein benannter Formelausdruck. Normalerweise steht ein Bereichsname für einen Zellbereich, er kann aber auch für externe Daten stehen. Die Benennung von Bereichen erlaubt es Ihnen, den referenzierten Dingen sprechende Namen zu verleihen. Bereichsnamen können daher in Formeln als Adressen dienen. Wenn man zum Beispiel einen Bereich namens „Auswertung“ hat, kann man die Formel „=SUMME(Auswertung)“ verwenden. Über die Konstanten der Gruppe NamedRangeFlag legen Sie fest, wie ein Bereichsname zu nutzen ist (s. Tabelle 216).

Tabelle 216. *Die Konstantengruppe com.sun.star.sheet.NamedRangeFlag.*

Wert	Name	Beschreibung
1	FILTER_CRITERIA	Der Bereich enthält Filterkriterien.
2	PRINT_AREA	Der Bereich kann als Druckbereich genutzt werden.
4	COLUMN_HEADER	Der Bereich kann als Spaltenüberschrift zum Drucken genutzt werden.
8	ROW_HEADER	Der Bereich kann als Zeilenüberschrift zum Drucken genutzt werden.

Der Service NamedRange unterstützt die Methoden der Tabelle 217.

Tabelle 217. *Methoden im Service com.sun.star.sheet.NamedRange.*

Methode	Beschreibung
getReferredCells()	Zugriff auf den vom Bereichsnamen referenzierten Zellbereich.

Methode	Beschreibung
getContent()	Der Inhalt des benannten Bereichs ist ein String und kann eine Referenz sein auf eine Zelle, einen Zellbereich oder einen Formelausdruck.
setContent(String)	Setzt den Inhalt des benannten Bereichs.
getReferencePosition()	Ermittelt die Zelladresse, die als Basis für relative Bezüge im Inhalt dient.
setReferencePosition(CellAddress)	Setzt die Bezugsposition.
getType()	Ermittelt den Typ als NamedRangeFlag-Konstante (s. Tabelle 216).
setType(NamedRangeFlag)	Setzt den Typ des Bereichsnamens.

Die Dokumenteigenschaft NamedRanges enthält die Kollektion aller Bereichsnamen im Dokument. Man kann auf jeden einzelnen Namen über Namen oder Index zugreifen.

Die Methode addNewByName() akzeptiert vier Argumente: Name, Inhalt, Position und Typ. Das vierte Argument ist eine Kombination von Flags, die die Nutzung des Bereichsnamens bestimmt – der gebräuchlichste Wert ist 0, der allerdings kein definierter Konstantenwert ist. Name und Inhalt sind beide vom Typ String. Das dritte Argument Position enthält die Basiszelladresse für relative Zellbezüge.

Listing 461. *Erzeugt einen Bereichsnamen als Bezug auf $Tabelle1.$B$3:$D$6.*

```
Sub AddNamedRange()
  Dim oRange     'Der erzeugte Bereich
  Dim oRanges    'Alle Bereichsnamen
  Dim sName$     'Name des neuen benannten Bereichs
  Dim oCell      'Zellobjekt
  Dim s$

  sName$ = "MeinBName"
  oRanges = ThisComponent.NamedRanges
  If Not oRanges.hasByName(sName$) Then
    REM Ich kann die Zelladresse auch dadurch bestimmen, dass ich erst auf die
    REM Zelle zugreife und dann die Adresse aus dem Zellobjekt abfrage.
    Dim oCellAddress As New com.sun.star.table.CellAddress
    oCellAddress.Sheet = 0      'Das erste Tabellenblatt
    oCellAddress.Column = 1     'Spalte B
    oCellAddress.Row = 2        'Zeile 3

    REM Das erste Argument ist der Bereichsname.
    REM Das zweite Argument ist eine Formel oder ein Ausdruck.
    REM Normalerweise ist es ein String, der einen Bereich definiert.
    REM Das dritte Argument legt die Basisadresse für relative Zellbezüge fest.
    REM Das vierte Argument ist ein Satz von Flags für die Nutzung des Bereichs.
    REM Es gelten die Werte der Konstantengruppe NamedRangeFlag,
    REM aber die meisten Bereiche verwenden 0.
    s$ = "$Tabelle1.$B$3:$D$6"
    oRanges.addNewByName(sName$, s$, oCellAddress, 0)
  End If
  REM Zugriff auf einen Bereich über den neuen Bereichsnamen.
  oRange = ThisComponent.NamedRanges.getByName(sName$)

  REM Ausgabe des Strings aus der Zelle $Tabelle1.$B$3
  oCell = oRange.getReferredCells().getCellByPosition(0, 0)
  Print oCell.getString()
End Sub
```

Das dritte Argument, eine Zelladresse, dient als Basisadresse für relative Zellbezüge. Wenn der Zellbereich nicht als absolute Adresse angegeben ist, wird der referenzierte Bereich unterschiedlich sein, je nachdem, an welcher Position im Tabellenblatt der Bereich genutzt wird. Das relative Verhalten zeigt sich im Listing 462, in dem auch ein weiteres Anwendungsgebiet eines Bereichsnamens demonstriert wird – die Definition einer Gleichung. Das Makro erzeugt den Bereichsnamen „AddLinks", der auf der Grundlage der Referenzzelle C3 die Gleichung „A3+B3" benennt. Die Zellen A3 und B3 sind die beiden Zellen, die links an C3 anschließen, so dass die Gleichung „=AddLinks" die Summe der beiden Zellen liefert, die direkt links neben der Zelle mit der Gleichung sind. Wenn nun die Referenzzelle auf C4 geändert wird – das heißt, auf eine Zelle unterhalb A3 und B3 –, so wird die Gleichung AddLinks die Summe der beiden Zellen links in der Zeile über der aktuellen Zelle berechnen.

Listing 462. *Erzeugung des Bereichsnamens AddLinks.*

```
Sub AddNamedFunction()
  Dim oSheet          'Tabellenblatt, das den Bereichsnamen enthält
  Dim oCellAddress    'Adresse des relativen Bezugs
  Dim oRanges         'Alle Bereichsnamen
  Dim oRange          'Ein einzelner Zellbereich
  Dim sName As String 'Name der zu erzeugenden Gleichung

  sName = "AddLinks"
  oRanges = ThisComponent.NamedRanges
  If Not oRanges.hasByName(sName) Then
    oSheet = ThisComponent.getSheets().getByIndex(0)
    oRange = oSheet.getCellRangeByName("C3")
    oCellAddress = oRange.getCellAddress()
    oRanges.addNewByName(sName, "A3+B3", oCellAddress, 0)
  End If
End Sub
```

Mit der Methode addNewFromTitles(CellRangeAddress, Rand) werden die Bereichsnamen aus Titeln in einem Zellbereich erstellt. Der Wert aus der Enumeration Border (s. Tabelle 218) wird für den Rand des Zellbereichs gewählt, in dem die Titel stehen.

Tabelle 218. *Die Enumeration com.sun.star.sheet.Border.*

Wert	Beschreibung
TOP	Der obere Rand.
BOTTOM	Der untere Rand.
RIGHT	Der rechte Rand.
LEFT	Der linke Rand.

Das nächste Makro erzeugt drei Bereichsnamen aus der oberen Reihe eines Zellbereichs.

Listing 463. *Erzeugung mehrfacher Bereichsnamen.*

```
Sub AddManyNamedRanges()
  Dim oSheet    'Tabellenblatt, das den Bereichsnamen enthält
  Dim oAddress  'Bereichsadresse
  Dim oRanges   'Alle Bereichsnamen
  Dim oRange    'Ein einzelner Zellbereich

  oRanges = ThisComponent.NamedRanges
  oSheet = ThisComponent.getSheets().getByIndex(0)
  oRange = oSheet.getCellRangeByName("A1:C20")
  oAddress = oRange.getRangeAddress()
  oRanges.addNewFromTitles(oAddress, com.sun.star.sheet.Border.TOP)
End Sub
```

Mit der Methode outputList(CellAddress) wird eine Liste der Bereichsnamen in ein Tabellenblatt geschrieben. Die erste Spalte enthält den jeweiligen Bereichsnamen, die zweite Spalte den Inhalt. Und schließlich gibt es noch die Methode removeByName(name), mit der ein Bereichsname gelöscht wird.

15.6.2. Datenbankbereich

Obwohl ein Datenbankbereich wie ein normaler benannter Bereich benutzt werden kann, so ist er doch auch als Zellbereich eines Tabellenblatts definiert, der wie eine Datenbank nutzbar ist. Jede Zeile in einem solchen Bereich gilt als Datensatz und jede Zelle gilt als Feld. Man kann den Bereich sortieren, gruppieren, durchsuchen und auf seiner Grundlage Berechnungen durchführen, genauso wie mit einer Datenbank.

Ein Datenbankbereich bietet die Funktionalitäten, die für datenbankspezifische Aktivitäten nützlich sind. Zum Beispiel können Sie die erste Zeile als Überschriften kennzeichnen. Im GUI öffnen Sie über das Menü **Daten | Bereich festlegen** den Dialog „Datenbankbereich festlegen“.

In einem Makro ist es die Eigenschaft DatabaseRanges, über die Sie auf einen Datenbankbereich zugreifen und einen Datenbankbereich erzeugen und löschen können. Das folgende Makro erzeugt einen Datenbankbereich mit dem Namen „MeineDatenbank“ und legt den Bereich zur Nutzung als Autofilter fest.

Listing 464. *Erzeugung eines Datenbankbereichs und eines Autofilters.*

```
Sub AddNewDatabaseRange()
  Dim oRange 'Das Objekt DatabaseRange
  Dim oAddr  'Adresse des Zellbereichs für den Datenbankbereich
  Dim oSheet 'Tabellenblatt1, in dem der Bereich sein wird
  Dim oDoc   'Referenz auf ThisComponent in kürzerer Form

  oDoc = ThisComponent
  If Not oDoc.DatabaseRanges.hasByName("MeineDatenbank") Then
    oSheet = ThisComponent.getSheets().getByIndex(0)
    oRange = oSheet.getCellRangeByName("A1:F10")
    oAddr = oRange.getRangeAddress()
    oDoc.DatabaseRanges.addNewByName("MeineDatenbank", oAddr)
  End If
  oRange = oDoc.DatabaseRanges.getByName("MeineDatenbank")
  oRange.AutoFilter = True
End Sub
```

15.6.3. Filter

Mit Filtern beschränken Sie die Anzahl der sichtbaren Zeilen in einem Tabellenblatt. Allgemeine Filter, die allen Arten der Datenmanipulation gemeinsam sind, werden automatisch von der Funktionalität des Autofilters bereitgestellt. Sie können auch Ihre eigenen Filter definieren.

Alle Filterbedingungen werden im Service SheetFilterDescriptor gekapselt, der von einem Zellbereich, also auch von einem Tabellenblatt mit der Methode createFilterDescriptor(Boolean) erzeugt wird. True bedeutet, dass ein neuer, leerer Filterdeskriptor erzeugt wird. False bedeutet, dass der vorhandene Deskriptor verwendet wird.

Der Deskriptor unterstützt die Methoden setFilterFields() und getFilterFields(), um Filterfelder als Array von TableFilterField-Structs anzulegen oder um darauf zuzugreifen. Die vom Service SheetFilterDescriptor definierten Eigenschaften steuern den Filtervorgang (s. Tabelle 219.

Tabelle 219. *Eigenschaften im Service com.sun.star.sheet.SheetFilterDescriptor.*

Eigenschaft	Beschreibung
IsCaseSensitive	Falls True, wird bei einem Stringvergleich Groß- und Kleinschreibung unterschieden.
SkipDuplicates	Falls True, werden keine Ergebnisduplikate in das Resultat aufgenommen.
UseRegularExpressions	Falls True, werden Stringwerte im Struct TableFilterField als reguläre Ausdrücke interpretiert.
SaveOutputPosition	Falls True (und CopyOutputData ist True), wird OutputPosition für einen späteren Aufruf gespeichert.
Orientation	Legt fest, ob Spalten oder Zeilen gefiltert werden, als Enumeration TableOrientation (s. Tabelle 207).
ContainsHeader	Falls True, wird die erste Zeile (oder Spalte) als Überschrift angesehen und nicht gefiltert.
CopyOutputData	Falls True, werden die gefilterten Daten an die OutputPosition kopiert.
OutputPosition	Legt fest, wohin die gefilterten Daten kopiert werden, als Zelladresse (s. Tabelle 172).
MaxFieldCount	Die maximale Anzahl Filterfelder im Deskriptor, als Long Integer.

Jedes einzelne Filterfeld wird in einem TableFilterField-Struct gespeichert (s. Tabelle 220).

Tabelle 220. *Eigenschaften des Structs com.sun.star.sheet.TableFilterField.*

Eigenschaft	Beschreibung
Connection	Legt fest, wie die Bedingung mit der vorherigen Bedingung verknüpft ist, als FilterConnection (s. Tabelle 222).
Field	Legt fest, auf welches Feld (Spalte) die Bedingung angewendet wird, als Long Integer.
Operator	Legt den Typ der Bedingung fest, als FilterOperator (s. Tabelle 221).
IsNumeric	Falls True, wird die Eigenschaft NumericValue genutzt, ansonsten die Eigenschaft StringValue
NumericValue	Legt einen numerischen Wert für die Bedingung fest, als Double.
StringValue	Legt einen Stringwert für die Bedingung fest.

Der Operator enthält einen Wert aus der Enumeration com.sun.star.sheet.FilterOperator (s. Tabelle 221).

Tabelle 221. *Die Enumeration com.sun.star.sheet.FilterOperator.*

Name	Wert	Beschreibung
EMPTY	0	Wählt leere Einträge aus.
NOT_EMPTY	1	Wählt nicht-leere Einträge aus.
EQUAL	2	Der Wert des Eintrags muss gleich dem angegebenen Wert sein.
NOT_EQUAL	3	Der Wert des Eintrags muss ungleich dem angegebenen Wert sein.
GREATER	4	Der Wert des Eintrags muss größer als der angegebene Wert sein.
GREATER_EQUAL	5	Der Wert des Eintrags muss größer als oder gleich groß wie der angegebene Wert sein.
LESS	6	Der Wert des Eintrags muss kleiner als der angegebene Wert sein.
LESS_EQUAL	7	Der Wert des Eintrags muss kleiner als oder gleich groß wie der angegebene Wert sein.
TOP_VALUES	8	Wählt eine angegebene Anzahl an Einträgen mit den höchsten Werten aus.
TOP_PERCENT	9	Wählt eine prozentuale Menge an Einträgen mit den höchsten Werten aus.
BOTTOM_VALUES	10	Wählt eine angegebene Anzahl an Einträgen mit den niedrigsten Werten aus.
BOTTOM_PERCENT	11	Wählt eine prozentuale Menge an Einträgen mit den niedrigsten Werten aus.

Die Bedingungen werden über die Enumeration FilterConnection kombiniert (s. Tabelle 222).

Tabelle 222. *Die Enumeration com.sun.star.sheet.FilterConnection.*

Wert	Beschreibung
AND	Beide Bedingungen müssen zutreffen.
OR	Mindestens eine Bedingung muss zutreffen.

Seit OOo 3.2 ist die Liste der Operatoren erweitert und nicht mehr als Enumeration, sondern als Konstantengruppe com.sun.star.sheet.FilterOperator2 definiert. Damit beide Zuordnungen möglich bleiben, bietet der Filterdeskriptor neben den Methoden setFilterFields() und getFilterFields() auch die Methoden setFilterFields2() und getFilterFields2() an, die das Struct com.sun.star.sheet.TableFilterField2 verwenden, dessen einziger Unterschied zum TableFilterField darin besteht, dass als Operator ein Wert aus der Konstantengruppe FilterOperator2 erwartet wird.

Tabelle 223. *Die Konstantengruppe com.sun.star.sheet.FilterOperator2.*

Name	Wert	Beschreibung
EMPTY bis BOTTOM_PERCENT	0 bis 11	Gleich mit den Werten der Enumeration FilterOperator (s. Tabelle 221).
CONTAINS	12	Wählt Einträge aus, die den String enthalten.
DOES_NOT_CONTAIN	13	Wählt Einträge aus, die den String nicht enthalten.
BEGINS_WITH	14	Wählt Einträge aus, die mit dem String beginnen.
DOES_NOT_BEGIN_WITH	15	Wählt Einträge aus, die nicht mit dem String beginnen.
ENDS_WITH	16	Wählt Einträge aus, die mit dem String enden.
DOES_NOT_END_WITH	17	Wählt Einträge aus, die nicht mit dem String enden.

Zusammenfassend ergibt sich folgender Ablauf (mit einem einfachen Beispiel im Listing 465):

1. Sie erzeugen von einem Zellbereich oder einer Tabelle mit createFilterDescriptor(True) einen neuen Filterdeskriptor.
2. Sie legen ein Array von TableFilterField-Structs zur Definition der Filterbedingungen an, für jede zu filternde Spalte ein Arrayelement. Als Operator gelten die Werte aus der Enumeration FilterOperator. Wenn Sie stattdessen ein Array von TableFilterField2-Structs anlegen, dann gelten als Operator die Werte aus der Konstantengruppe FilterOperator2.
3. Sie laden das Array mit der Methode setFilterFields() (beziehungsweise setFilterFields2()) in den Filterdeskriptor.
4. Sie legen die weiteren Eigenschaften des Filterdeskriptors fest.
5. Mit der Zellbereichsmethode filter(Filterdeskriptor) wenden Sie den Filter an.

Listing 465. *Ein einfacher Tabellenblattfilter.*

```
Sub SimpleSheetFilter()
  Dim oSheet       'Tabellenblatt für den Filter
  Dim oFilterDesc  'Filterdeskriptor
  Dim oFields(0) As New com.sun.star.sheet.TableFilterField  'Array Filterbedingungen

  oSheet = ThisComponent.getSheets().getByIndex(0)

  REM Wenn das Argument True ist, wird ein leerer Filterdeskriptor erzeugt.
  REM Wenn das Argument False ist, wird ein Deskriptor
  REM mit den vorherigen Einstellungen erzeugt.
  oFilterDesc = oSheet.createFilterDescriptor(True)
```

```
  With oFields(0)
    REM Ich könnte die Eigenschaft Connection nutzen, um
    REM die Verknüpfung mit dem vorherigen Feld anzugeben.
    REM Dies ist aber das erste Feld, somit ist es nicht nötig.
    '.Connection = com.sun.star.sheet.FilterConnection.AND
    '.Connection = com.sun.star.sheet.FilterConnection.OR

    REM Die Eigenschaft Field ist die nullbasierte Zählung
    REM der zu filternden Spalte. Wenn Sie die Zelle haben,
    REM können Sie auch schreiben:
    '.Field = oCell.CellAddress.Column
    .Field = 5

    REM Sollen Zahlen oder Strings verglichen werden? Zahlen!
    .IsNumeric = True

    REM Es wird die Eigenschaft NumericValue genutzt,
    REM weil oben .IsNumeric = True steht.
    .NumericValue = 80

    REM Wäre IsNumeric False, so würde die Eigenschaft StringValue genutzt.
    REM .StringValue = "was auch immer"

    REM Gültige Operatoren sind:
    REM EMPTY, NOT_EMPTY, EQUAL, NOT_EQUAL,
    REM GREATER, GREATER_EQUAL, LESS, LESS_EQUAL,
    REM TOP_VALUES, TOP_PERCENT, BOTTOM_VALUES und BOTTOM_PERCENT
    REM (s. Tabelle 221)
    .Operator = com.sun.star.sheet.FilterOperator.GREATER_EQUAL
  End With

  REM Der Filterdeskriptor unterstützt die folgenden Eigenschaften:
  REM IsCaseSensitive, SkipDuplicates, UseRegularExpressions,
  REM SaveOutputPosition, Orientation, ContainsHeader,
  REM CopyOutputData, OutputPosition und MaxFieldCount
  REM (s. Tabelle 219).
  oFilterDesc.setFilterFields(oFields())
  oFilterDesc.ContainsHeader = True
  oSheet.filter(oFilterDesc)
End Sub
```

Wenn ein Filter auf ein Tabellenblatt angewendet wird, ersetzt er alle im Tabellenblatt vorhandenen Filter. Um alle Filter eines Tabellenblatts zu löschen, braucht man nur einen leeren Filter einzufügen.

Listing 466. *Löschung des aktuellen Tabellenblattfilters.*

```
Sub RemoveSheetFilter()
  Dim oSheet          'Zu filterndes Tabellenblatt
  Dim oFilterDesc     'Filterdeskriptor

  oSheet = ThisComponent.getSheets().getByIndex(0)
  oFilterDesc = oSheet.createFilterDescriptor(True)
  oSheet.filter(oFilterDesc)
End Sub
```

Ein Filter kann auch mehrere Spalten filtern.

Listing 467. *Ein einfacher Tabellenblattfilter, der zwei Spalten filtert.*

```
Sub SimpleSheetFilter_2()
  Dim oSheet          'Zu filterndes Tabellenblatt
```

```
    Dim oRange          'Zu filternder Bereich
    Dim oFilterDesc     'Filterdeskriptor
    Dim oFields(1) As New com.sun.star.sheet.TableFilterField  'Array Filterbedingungen

    oSheet = ThisComponent.getSheets().getByIndex(0)
    oRange = oSheet.getCellRangeByName("E12:G19")

    REM Wenn das Argument True ist, wird ein leerer Filterdeskriptor erzeugt.
    oFilterDesc = oRange.createFilterDescriptor(True)

    REM Aufbau eines Feldes zur Anzeige von Zellen,
    REM deren Inhalt mit dem Buchstaben b beginnt.
    With oFields(0)
      .Field = 0              'Filterspalte A
      .IsNumeric = False      'String, keine Zahl
      .StringValue = "b.*"    'Regulärer Ausdruck: alles was mit b beginnt
      .Operator = com.sun.star.sheet.FilterOperator.EQUAL
      REM Wenn man als Feldtyp TableFilterField2 gewählt hätte,
      REM bräuchte man keinen regulären Ausdruck, sondern:
      '.StringValue = "b"
      '.Operator = com.sun.star.sheet.FilterOperator2.BEGINS_WITH
    End With

    REM Aufbau eines Feldes, das beide Bedingungen festlegt (Connection)
    REM Diese neue Bedingung erfordert einen Wert größer oder gleich 70.
    With oFields(1)
      .Connection = com.sun.star.sheet.FilterConnection.AND
      .Field = 5              ' Filterspalte F.
      .IsNumeric = True       ' Zahl, kein String.
      .NumericValue = 70      ' Werte größer oder gleich 70
      .Operator = com.sun.star.sheet.FilterOperator.GREATER_EQUAL
    End With

    oFilterDesc.setFilterFields(oFields())
    oFilterDesc.ContainsHeader = False
    oFilterDesc.UseRegularExpressions = True
    oSheet.filter(oFilterDesc)
  End Sub
```

Ein Spezialfilter unterstützt bis zu acht Filterbedingungen im Gegensatz zu nur drei, die vom einfachen Filter akzeptiert werden. Die Kriterien für einen Spezialfilter werden im Tabellenblatt gespeichert. Der erste Schritt zu einem Spezialfilter ist also die Eingabe der Filterkriterien im Tabellenblatt.

1. Suchen Sie sich einen freien Platz im Calc-Dokument. Dieser freie Platz kann in jedem beliebigen Tabellenblatt an jeder nur möglichen Stelle im Dokument sein.
2. Kopieren Sie die Spaltenüberschriften von dem zu filternden Bereich an die Stelle der Filterkriterien.
3. Tragen Sie unter die Spaltenköpfe die Filterkriterien ein. Das Kriterium jeder Spalte einer Zeile wird mit And verknüpft. Die Kriterien jeder Zeile werden mit Or verknüpft.

Die Anwendung eines Spezialfilters mit einem Makro ist einfach. Der Zellbereich mit den Filterkriterien wird zur Erzeugung des Filterdeskriptors herangezogen, der dann den Datenbereich filtern wird.

Listing 468. *Ein Spezialfilter.*

```
Sub UseAnAdvancedFilter()
  Dim oSheet     'Ein Tabellenblatt aus dem Dokument
  Dim oRanges    'Die Eigenschaft NamedRanges
  Dim oCritRange 'Zellbereich mit den Filterkriterien
  Dim oDataRange 'Zellbereich mit den zu filternden Daten
  Dim oFiltDesc  'Filterdeskriptor

  REM Der Zellbereich, der die Filterkriterien enthält
  oSheet = ThisComponent.getSheets().getByIndex(1)
  oCritRange = oSheet.getCellRangeByName("A1:G3")

  REM Sie können den Bereich der Filterkriterien auch
  REM über einen Bereichsnamen festlegen.
  REM oRanges = ThisComponent.NamedRanges
  REM oRange = oRanges.getByName("DurchschnittKleiner80")
  REM oCritRange = oRange.getReferredCells()

  REM Die Daten, die ich filtern will
  oSheet = ThisComponent.getSheets().getByIndex(0)
  oDataRange = oSheet.getCellRangeByName("A1:G16")

  oFiltDesc = oCritRange.createFilterDescriptorByObject(oDataRange)
  oDataRange.filter(oFiltDesc)
End Sub
```

15.6.4. Dokumente und Tabellenblätter schützen

Calc-Dokumente und Tabellenblätter unterstützen das Interface XProtectable. Mit den Methoden protect(password) und unprotect(password) aktivieren oder deaktivieren Sie den Schutz vor Änderungen. Das Passwort wird als String übergeben. Die Methode isProtected() gibt True zurück, wenn der Schutz momentan aktiv ist.

15.6.5. Steuerung der Neuberechnung

Ein Calc-Dokument berechnet Formeln automatisch neu, wenn die Zellen, auf die sich die Formeln beziehen, geändert werden. Gelegentlich ist es aber nützlich, die automatische Neuberechnung auszusetzen. Die Methoden der Tabelle 224 erlauben Ihnen, die Neuberechnung im gesamten Dokument zu steuern.

Tabelle 224. *Methoden im Interface com.sun.star.sheet.XCalculatable.*

Methode	Beschreibung
calculate()	Berechnet alle Zellen mit geändertem Inhalt neu.
calculateAll()	Berechnet alle Zellen neu.
isAutomaticCalculationEnabled()	True, falls die automatische Neuberechnung aktiviert ist.
enableAutomaticCalculation(Boolean)	Aktiviert oder deaktiviert die automatische Neuberechnung.

15.6.6. Zielwertsuche

Die Zielwertsuche versucht, Gleichungen mit einer unbekannten Variablen zu lösen. Nachdem man also eine Formel definiert hat mit mehreren festen Werten und einem variablen Wert, versucht die Zielwertsuche, einen passenden Wert für die unbekannte Variable zu finden.

Nehmen wir ein einfaches Beispiel. Wenn Sie von einer Klippe springen, werden Sie durch die Schwerkraft jede Sekunde um 9,81 Meter beschleunigt. Das heißt, dass Sie in einer Sekunde eine Geschwindigkeit von 9,81 m/s haben und nach zwei Sekunden eine Geschwindigkeit von 19,62 m/s.

Die Gleichung dazu lautet „Geschwindigkeit = Beschleunigung * Zeit“. Die Beschleunigung ist wegen der Schwerkraft ein fester Wert, und ich will nun wissen, wie lang es dauert, bis ich eine Geschwindigkeit von 30 Metern pro Sekunde habe. Zugegeben, das Beispiel ist trivial, aber leicht verständlich.

Mit der Dokument-Methode seekGoal(CellAddress, CellAddress, String) starten Sie eine Zielwertsuche. Die erste Zelladresse kennzeichnet die Zelle, in der die zu lösende Formel steht. Die zweite Zelladresse kennzeichnet die Zelle mit der veränderlichen Variablen. Geben Sie den bestmöglichen Schätzwert in diese Zelle ein. Das letzte Stringargument enthält den Wert, den die Formel als Ergebnis haben soll. Das Makro im Listing 469 gibt die Formel ein und ruft eine Zielwertsuche auf.

Listing 469. *Eine einfache Zielwertsuche.*

```
Sub GoalSeekExample
  Dim oSheet   'Das erste Tabellenblatt
  Dim oGCell   'B24, Wert für die Schwerkraft: 9,81
  Dim oTCell   'C24, Zeitwert
  Dim oRCell   'Die Gleichung "=B24*C24"
  Dim oGoal    'Das Zielwertobjekt
  oSheet = ThisComponent.Sheets(0)
  oGCell = oSheet.getCellByPosition(1, 23)
  oGCell.setValue(9.81)

  oTCell = oSheet.getCellByPosition(2, 23)
  oTCell.setValue(1)

  oRCell = oSheet.getCellByPosition(0, 23)
  oRCell.setFormula("=B24 * C24")

  oGoal = ThisComponent.seekGoal(oRCell.CellAddress, oTCell.CellAddress, "30")
  MsgBox "Ergebnis = " & oGoal.Result & " Sekunden" & Chr$(10) &_
         "Das Ergebnis hat sich um " & oGoal.Divergence &_
         " in der letzten Iteration verändert.", 0, "Zielwertsuche"
End Sub
```

Die Methode seekGoal() gibt ein Struct mit zwei Fließkommazahlen vom Typ Double zurück. Die Eigenschaft Result enthält den Lösungsvorschlag. Die Eigenschaft Divergence enthält die Differenz zwischen dem letzten Näherungsschritt und dem ausgewiesenen Ergebnis. Ist die Differenz klein, ist das Ergebnis wahrscheinlich hinreichend korrekt. Ist jedoch die Differenz groß, ist das Ergebnis wahrscheinlich nicht genau genug. Ich habe einmal mit einem Test gar kein Ergebnis erzielt. Die Differenz betrug grob 1,0E308.

Lassen Sie sich nicht durch die einfache Formel des Beispiels täuschen. Ich habe die Zielwertsuche als Lösung für eine Aufgabe gewählt, die in keine feste Formel gefasst werden konnte – ich hätte zur Lösung einen numerischen Algorithmus verwenden müssen.

15.7. Eigene Tabellenfunktionen schreiben

Mit OOo ist es trivial, eigene Funktionen zu schreiben und anzuwenden, die vom Calc-Dokument erkannt werden. Es ist ebenso einfach, wie ein Makro zu schreiben und direkt aufzurufen. Als Beispiel diene das Listing 470, das Informationen über das übergebene Argument als String zurückgibt. Tabelle 225 zeigt die Rückgabewerte für verschiedene Argumente.

Listing 470. *Eigene Tabellenfunktion, die eine Information über das Argument ausgibt.*

```
Function WahooFunc(Optional x) As Variant
  Dim s$
  s = "Ich bin in WahooFunc. "
```

```
  If IsMissing(x) Then
    s = s & "Es wurde kein Argument angegeben."
  ElseIf Not IsArray(x) Then
    s = s & "Das skalare Argument (" & CStr(x) & ") ist vom Typ " & TypeName(x)
  Else
    s = s & "Das Argument ist ein Array (" & LBound(x, 1) & " To " & UBound(x, 1) &_
            ", " & LBound(x, 2) & " To " & UBound(x, 2) & ")"
  End If
  WahooFunc = s
End Function
```

Tabelle 225. *Rückgabe der Funktion WahooFunc für verschiedene Argumente (E9 = 2).*

Funktion	Rückgabe
„=WahooFunc()“	Ich bin in WahooFunc. Es wurde kein Argument angegeben.
„=WahooFunc(E9)“	Ich bin in WahooFunc. Das skalare Argument (2) ist vom Typ Double.
„=WahooFunc(2)“	Ich bin in WahooFunc. Das skalare Argument (2) ist vom Typ Double.
„=WahooFunc(A11:C15)“	Ich bin in WahooFunc. Das Argument ist ein Array (1 To 5, 1 To 3).

Das der Funktion übergebene Argument kann fehlen. Wenn das Argument als optional deklariert ist, testen Sie mit der Funktion IsMissing(), ob ein Wert übergeben wurde oder nicht, zum Beispiel bei „=WahooFunc()“.

Sie können einen einfachen skalaren Wert entweder als Konstante oder als Referenz auf eine einzelne Zelle übergeben, zum Beispiel „=WahooFunc(32)“ und „=WahooFunc(E7)“. Beide übergeben der Funktion einen skalaren Wert, das erste Beispiel als numerische Konstante 32 und das zweite Beispiel als Inhalt der Zelle E7.

Tipp Der tatsächliche Typ des ihrer Funktion übergebenen Arguments hängt davon ab, wie sie aufgerufen wird (s. Tabelle 225). Wenn das Argument ein Zellbereich ist, werden die Daten als zweidimensionales Array übergeben, das nicht wie ansonsten gewohnt nullbasiert ist.

Wenn das Argument auf einen Zellbereich weist, wird der Funktion ein zweidimensionales Array übergeben. Zum Beispiel übergibt „=WahooFunc(E7:F32)“ die Inhalte der Zellen im Bereich E7 bis F32 als zweidimensionales Array. Achten Sie unbedingt darauf, die Funktionen LBound() und UBound() zu verwenden, denn die Untergrenze liegt bei 1 statt bei dem eher erwarteten Wert 0 (s. Listing 471).

Listing 471. *Eigene Tabellenfunktion, die die Summe aller Elemente ausgibt.*

```
Function SumAll(myArray As Variant)
  Dim iRow%, iCol%  'Zeilenindex, Spaltenindex
  Dim d As Double
  For iRow = LBound(myArray, 1) To UBound(myArray, 1)
    For iCol = LBound(myArray, 2) To UBound(myArray, 2)
      d = d + myArray(iRow, iCol)
    Next
  Next
  SumAll = d
End Function
```

Unterschiedliche OOo-Versionen verhalten sich unterschiedlich, was eigene Calc-Funktionen betrifft. Die Calc-Funktion muss im Calc-Dokument sichtbar und erreichbar sein. Also speichere ich meine Calc-Funktionen in der Standardbibliothek des Dokuments, in dem die Funktionen verwendet werden sollen. Wenn ich eine neue Funktion schreibe, kann es manchmal vorkommen, dass sie erst erkannt wird, wenn das Dokument gespeichert, geschlossen und wieder geöffnet wird.

Tipp Falls Calc eine neu erstellte Funktion nicht erkennt, dann speichern Sie das Dokument, schließen es und öffnen es erneut.

Achtung Geben Sie in einer Calc-Funktion keinen Uhrzeit- oder Datumswert zurück. Geben Sie stattdessen einen Double-Wert zurück.

Die Rückgabewerte von als Calc-Funktionen genutzten Makros werden zu Double oder zu String konvertiert, bevor sie in einer Zelle gespeichert werden. Der Datumstyp wird zu einem String konvertiert, der sich natürlich nicht für Datumsberechnungen eignet. Richtig ist es, ein Datum oder eine Uhrzeit zu einem Double-Wert zu konvertieren und diesen dann zurückzugeben. Calc wird den Wert gemäß dem Zellformat als Datum oder Uhrzeit behandeln.

15.8. Der aktuelle Controller

Jedes OOo-Dokument hat einen Controller, der die Interaktion mit dem Benutzer regelt. Daher weiß der aktuelle Controller, welcher Text ausgewählt ist, er kennt die Position des aktuellen Cursors und das momentan aktive Tabellenblatt.

15.8.1. Ausgewählte Zellen

Der Controller des Dokuments interagiert mit dem Benutzer und weiß daher, welche Zellen ausgewählt sind. In einem Calc-Dokument kann eine Zelle auf unterschiedliche Art ausgewählt sein. Die Nummerierung der folgenden Fälle dient nur dem Zweck des praktischen Verweises, sie stellt keine Rangfolge dar.

1. Einzelne ausgewählte Zelle. Um eine Zelle komplett auszuwählen, einfachklicken Sie in die Zelle und klicken noch einmal mit gedrückter Umschalttaste.
2. Partielle Textauswahl in einer einzelnen Zelle. Doppelklicken Sie in eine einzelne Zelle und selektieren einen Textanteil.
3. Keine Auswahl erkennbar. Einfachklicken Sie in eine Zelle oder wechseln Sie mit dem Tabulator zwischen den Zellen.
4. Mehrere Zellen ausgewählt. Einfachklicken Sie in eine Zelle und ziehen Sie dann den Cursor weiter.
5. Mehrere unverbundene Zellbereiche ausgewählt. Wählen Sie eine Zellgruppe aus. Halten Sie dann die Strg-Taste gedrückt und wählen weitere aus.

Bisher ist es mir noch nicht gelungen, zwischen den ersten drei Fällen zu unterscheiden: es ist einfach immer eine Zelle ausgewählt. Wenn nur eine Zelle ausgewählt ist, gibt der aktuelle Controller als aktuelle Auswahl die Zelle des Tabellenblatts zurück, in der sich der Cursor befindet. In diesem Fall (das heißt den Fällen 1 bis 3) unterstützt das Auswahl-Objekt den Service SheetCell (s. Listing 472).

Wenn mehrere Zellen als einzelner Bereich ausgewählt sind (Fall 4), ist das Auswahlobjekt ein SheetCellRange. Prüfen Sie, ob das Auswahlobjekt den Service SheetCellRanges unterstützt. Wenn ja, dann sind mehrere Zellbereiche ausgewählt. Mit der Methode getCount() des Auswahlobjekts finden Sie heraus, wie viele Bereiche ausgewählt sind.

Listing 472. *Test, ob irgendetwas ausgewählt ist.*

```
Function CalcIsAnythingSelected(oDoc) As Boolean
  Dim oSelections    'Die aktuelle Auswahl

  REM Grundannahme: nichts ist ausgewählt.
```

```
  CalcIsAnythingSelected = False
  If IsNull(oDoc) Then Exit Function
  REM Die aktuelle Auswahl im aktuellen Controller.
  REM Falls es keinen aktuellen Controller gibt, wird NULL zurückgegeben.
  oSelections = oDoc.getCurrentSelection()
  If IsNull(oSelections) Then Exit Function

  If oSelections.supportsService("com.sun.star.sheet.SheetCell") Then
    Print "Einzelzelle ausgewählt = " & oSelections.getImplementationName()
    Print "getString() = " & oSelections.getString()
  ElseIf oSelections.supportsService("com.sun.star.sheet.SheetCellRange") Then
    Print "Einzelner Zellbereich ausgewählt = " & oSelections.getImplementationName()
  ElseIf oSelections.supportsService("com.sun.star.sheet.SheetCellRanges") Then
    Print "Mehrere Zellbereiche ausgewählt = " & oSelections.getImplementationName()
    Print "Anzahl = " & oSelections.getCount()
  Else
    Print "Etwas anderes ausgewählt = " & oSelections.getImplementationName()
  End If
  CalcIsAnythingSelected = True
End Function
```

Enumeration der ausgewählten Zellen

Listing 473 ist eine Hilfsroutine, die den Text der Zellen eines Bereichs mit einem bestimmten Wert überschreibt. Obwohl Listing 473 speziell auf einen Zellbereich konzipiert ist, werden Methoden verwendet, die auch von einer Einzelzelle unterstützt werden. Das Makro kann daher auf eine Zelle oder einen Zellbereich angewendet werden. Die im Listing 473 verwendete Methode ist eine Modifizierung eines Algorithmus, den ich von Sasa Kelecevic aus der OOo-dev-Mailingliste habe.

Listing 473. *Ersetzt den Text aller Zellen eines Bereichs.*

```
Sub SetRangeText(oRange, s As String)
  Dim nCol As Long 'Spaltenindex
  Dim nRow As Long 'Zeilenindex
  Dim oCols        'Spalten im ausgewählten Bereich
  Dim oRows        'Zeilen im ausgewählten Bereich
  oCols = oRange.Columns : oRows = oRange.Rows
  For nCol = 0 To oCols.getCount() - 1
    For nRow = 0 To oRows.getCount() - 1
      oRange.getCellByPosition(nCol, nRow).setString(s)
    Next
  Next
End Sub
```

Zur Demonstration des Zugriffs auf alle ausgewählten Zellen überschreibt das Makro im Listing 474 den Text jeder ausgewählten Zelle mit einem bestimmten Wert.

Listing 474. *Überschreibt alle ausgewählten Zellen mit einem Text.*

```
Sub SetSelectedCells(s As String)
  Dim oSelections 'Das Auswahlobjekt
  Dim oCell       'Falls eine Einzelzelle ausgewählt ist
  Dim oRanges     'Falls mehrere unverbundene Zellbereiche ausgewählt sind
  Dim i As Long   'Indexvariable

  REM Die aktuelle Auswahl im aktuellen Controller.
  REM Falls es keinen aktuellen Controller gibt, wird NULL zurückgegeben.
  oSelections = ThisComponent.getCurrentSelection()
  If IsNull(oSelections) Then Exit Sub
```

```
  If oSelections.supportsService("com.sun.star.sheet.SheetCell") Then
    oCell = oSelections
    oCell.setString(s)

    REM Ich könnte hier eine Zelle wie einen Zellbereich behandeln,
    REM aber nur, weil eine Zelle auch Spalte und Zeile zurückgeben kann.
    REM SetRangeText(oSelections, s)
  ElseIf oSelections.supportsService("com.sun.star.sheet.SheetCellRange") Then
    SetRangeText(oSelections, s)
  ElseIf oSelections.supportsService("com.sun.star.sheet.SheetCellRanges") Then
    oRanges = oSelections
    For i = 0 To oRanges.getCount() - 1
      SetRangeText(oRanges.getByIndex(i), s)
    Next
  Else
    Print "Etwas anderes ausgewählt = " & oSelections.getImplementationName()
  End If
End Sub
```

Als ich das Makro im Listing 474 schrieb, stieß ich auf ein anfangs verblüffendes Verhalten. Meine ursprüngliche Version erzeugte manchmal beim Eintritt in die Subroutine einen Laufzeitfehler, und zwar abhängig von den ausgewählten Werten. Der Laufzeitfehler beklagte, dass eine nicht unterstützte Eigenschaft oder ein ungültiger Wert benutzt werde. Zur Erklärung (und Lösung) des Problems weise ich Sie auf zwei Stellen im Code hin, an denen ich oSelections einer temporären Variablen zuweise und danach diese Variable verwende. Listing 475 zeigt die betreffenden Codeausschnitte aus dem Listing 474.

Listing 475. *Ich weise oSelections einer temporären Variablen zu und verwende diese.*

```
oCell = oSelections
oCell.setString(s)
..
oRanges = oSelections
For i = 0 To oRanges.getCount() - 1
  SetRangeText(oRanges.getByIndex(i), s)
Next
```

Wenn sich oSelections auf ein Zellobjekt bezieht, wird die Objektmethode getCount() nicht unterstützt. Wenn sich oSelections auf ein SheetCellRanges-Objekt bezieht, wird die Objektmethode setString() nicht unterstützt. Ich kann nur annehmen, dass der Basic-Interpreter versucht, die Eigenschaften und Methoden aufzulösen, die ein Objekt referenziert, wenn es einen Wert erhält. Wenn zum Beispiel das Auswahlobjekt eine Zelle ist, wird ein Laufzeitfehler ausgelöst, weil eine Zelle die Methode getCount() nicht unterstützt. Wenn andererseits mehr als eine Zelle ausgewählt ist, unterstützt das Auswahlobjekt nicht die Methode setString(). Obwohl ich dieses Problem nicht zum ersten Mal erlebt hatte, war es doch das erste Mal, dass ich die Ursache des Problems und eine Lösung finden konnte.

Text auswählen

Der aktuelle Controller wird zur Ermittlung der aktuellen Auswahl benötigt, über ihn kann aber auch die aktuelle Auswahl vorgenommen werden. Mit der Methode select(obj) des aktuellen Controllers werden Zellen in einem Tabellenblatt ausgewählt. Die Dokumentation sagt im Wesentlichen, dass der Controller das als Argument übergebene Objekt auswählt, wenn er es erkennt und auswählen kann. Listing 476 zeigt, wie man Zellen auswählt.

Listing 476. *Wählt erst B28:D33 und danach stattdessen die Zelle A1.*

```
Dim oSheet, oRange, oCell, oController
oController = ThisComponent.getCurrentController()
oSheet = ThisComponent.Sheets(3)
oRange = oSheet.getCellRangeByName("B28:D33")
oController.select(oRange)
oCell = oSheet.getCellByPosition(0, 0)
oController.select(oCell)
```

Die aktive Zelle

Die aktive Zelle hat den Cursor. Es gibt eine aktive Zelle, auch wenn mehrere Zellen ausgewählt sind. Leider bietet OOo keine Methode, die aktive Zelle zurückzugeben, wenn mehr als eine Zelle ausgewählt ist. Paolo Mantovani hat eine hübsche Lösung für dieses Problem in der Dev-Mailingliste gepostet, s. Listing 477. Der Nachteil des Makros im Listing 477 besteht allerdings darin, dass die aktive Zelle nach Ablauf des Makros nicht mehr aktiv ist.

Listing 477. *Ermittlung der aktiven Zelle.*

```
REM Autor: Paolo Mantovani
REM E-Mail: mantovani.paolo@tin.it
Sub RetrieveTheActiveCell()
  Dim oOldSelection 'Die ursprüngliche Auswahl von Zellbereichen
  Dim oRanges       'Ein vom Dokument erzeugter leerer Zellbereich
  Dim oActiveCell   'Die aktuell aktive Zelle
  Dim oConv         'Der Service zur Zelladresskonversion
  Dim oDoc
  oDoc = ThisComponent

  REM Sicherung der aktuellen Auswahl.
  oOldSelection = oDoc.CurrentSelection

  REM Erzeugt einen leeren Service SheetCellRanges und wählt ihn aus.
  REM Dadurch bleibt NUR die aktive Zelle ausgewählt.
  oRanges = oDoc.createInstance("com.sun.star.sheet.SheetCellRanges")
  oDoc.CurrentController.Select(oRanges)

  REM Zugriff auf die aktive Zelle.
  oActiveCell = oDoc.CurrentSelection

  oConv = oDoc.createInstance("com.sun.star.table.CellAddressConversion")
  oConv.Address = oActiveCell.getCellAddress
  Print oConv.UserInterfaceRepresentation
  Print oConv.PersistentRepresentation

  REM Wiederherstellung der alten Auswahl, aber Verlust der vorher aktiven Zelle.
  oDoc.CurrentController.Select(oOldSelection)
End Sub
```

Es gibt noch eine andere Lösung – ohne Verlust der aktiven Zelle (s. Listing 478). Sie geht auf eine Idee von Laurent Godard zurück, der herausfand, dass die Information über die aktiven Zellen aller Tabellenblätter in dem String stecken, der von der Controller-Methode getViewData() zurückgegeben wird. Dieser String beschreibt die augenblickliche Ansicht (View). Wenn man ihn sichert, kann man diese Ansicht später mit der Controller-Methode restoreViewData(data) wiederherstellen. Außerdem wird der String beim Speichern einer Calc-Datei erzeugt und in die interne Datei settings.xml eingefügt. Zu den Einzelheiten s. die Diskussion vom Januar 2006:
http://openoffice.2283327.n4.nabble.com/api-dev-ViewData-string-meaning-td2765848.html.

Leider ist der String nicht dokumentiert und daher nach Aussage der Entwickler nicht vor Änderungen der Struktur geschützt. Da er sich jedoch bis heute (AOO 4.1.2 bzw. LO 5.0.1.2) in unveränderter Form gehalten hat, kann man es je nach Makroverwendung vielleicht wagen, sich auch für die Zukunft darauf zu verlassen. Gleichwohl bleibt ein Restrisiko.

Listing 478. *Ermittlung der aktiven Zelle aus den ViewData.*

```
REM Autor: Michael Büssow
REM Nach einer Idee von Laurent Godard
Function ActiveCell()
  Dim oDoc                 'Das aktuelle Dokument
  Dim oCtrl                'Der aktuelle Controller
  Dim sViewData As String  'Die aktuelle Ansicht der gesamten Calc-Datei
  Dim sViewCells As String 'Die aktuelle Ansicht des aktiven Tabellenblatts
  Dim iActiveSheet As Long 'Zählung des aktiven Tabellenblatts
  Dim iActiveCol As Long   'Zählung der Spalte der aktiven Zelle
  Dim iActiveRow As Long   'Zählung der Zeile der aktiven Zelle

  oDoc = ThisComponent
  oCtrl = oDoc.CurrentController
  iActiveSheet = oCtrl.ActiveSheet.RangeAddress.Sheet
  sViewData = oCtrl.ViewData 'Basic-Kürzel für oCtrl.getViewData()

  REM Die Struktur des ViewData-Strings besteht aus durch Semikolon getrennten Gruppen.
  REM Die Tabellenblätter werden fortlaufend in jeweils eigenen Gruppen beschrieben,
  REM beginnend mit der vierten Gruppe (= Tabellenblatt 0).
  sViewCells = Split(sViewData, ";")(3 + iActiveSheet)

  REM Eine Tabellenblattgruppe besteht aus 11 Positionen, die durch Schrägstrich
  REM (bei mehr als 8192 Zeilen durch Pluszeichen) getrennt sind.
  sViewCells = Join(Split(sViewCells, "+"), "/") 'Austausch von + gegen /

  REM Die ersten beiden Positionen enthalten die Spalten- und Zeilenzählung
  REM der aktiven Zelle.
  iActiveCol = Split(sViewCells, "/")(0)
  iActiveRow = Split(sViewCells, "/")(1)

  ActiveCell = oDoc.Sheets(iActiveSheet).getCellByPosition(iActiveCol, iActiveRow)
End Function
```

15.8.2. Allgemeine Funktionalität

Wenn man nach Funktionalitäten für die Bildschirmausgabe sucht, beginnt man am besten mit dem aktuellen Controller. Tabelle 226 und Tabelle 227 listen die meisten der vom aktuellen Controller unterstützten Methoden und Eigenschaften auf, die noch nicht behandelt wurden.

Tabelle 226. *Vom aktuellen Controller unterstützte und noch nicht behandelte Methoden.*

Methode	Beschreibung
getActiveSheet()	Zugriff auf das aktive Tabellenblatt.
setActiveSheet(XSpreadsheet)	Aktiviert das spezifizierte Tabellenblatt.
getIsWindowSplit()	Gibt True zurück, wenn die Ansicht geteilt ist.
getSplitHorizontal()	Horizontale Teilungsposition (in Pixeln als Long Integer).
getSplitVertical()	Vertikale Teilungsposition (in Pixeln als Long Integer).
getSplitColumn()	Spalte, vor der die Ansicht geteilt ist (Long Integer).

Methode	Beschreibung
getSplitRow()	Zeile, vor der die Ansicht geteilt ist (Long Integer).
splitAtPosition(x, y)	Teilt die Ansicht an der spezifizierten Position. Wenn x=0 ist, wird nur horizontal geteilt. Wenn y=0 ist, wird nur vertikal geteilt.
hasFrozenPanes()	True, wenn die Ansicht fixierte Ausschnitte hat.
freezeAtPosition(nSpalte, nZeile)	Fixiert Ausschnitte mit der angegebenen Anzahl an Spalten und Zeilen.
getFirstVisibleColumn()	Gibt die erste sichtbare Spalte im sichtbaren Ausschnitt als Long Integer zurück.
setFirstVisibleColumn(Long)	Legt die erste sichtbare Spalte im sichtbaren Ausschnitt fest.
getFirstVisibleRow()	Gibt die erste sichtbare Zeile im sichtbaren Ausschnitt als Long Integer zurück.
setFirstVisibleRow(Long)	Legt die erste sichtbare Zeile im sichtbaren Ausschnitt fest.
getVisibleRange()	Gibt den Bereich des sichtbaren Ausschnitts als CellRangeAddress zurück.

Tabelle 227. *Vom aktuellen Controller unterstützte und noch nicht behandelte Eigenschaften.*

Eigenschaft	Beschreibung
ShowFormulas	Falls True, werden die Formeln statt ihrer Ergebnisse angezeigt.
ShowZeroValues	Falls True, werden Nullwerte angezeigt.
IsValueHighlightingEnabled	Falls True, werden Strings, numerische Werte und Formeln in unterschiedlichen Farben angezeigt.
ShowNotes	Falls True, wird eine Markierung für Zellkommentare angezeigt.
HasVerticalScrollBar	Falls True, gibt es in der Ansicht eine vertikale Bildlaufleiste.
HasHorizontalScrollBar	Falls True, gibt es in der Ansicht eine horizontale Bildlaufleiste.
HasSheetTabs	Falls True, gibt es in der Ansicht Tabellenregister.
IsOutlineSymbolsSet	Falls True, werden Gliederungssymbole angezeigt.
HasColumnRowHeaders	Falls True, sind Spalten- und Zeilenköpfe sichtbar.
ShowGrid	Falls True, werden Gitterlinien angezeigt.
GridColor	Farbe der Gitterlinien als Long Integer.
ShowHelpLines	Falls True, werden Hilfslinien während der Bewegung von Zeichnungsobjekten angezeigt.
ShowAnchor	Falls True, werden Ankersymbole bei der Auswahl von Zeichnungsobjekten angezeigt.
ShowPageBreaks	Falls True, werden Seitenumbrüche angezeigt.
SolidHandles	Falls True, werden bei der Auswahl von Zeichnungsobjekten die Griffe ohne 3D-Wirkung angezeigt.
ShowObjects	Falls True, werden eingebettete Objekte angezeigt.
ShowCharts	Falls True, werden Diagramme angezeigt.
ShowDrawing	Falls True, werden Zeichnungsobjekte angezeigt.
HideSpellMarks	Falls True, wird die Markierung falsch geschriebener Wörter nicht angezeigt. Diese Eigenschaft scheint nicht aktiviert zu sein.

Eigenschaft	Beschreibung
ZoomType	Zoomfaktor des Dokuments als Wert der Konstantengruppe com.sun.star.view.DocumentZoomType: • OPTIMAL = 0 – Die komplette Seitenbreite (ohne Ränder) wird angezeigt. • PAGE_WIDTH = 1 – Die Seitenbreite der aktuellen Auswahl wird angezeigt. • ENTIRE_PAGE = 2 – Eine komplette Seite wird angezeigt. • BY_VALUE = 3 – Der Zoom ist relativ und wird von ZoomValue festgelegt. • PAGE_WIDTH_EXACT = 4 – Die aktuelle Breite bis zum exakten Seitenende wird angezeigt.
ZoomValue	Prozentualer Zoomfaktor (Short Integer), falls ZoomType BY_VALUE ist.

15.9. Calc aus Microsoft Office steuern

Es zeigt sich, dass man OOo auch mitten aus der Microsoft-Office-Produktfamilie steuern kann. Der Trick besteht darin, einen Service-Manager zu erzeugen, der OOo startet, wenn es noch nicht läuft. Aus Microsoft Office auf OOo-Dokumente zuzugreifen gestaltet sich ähnlich wie der Zugriff mit anderen Sprachen als StarBasic. Allerdings bietet StarBasic angenehme Abkürzungen, die in Microsoft Office nicht zur Verfügung stehen. Wenn ich zum Beispiel in StarBasic das dritte Tabellenblatt haben will, schreibe ich einfach oDoc.Sheets(2). In Microsoft Office ist es aber nicht möglich, die Eigenschaft Sheets als Array anzusprechen. Das Makro im Listing 479 habe ich in Microsoft Excel aus Microsoft Visual Basic aufgerufen.

Listing 479. *ControlOOo() zeigt, wie man OOo aus Excel heraus steuert.*

```
Sub ControlOOo()
  REM Der Service-Manager ist immer als erstes zu erzeugen.
  REM Falls OOo nicht läuft, wird es gestartet.
  Set oManager = CreateObject("com.sun.star.ServiceManager")

  REM Erzeugt den Desktop.
  Set oDesktop = oManager.createInstance("com.sun.star.frame.Desktop")

  REM Öffnet ein neues, leeres Calc-Dokument.
  Dim args()
  Dim s As String
  Set s = "private:factory/scalc"
  Set oDoc = oDesktop.loadComponentFromURL(s, "_blank", 0, args())

  Dim oSheet As Object
  Dim oSheets As Object
  Dim oCell As Object

  Set oSheets = oDoc.Sheets.CreateEnumeration
  Set oSheet = oSheets.nextElement
  Set oCell = oSheet.getCellByPosition(0, 0)
  oCell.setFormula("Hallo aus Excel") 'Zelle A1
  oCell.CellBackColor = RGB(127, 127, 127)
End Sub
```

15.10. Zugriff auf Calc-Funktionen

Man kann aus einem Makro Calc-Funktionen aufrufen. Die Methode callFunction des Service FunctionAccess erwartet zwei Argumente. Das erste ist der Name der Funktion als String in Englisch. Die englischsprachigen Entsprechungen der Funktionen in der deutschen OO-Version finden Sie hier:

http://www.ooowiki.de/DeutschEnglischCalcFunktionen.html. Das zweite Argument besteht aus einem Array der Argumente der Calc-Funktion, auch wenn die Funktion nur ein einziges Argument erwartet.

Listing 480. *Direkter Aufruf der Funktion MIN.*

```
Sub CallFunction
  Dim oFA
  oFA = CreateUnoService("com.sun.star.sheet.FunctionAccess")

  ' Berechnet die kleinste Zahl einer Reihe von Argumenten.
  Print oFA.callFunction("MIN", Array(10, 23, 5, 345))
End Sub
```

15.11. URLs in Calc-Zellen

In einem Calc-Dokument werden Verknüpfungen (URLs) in Textfeldern gespeichert. Das folgende Makro enumeriert den Textinhalt der Zelle A1 des ersten Tabellenblatts nach URLs. Beachten Sie, dass sich die Textrepräsentation vom URL unterscheiden kann.

Listing 481. *Suche nach URLs in einer Zelle.*

```
Sub FindHyperLinkInCell
  Dim oCell          'Die zu durchsuchende Zelle
  Dim oParEnum       'Enumeration des Zelltextinhalts
  Dim oParElement    'Ein Element des Textinhalts
  Dim oEnum          'Enumeration des Inhalts eines Elements
  Dim oElement       'Ein Element der inneren Enumeration

  oCell = ThisComponent.Sheets(0).getCellByPosition(0, 0)
  REM Alle Bestandteile des Zelltexts:
  oParEnum = oCell.getText().createEnumeration()
  Do While oParEnum.hasMoreElements()
    oParElement = oParEnum.nextElement()
    REM Alle Bestandteile eines einzelnen Textinhaltteils:
    oEnum = oParElement.createEnumeration()
    Do While oEnum.hasMoreElements()
      oElement = oEnum.nextElement()
      REM Uns interessieren nur Textfelder.
      If oElement.TextPortionType = "TextField" Then
        If oElement.TextField.supportsService("com.sun.star.text.TextField.URL") Then
          'STRING   Representation =
          'STRING   TargetFrame =
          'STRING   URL =
          'INTEGER  Format =
          Print oElement.TextField.URL
        End If
      End If
    Loop
  Loop
End Sub
```

Allerdings ist der Zugriff auf Text Portions nur lesend möglich. Wenn Sie also Hyperlinks in Zellen ändern wollen, müssen Sie die Textfelder der Zelle direkt referenzieren, entweder über den Indexwert:

```
oFields = oCell.getTextFields()
For i = 0 To oFields.getCount() - 1
  oField = oFields.getByIndex(i)
```

oder über die Enumeration:

```
oFieldsEnum = oCell.getTextFields().createEnumeration()
Do While oFieldsEnum.hasMoreElements()
  oField = oFieldsEnum.nextElement()
```

Das zurückgegebene Feld-Objekt unterstützt jedoch nicht den Service „com.sun.star.text.TextField.URL“. Es ist praktisch das Model-Objekt mit nur drei der im URL-Service definierten Eigenschaften: „Representation“, „TargetFrame“ und „URL“. Die vierte Eigenschaft „Format“ ist nur für das Ansichts(View)-Objekt von Interesse, zum Beispiel das TextPortion-Objekt, das sich die drei anderen Eigenschaften vom Model holt.

Das folgende Listing 482 zeigt, wie in der Zelle A1 ein im Text vorhandener Hyperlink geändert wird. Natürlich können Sie diese Technik auch für den lesenden Zugriff verwenden.

Listing 482. *Änderung eines URL in einer Zelle.*

```
Sub WriteHyperLinkInCell
  Dim oCell      'Die Zelle mit dem Hyperlink
  Dim oFields    'Alle Textfelder
  Dim oField     'Ein Textfeld
  Dim i%         'Zähler

  oCell = ThisComponent.Sheets(0).getCellByPosition(0, 0)
  oFields = oCell.getTextFields()
  For i = 0 To oFields.getCount() - 1
    oField = oFields.getByIndex(i)
    If oField.getPropertySetInfo().hasPropertyByName("Representation") Then
      oField.Representation = "Neue Datei"
      oField.URL = "file:///home/volker/neuedatei.pdf"
      Exit For
    End If
  Next
End Sub
```

Das Einfügen eines Hyperlinks in eine Zelle ist erheblich simpler. Das folgende Listing 483 ersetzt den Inhalt der Zelle A1 durch einen URL.

Listing 483. *Einfügen eines URL in eine Zelle.*

```
Sub InsertHyperlinkIntoCell
  Dim oCell      'Die Zelle mit dem Hyperlink
  Dim oFields    'Alle Textfelder
  Dim oField     'Ein Textfeld

  oCell = ThisComponent.Sheets(0).getCellByPosition(0, 0)
  oField = ThisComponent.createInstance("com.sun.star.text.TextField.URL")
  oField.Representation = "Datei"
  oField.URL = ConvertToURL("/home/volker/datei.pdf")
  oText = oCell.getText()
  'Das Einfügen von Textfeldern in Zellen geht wie im Writer vonstatten.
  oText.insertTextContent(oText.createTextCursor(), oField, True)
End Sub
```

15.12. Import und Export von XML-Dateien in Calc (V. Lenhardt)

Dieser Abschnitt behandelt einfache Methoden zum Import und Export von XML-Dateien. Je nach Bedarf gibt es mehrere Wege.

XML ist ein Standard für den Austausch von Datenbankinhalten. Es ist kein Standard für Bezeichnungen oder für Datensatzstrukturen. Es ist eine formale Beschreibung von Kennzeichnern (Tags)

und der Kombination von Tags zu einer hierarchischen Datenstruktur. Man kann die Namen der Tags und die Namen der optionalen Attribute innerhalb gewisser Grenzen frei bestimmen. Sie kennen Tags von HTML, das man gewissermaßen als einen Spezial-XML-Standard betrachten kann. Versuchen Sie aber nicht, HTML-Dateien mit den Mitteln des XML-Imports zu importieren, denn in HTML sind größere Freiheiten als in XML erlaubt. Daher würden Sie in eine Flut von Fehlermeldungen laufen. Wenn Sie mehr über XML erfahren wollen, beginnen Sie mit Wikipedia. Für deutsche Leser gibt es den XML-Bereich des großartigen SELFHTML: http://de.selfhtml.org/xml/index.htm. Es ist schade, dass das Projekt „SELFHTML in English“ zur Zeit suspendiert ist. Allerdings existiert eine französische Fassung: http://fr.selfhtml.org/xml/index.htm.

XML-Dateien sind einfache Textdateien. Wenn Sie regelmäßig den XML-Report einer Datenbank nach Calc oder Base konvertieren müssen, sollten Sie an einen Filter mit XSLT als Konvertierungswerkzeug denken. Ich werde dieses Thema nicht weiter verfolgen, da es über Basic hinausgeht.

Gelegentlich werden Sie aber aus XML-Daten ein Tabellenblatt erstellen wollen. Dabei benötigen Sie nur einen Teil der Daten, und Sie brauchen auch keine Kopie der Datenstruktur. In diesem Abschnitt werden Sie erfahren, wie man das macht.

Nahezu alle modernen Programmiersprachen haben zwei Werkzeuge zum Import von XML-Daten eingebunden: SAX und DOM. In Basic gibt es beide nicht, aber die OOo-UNO-API bietet die notwendige Funktionalität, die somit auch mit Basic zur Verfügung steht. Der Hauptunterschied zwischen diesen beiden Werkzeugen liegt in der Art und Weise, wie die Daten aus der XML-Datei gelesen werden. SAX liest die Daten kontinuierlich, und der Basic-Code muss über Listeners Schritt halten und die Datenteile verwenden, während sie hereinkommen. Das ist verhältnismäßig langsam, kann aber manchmal der einzige Weg sein, bei begrenztem Arbeitsspeicher eine große Datei einzulesen. DOM andererseits liest alle Daten auf einmal in den Arbeitsspeicher, in ein hierarchisch strukturiertes Objekt. Das geht sehr schnell, und der Basic-Code hat direkten Zugriff auf jedes einzelne Element. Man kann SAX im Vergleich mit DOM als Kassettenrekorder gegenüber einer CD betrachten.

Dieser Abschnitt wird sich nur mit DOM beschäftigen. Dabei werde ich es so einfach wie möglich halten und alle Fragen der Validierung, der Namensräume, der Verarbeitungsanweisungen und der Fehlerbehandlung ausschließen.

Achtung Es ist sehr wichtig, dass Sie die Datenstruktur der zu importierenden XML-Datei kennen. Falls nicht, gibt es Wege, es zu erfahren: suchen Sie nach einer DTD-Datei (Document Type Definition) oder durchforschen Sie die Daten oder fragen Sie am besten den Datenbankadministrator.

15.12.1. Import einer XML-Datei

Wenn Sie XML-Dateien einlesen, müssen Sie berücksichtigen, dass Sie es mit einfachen Textdateien zu tun haben, also müssen Sie als erstes den Service com.sun.star.ucb.SimpleFileAccess und einen Input-Strom erzeugen. Als nächster Schritt folgt die Erzeugung des Service com.sun.star.xml.dom.DocumentBuilder, der das Interface com.sun.star.xml.dom.XDocumentBuilder einbindet. Mit dessen Methode parse(oInputStream) lesen Sie die Daten ein. Aber es geht auch viel einfacher, wenn die Daten als Datei auf dem Datenträger gespeichert sind: mit der Methode parseURI(URL). Beide Methoden geben den kompletten Datenbaum als DOM-Dokument-Objekt zurück (s. Listing 484).

Listing 484. *Demonstration des Imports von XML-Daten.*

```
REM Baut aus einer XML-Datei einen DOM-Dokument-Baum auf.
Function BuildDOMDoc(sUrl As String)
  Dim oDocBuilder
  Dim oDOM
  BuildDOMDoc = oDOM  'oDOM ist zu diesem Zeitpunkt Empty.
  On Error Goto Catch

  oDocBuilder = CreateUnoService("com.sun.star.xml.dom.DocumentBuilder")
```

```
    REM oDocBuilder liest die komplette XML-Datei ein und gibt ein Objekt zurück,
    REM das eine Baumstruktur aller "Knoten" (Nodes) enthält.
    oDOM = oDocBuilder.parseURI(sURL)

    REM Es gab keine Fehler.
    oDOM.normalize()
    BuildDOMDoc = oDOM
    Exit Function

    Catch:
    MsgBox "Fehler beim Import der XML-Daten " & Chr$(10) & Chr$(10) _
      & Error$ & " " & Chr$(10) & " ", 16, "XML-Import"
End Function
```

Stellen Sie sich vor, Ihre Firma bietet einen besonderen Dokumentenservice an, den jeder Kunde, allein oder mit anderen, eine frei wählbare Zeitlang unter einem frei wählbaren Titel nutzen kann. (Dieses Beispiel ist aus dem wirklichen Leben). Eine Übersicht über alle aktuellen Service-Datensätze können Sie als XML-Datenbankreport anfordern. Jeder einzelne Service wird als Slot bezeichnet, manche sind aktiv, andere sind reserviert, wieder andere sind noch frei. Die konkrete Ausgestaltung steckt im „document“, auf das wir nicht weiter eingehen müssen. Ein Auszug von zwei Datensätzen aus der XML-Datei könnte so aussehen:

Listing 485. *Beispiel einer XML-Datei.*

```
<slots>
  <slot status="active" ID="10317">
    <document ID="1234" title="Bumm bumm" />
    <validTo>12/03/2011</validTo>
    <customer name="Becker" gender="m" ID="11051" email="bb@tennis.de" />
  </slot>
  <slot status="free" ID="60072">
  </slot>
</slots>
```

Jedes XML-Element beginnt mit seinem Namen in Winkelklammern und endet mit einem Schrägstrich (/) vor seinem Namen in einem weiteren Satz von Winkelklammern. Optional kann ein Element mehrere benannte Attribute (innerhalb des Starttags) und mehrere Kindelemente oder ersatzweise einen einzigen Textinhalt (zwischen Start- und Endetag) erhalten. Wenn es weder Kindelemente noch einen Textinhalt gibt, kann man eine Kurznotation verwenden: man setzt an das Ende des Starttags einen Schrägstrich und lässt das Endetag ganz weg. Üblicherweise werden nur Elemente aufgeführt, die irgend einen Inhalt haben (Attribute, Textinhalt, Elemente). Formatierungen zur besseren Lesbarkeit wie Zeilenumbrüche, Leerzeichen und Tabulatoren – vernachlässigbare weiße Zeichen (ignorable whitespace) genannt – bilden keinen Teil der Daten, wenn sie sich zwischen verschiedenen Elementen befinden, werden aber als Textknoten importiert.

Ich habe die meisten Elemente und Attribute der Originalstruktur weggelassen, um das oben gezeigte Beispiel nicht zu verwirrend zu gestalten (und ja, ich habe die Daten geändert).

All diese Elemente, Attribute und Textinhalte werden im DOM-Baum als Knoten repräsentiert, einschließlich des DOM-Dokuments selbst als Wurzelknoten (root node). Die API-Dokumentation zu com.sun.star.xml.dom.XNode beschreibt diesen primären DOM-Datentyp wie folgt:

„Das Node-Interface ist der primäre Datentyp für das gesamte Dokument-Objekt-Modell. Es repräsentiert einen Einzelknoten in einem Dokument-Baum. Während alle Objekte, die das Node-Interface einbinden, Methoden zum Umgang mit Kindobjekten aufweisen, müssen aber nicht alle Objekte, die das Node-Interface einbinden, auch Kinder haben. Zum Beispiel dürfen Textknoten keine

Kindobjekte haben. Wenn man solchen Knoten Kindobjekte hinzufügt, wird ein DOMException-Laufzeitfehler erzeugt.

Die Eigenschaften ‚nodeName', ‚nodeValue' und ‚attributes' sind als Hilfsmechanismus vorgesehen, um an die Knoteninformationen zu kommen, ohne sich zu dem spezifischen abgeleiteten Interface hinabbegeben zu müssen. In den Fällen, in denen es offensichtlich keine Abbildungen für Eigenschaften eines bestimmten Knotentyps gibt (zum Beispiel nodeValue für ein Element oder attributes für einen Kommentar), wird Null zurückgegeben. Beachten Sie, dass die spezialisierten Interfaces weitere und passendere Mechanismen zum Lesen und Schreiben relevanter Informationen bereitstellen."

Tabelle 228 zeigt die von XNode abstammenden spezialisierten Knoten-Interfaces mit Typ, Namen und Wert. Der Wert des Knotentyps stammt aus der Enumeration com.sun.star.xml.dom.NodeType.

Tabelle 228. *Spezialisierte XML-Knoten.*

Interface	Knotentyp	Knotenname	Knotenwert
XAttr	ATTRIBUTE_NODE	Der Name des Attributs.	Der Wert des Attributs.
XCDATASection	CDATA_SECTION_NODE	"#cdata-section"	Der Inhalt der CDATA-Sektion.
XComment	COMMENT_NODE	"#comment"	Der Inhalt des Kommentars.
XDocument	DOCUMENT_NODE	"#document"	Null
XDocumentFragment	DOCUMENT_FRAGMENT_NODE	"#document-fragment"	Null
XDocumentType	DOCUMENT_TYPE_NODE	Der Name des Dokumenttyps.	Null
XElement	ELEMENT_NODE	Der Tagname.	Null
XEntity	ENTITY_NODE	Der Entity-Name	Null
XEntityReference	ENTITY_REFERENCE_NODE	Der Name der referenzierten Entity.	Null
XNotation	NOTATION_NODE	Der Name der Notation.	Null
XProcessingInstruction	PROCESSING_INSTRUCTION_NODE	Die Zielanwendung.	Der gesamte Inhalt außer dem Ziel.
XText	TEXT_NODE	"#text"	Der Inhalt des Textknotens.

Die von XNode eingesetzten Methoden finden Sie in der Tabelle 229.

Tabelle 229. *Methoden im Interface com.sun.star.xml.dom.XNode.*

Methode	Beschreibung (aus der API-Dokumentation übersetzt)
appendChild(newChild)	Fügt dem Knoten ein neues Kind am Ende der Liste der Kindknoten an.
cloneNode(deep As boolean)	Gibt ein Duplikat dieses Knotens zurück, d.h. es dient als allgemeiner Kopierkonstruktor für Knoten. Argument deep: True = Duplikat zusammen mit den Kindern. False = Duplikat ohne Kinder.
getAttributes()	Gibt ein Objekt NamedNodeMap zurück, das die Attribute dieses Knotens enthält, wenn es ein Element ist, ansonsten Null.
getChildNodes()	Gibt ein Objekt NodeList zurück, dass alle Kinder dieses Knotens enthält.
getFirstChild()	Gibt das erste Kind dieses Knotens zurück.
getLastChild()	Gibt das letzte Kind dieses Knotens zurück.
getLocalName()	Gibt den lokalen Teil des qualifizierten Namens dieses Knotens zurück.

Methode	Beschreibung (aus der API-Dokumentation übersetzt)
getNamespaceURI()	Gibt den URI des Namensraums dieses Knotens zurück, oder Null, falls nicht qualifiziert.
getNextSibling()	Gibt den Knoten zurück, der diesem Knoten unmittelbar folgt.
getNodeName()	Gibt den Namen dieses Knotens zurück, abhängig vom Typ, s. Tabelle 228.
getNodeType()	Gibt eine Kennung zurück, die den Typ des Objekts repräsentiert, s. Tabelle 228.
getNodeValue()	Gibt den Wert dieses Knotens zurück.
getOwnerDocument()	Gibt das Dokument-Objekt zurück, das mit diesem Knoten verbunden ist.
getParentNode()	Gibt das Elternteil dieses Knotens zurück.
getPrefix()	Gibt das Namensraum-Präfix dieses Knotens zurück, oder Null, falls nicht qualifiziert.
getPreviousSibling()	Gibt den Knoten zurück, der diesem Knoten unmittelbar vorangeht.
hasAttributes()	Gibt True zurück, wenn dieser Knoten Attribute besitzt (wenn er ein Element ist).
hasChildNodes()	Gibt True zurück, wenn dieser Knoten Kinder besitzt.
insertBefore(newChild, refChild)	Fügt dem Knoten ein neues Kind vor dem existierenden Kind refChild ein.
isSupported(feature, ver)	Prüft, ob die DOM-Variante ein besonderes Merkmal eingebunden hat und ob dieses Merkmal von diesem Knoten unterstützt wird.
normalize()	Versetzt alle Textknoten in der vollen Tiefe des unter diesem Knoten liegenden Baumes, inklusive Attributknoten, in eine „Normal“-Form, in der die Knoten nur durch ihre Struktur (d.h. Elemente, Kommentare, Verarbeitungsinstruktionen, CDATA-Sektionen und Entity-Referenzen) Textknoten separieren. Das heißt, es gibt keine benachbarten Textknoten oder leere Textknoten.
removeChild(oldChild)	Löscht den Kindknoten oldChild aus der Liste der Kinder und gibt ihn zurück.
replaceChild(newChild, oldChild)	Ersetzt den Kindknoten oldChild durch newChild in der Liste der Kinder und gibt den Knoten oldChild zurück.
setNodeValue(nodeValue)	Setzt den Wert dieses Knotens, abhängig vom Typ, s. Tabelle 228.
setPrefix(prefix)	Setzt das Namensraum-Präfix dieses Knotens, oder Null, falls nicht qualifiziert.

Immer wenn eine der von XNode oder davon abstammenden Interfaces eingesetzten Methoden eine Liste von Knoten zurückgibt, ist es normalerweise eine enumerierte Liste, die das Interface com.sun.star.xml.dom.XNodeList einsetzt. Dieses Interface unterstützt zwei Methoden: getLength() zur Bestimmung der Anzahl der aufgelisteten Knoten, und item(Long) für den indexierten Zugriff auf einen bestimmten Knoten.

Es gibt aber auch Listen mit Namenszugriff: Zum Beispiel gibt die Methode getAttributes() eine benannte Liste zurück, die das Interface com.sun.star.xml.dom.XNamedNodeMap einbindet, allerdings nur, wenn getAttributes() von einem Elementknoten aufgerufen wird, ansonsten wird Null zurückgegeben. Das Interface XNamedNodeMap unterstützt – zusätzlich zu XNodeList – unter anderem die Methode getNamedItem(name) zum Zugriff auf ein spezifisches Attribut unter seinem Namen. Ein anderer Weg zu einem bestimmten Attributknoten besteht im Aufruf der Methode getAttributeNode(name), das von XElement zur Verfügung gestellt wird (eine Liste aller XElement-Methoden finden Sie in der Tabelle 231). Das Interface XAttr liefert die Methode getValue() zum Zugriff auf den Wert des Attributs. Man kann einen Attributwert sogar auf noch kürzere Art ermitteln, mit der Elementmethode getAttribute(name). Zusammengefasst haben Sie folgende Möglichkeiten, an den Wert eines Attributs mit dem Namen „ID“ zu gelangen:

Listing 486. *Drei Wege des Zugriffs auf ein Attribut eines Elements.*

```
'Der Umweg.
oAttList = oElement.getAttributes()
oAttNode = oAttList.getNamedItem("ID")
sAttVal = oAttNode.getValue()
'Die Abkürzung.
oAttNode = oElement.getAttributeNode("ID")
sAttVal = oAttNode.getValue()
'Der direkte Weg.
sAttVal = oElement.getAttribute("ID")
```

Wenn das DOM-Dokument erzeugt ist, können Sie damit beginnen, den Inhalt rekursiv durch Iteration über die Elemente auszulesen:

```
oElemList = oElem.getChildNodes()       'Beim Start ist oElem identisch mit oDOM.
For i = 0 To oElemList.getLength() - 1
  oElem = oElemList.item(i)
  Select Case oElem.getNodeType()
    Case com.sun.star.xml.dom.ELEMENT_NODE
      sName = oElem.getNodeName()
      If oElem.hasAttributes Then
        oAtts = oElem.getAttributes()
        For j = 0 To oAtts.getLength() - 1
          sAttName = oAtts.item(j).getName()
          sAttValue = oAtts.item(j).getValue()
          'Hier folgen weitere Arbeiten.
        Next
      End If
      If oElem.hasChildNodes() Then
        'Hier beginnt die Rekursion.
      End If
    Case com.sun.star.xml.dom.TEXT_NODE
      sText = oElem.getNodeValue()
      'Hier folgen weitere Arbeiten.
    'Zu weiteren Case-Anweisungen für Knotentypen s. Tabelle 228.
  End Select
Next
```

Tipp Wenn Sie auf einen Textknoten zwischen unterschiedlichen Elementknoten stoßen, haben Sie sicher vernachlässigbaren Leerraum gefunden. Ignorieren Sie ihn.

So, nun steht die Aufgabe an, eine Serien-E-Mail an alle Kunden zu schreiben, deren Slot-Nutzung bald ausläuft, um sie zu fragen, ob sie den Service verlängern wollen. Alles was Sie brauchen, ist eine Liste der Kunden (Name, Geschlecht, E-Mail) mit ihren Slots (ID und Titel). Es sollen nur solche Slots aufgeführt sein, deren Status aktiv ist (status="active") und die vor einem bestimmten Datum auslaufen.

Sie brauchen gar nicht über alle Knoten zu iterieren. Sie wissen, was Sie brauchen, und Sie kennen die Struktur der Daten. Mit der Methode getElementsByTagName(name) starten Sie mit einer schon begrenzten Anzahl von Elementen. Das Listing 488 zeigt, wie man die passenden Daten extrahiert und in ein Calc-Tabellenblatt kopiert.

Achtung Vorsicht ist geboten beim Einsatz der Methode getElementsByTagName. Es gibt einen störenden Bug in OOo (ich weiß nicht, mit welcher Version er zuerst auftrat, jedenfalls ist er in den Versionen 3.1 bis 3.3.1 zu bemerken). Wenn der betreffende Tagname 8 oder 16 Zeichen lang ist, wird das Ergebnis unzuverlässig, und OOo wird unstabil und kann abstürzen. Der Fehler tritt nicht seit LibreOffice 3.4.2 auf (ältere Versionen sind nicht getestet).

Das Beispielmakro benötigt natürlich eine XML-Datei, aus der die Daten entnommen werden können. Mit dem folgenden Listing 487 wird eine solche Textdatei bereitgestellt, deren Daten zufällig gemischt sind, damit bei mehrmaligem Aufruf eine gewisse Variabilität entsteht.

Listing 487. *Textdatei mit XML-Inhalten.*

```
REM Erstellt eine XML-Datei im Nutzer-Arbeitsverzeichnis.
REM Aus den erfundenen Daten wird zufällig ausgewählt.
REM Gibt den URL der Datei zurück.
Function MakeXMLExampleFile() As String
  Dim aDocs               'Array der Dokumentelemente
  Dim aCusts              'Array der Kundenelemente
  Dim lDate As Date       'Datumswert für die Nutzungsdauer
  Dim oSimpleFileAccess
  Dim oOutputStream
  Dim oTextOutput
  Dim oPathSettings       'Die Pfadeinstellungen der Anwendung.
  Dim sUrl As String      'Der URL der neuen Datei.
  Dim i%, j%, id%

  oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")

  REM Der URL der XML-Datei im Nutzer-Arbeitsverzeichnis.
  sURL = oPathSettings.Work & "/oome_xml_import.xml"

  aDocs = Array("<document ID=""1234"" title=""Service"" />", _
                "<document ID=""2345"" title=""Return"" />", _
                "<document ID=""3456"" title=""Slice"" />", _
                "<document ID=""4567"" title=""Topspin"" />", _
                "<document ID=""5678"" title=""Lob"" />", _
                "<document ID=""6789"" title=""Drive"" />", _
                "<document ID=""7890"" title=""Game"" />", _
                "<document ID=""8901"" title=""Set"" />", _
                "<document ID=""9012"" title=""Match"" />", _
                "<document ID=""10345"" title=""Grand Slam"" />")
  aCusts = Array( _
    "<customer name=""Agassi"" gender=""m"" ID=""16243"" email=""aa@tennis.com"" />", _
    "<customer name=""Graf"" gender=""f"" ID=""62431"" email=""sg@tennis.com"" />", _
    "<customer name=""Becker"" gender=""m"" ID=""24316"" email=""bb@tennis.de"" />", _
    "<customer name=""Federer"" gender=""m"" ID=""43162"" email=""rf@tennis.ch"" />", _
    "<customer name=""Navrátilová"" gender=""f"" ID=""16243""" _
     & " email=""mn@tennis.com"" />")

  oSimpleFileAccess = CreateUnoService("com.sun.star.ucb.SimpleFileAccess")
  With oSimpleFileAccess
    If .exists(sUrl) Then .kill(sUrl)
    oOutputStream = .openFileWrite(sUrl)
  End With

  oTextOutput = CreateUnoService("com.sun.star.io.TextOutputStream")
  With oTextOutput
    .OutputStream = oOutputStream
    .setEncoding("UTF-8")
    'XML-Startzeile
    .WriteString("<?xml version=""1.0"" encoding=""UTF-8""?>" & Chr$(10))
    .WriteString("<slots>" & Chr$(10))
```

```
    Randomize
    Do
      id = id + 1
      'Zufällige Statusbestimmung: 20% free, 20% reserved, 60% active
      Select Case Int(Rnd * 5)
        Case 0
          .WriteString("  <slot status=""free"" ID=""" & id & """>" & Chr$(10))
        Case  1
          .WriteString("  <slot status=""reserved"" ID=""" & id & """>" & Chr$(10))
          'Zufällige Auswahl eines der 5 Kunden
          .WriteString("    " & aCusts(Int(Rnd * 5)) & Chr$(10))
        Case Else
          .WriteString("  <slot status=""active"" ID=""" & id & """>" & Chr$(10))
          .WriteString("    " & aDocs(i) & Chr$(10))
          'Zufällige Laufzeitbestimmung: 1/3 100 Tage, 2/3 10 Tage
          If Int(Rnd * 3) = 0 Then
            lDate = DateValue(CDate(Now + 100))
          Else
            lDate = DateValue(CDate(Now + 10))
          End If
          .WriteString("    <validTo>" & lDate & "</validTo>" & Chr$(10))
          'j = zufälliger Kunde (von 5)
          j = Int(Rnd * 5)
          .WriteString("    " & aCusts(j) & Chr$(10))
          'Fifty-fifty: 1 oder 2 Kunden für ein Dokument
          If Int(Rnd * 2) = 1 Then
            .WriteString("    " & aCusts((j + 1) Mod 5) & Chr$(10))
          End If
        i = i + 1
      End Select
      .WriteString("  </slot>" & Chr$(10))
    Loop Until i = 10
    .WriteString("</slots>" & Chr$(10))
    .closeOutput()
  End With
  oOutputStream.closeOutput()
  Wait 500               'Lässt dem OS Zeit, die Datei auf der Platte zu speichern.
  MakeXMLExampleFile = sUrl
End Function
```

Listing 488. *Analyse einer XML-Datei mit dem Ziel, ausgewählte Daten in einer Calc-Tabelle zu speichern.*

```
REM Importiert eine XML-Datei mit Hilfe von DOM
REM und extrahiert ausgewählte Daten in eine Calc-Tabelle.
Sub XMLImport
  Dim sURL As String     'Der vollständige Pfad der XML-Datei in URL-Notation.
  Dim oCalcDoc           'Das Ziel-Tabellendokument.
  Dim oSheet             'Die Zieltabelle.
  Dim oDOM               'Der DOM-Dokument-Baum.
  Dim tDeadline As Date  'Ein bestimmtes Datum zur Datenfilterung.
  Dim noArgs(0) As New com.sun.star.beans.PropertyValue

  REM Der URL der XML-Datei als Beispiel mit erfundenen Daten.
  sUrl = MakeXMLExampleFile
  tDeadline = DateValue(CDate(Now + 20))

  REM Aufbau des DOM-Baums.
  oDOM = BuildDOMDoc(sUrl)
```

```
  If IsEmpty(oDOM) Then
    MsgBox "Fehler beim Laden der XML-Datei", 16, "Ladefehler"
    Exit Sub
  End If

  REM Kopiert ausgewählte Daten in ein neues Calc-Dokument.
  oCalcDoc = StarDesktop.loadComponentFromURL( _
             "private:factory/scalc", "_blank", 0, Array())
  oSheet = oCalcDoc.Sheets(0)
  GetData(oDOM, oSheet, tDeadline)

  REM Präsentiert die originale XML-Textdatei in einem neuen Writer-Dokument.
  REM Für LO muss FilterName spezifiziert werden, für AOO nicht.
  noArgs(0).Name = "FilterName"
  noArgs(0).Value = "Text"
  StarDesktop.loadComponentFromURL(sUrl, "_blank", 0, noArgs())
End Sub

REM Liest die relevanten Daten aus dem DOM-Dokument
REM und schreibt sie zeilenweise in ein Calc-Dokument.
Sub GetData(oDOMDoc, oSheet, tDeadline As Date)
  Dim oValidToList        'Liste aller Elemente namens "validTo" im DOM-Dokument.
  Dim oSlotElement        'slot = Elternteil des validTo-Knotens.
  Dim sID As String       'Wert des Slot-Attributs "ID".
  Dim i%, j%
  Dim oDocs               'Liste der Elemente namens "document" in einem Slot.
  Dim sTitle As String    'Wert des Attributs "title" des document-Elements.
  Dim oCusts              'Liste der Elemente namens "customer" in einem Slot
  Dim iEntries As Integer 'Anzahl der im Tabellenblatt geschriebenen Zeilen.

  REM Ein Slot-Knoten enthält alle Informationen über die Slot-Dienstleistung.
  REM Auflistung aller Knoten namens "validTo", so dass alle Slots, in denen das
  REM Attribut "status" den Wert "reserved" oder "free" hat, ausgeschlossen sind.

  oValidToList = oDOMDoc.getElementsByTagName("validTo")

  For i = 0 To oValidToList.getLength() - 1
    oSlotElement = oValidToList.item(i).getParentNode()
    If IsProperSlot(oSlotElement) Then
      sID = oSlotElement.getAttribute("ID")
      REM Ich weiß, dass das validTo-Element genau einen Textknoten hat,
      REM der einen Datumsstring enthält.
      If IsWithinDateTarget _
        (oValidToList.item(i).getFirstChild().getNodeValue(), tDeadline) > 0 Then
        REM Liest den Wert des Attributs "title" eines document-Elements.
        sTitle = ""
        oDocs = oSlotElement.getElementsByTagName("document")
        If oDocs.getLength() = 1 Then  'Es kann höchstens einen einzigen geben.
          If oDocs.item(0).hasAttribute("title") Then
            sTitle = oDocs.item(0).getAttribute("title")
          End If
        End If
        REM Holt die Liste der customer-Elemente.
        oCusts = oSlotElement.getElementsByTagName("customer")
        iEntries = iEntries + oCusts.getLength()
        For j = 0 To oCusts.getLength() - 1
```

```
            REM Schreibt die Daten in die Calc-Tabelle.
            ImportSlot(oCusts.item(j), sID, sTitle, oSheet)
          Next
        End If
      End If
    Next
    SortCustomers(oSheet, iEntries)
End Sub

REM Prüft, ob der slot-Knoten die passenden Attributwerte hat.
Function IsProperSlot(oSlot) As Boolean
    IsProperSlot = False
    If oSlot.hasAttribute("status") And oSlot.hasAttribute("ID") Then
      If oSlot.getAttribute("status") = "active" Then
        IsProperSlot = True
      End If
    End If
End Function

REM Prüft, ob das Datum in validTo vor dem Fristtermin liegt.
Function IsWithinDateTarget(sDate As String, tDeadline As Date) As Integer
    Dim tValidTo As Date
    On Error Goto Catch

    REM Falls die Basic-Funktion DateValue den String nicht
    REM in einen Datumswert konvertieren kann, wird der Laufzeitfehler 5
    REM ausgelöst. In diesem Fall wird -1 zurückgegeben, damit die aufrufende
    REM Routine die Möglichkeit hat, fehlerhafte Datenbankeinträge zu verfolgen.
    tValidTo = DateValue(sDate)
    If tDeadline < tValidTo Then
      IsWithinDateTarget = 0
    Else
      IsWithinDateTarget = 1
    End If
    Exit Function

    Catch:
    IsWithinDateTarget = -1
End Function

REM Import der relevanten Slot-Daten in eine Calc-Tabelle.
Sub ImportSlot(oCust, sID As String, sTitle As String, oSheet)
    REM Es gibt eine Zeile für jedes Dokument eines jeden Kunden.
    REM Wenn sich also mehrere Kunden ein Dokument teilen, gibt es ebenso viele
    REM Zeilen, wie es Kunden an einem Dokument gibt.
    Static iRow As Integer       'Zeilenzählung

    If iRow = 0 Then             'Überschrift
      oSheet.getCellByPosition(0, iRow).String = "Name"
      oSheet.getCellByPosition(1, iRow).String = "E-Mail"
      oSheet.getCellByPosition(2, iRow).String = "Geschlecht"
      oSheet.getCellByPosition(3, iRow).String = "ID"
      oSheet.getCellByPosition(4, iRow).String = "Titel"
    End If
    iRow = iRow + 1
    If oCust.hasAttribute("name") Then
      oSheet.getCellByPosition(0, iRow).String = _
        oCust.getAttribute("name")
    End If
    If oCust.hasAttribute("email") Then
```

```
      oSheet.getCellByPosition(1, iRow).String = _
        oCust.getAttribute("email")
    End If
    If oCust.hasAttribute("gender") Then
      oSheet.getCellByPosition(2, iRow).String = _
        oCust.getAttribute("gender")
    End If
    oSheet.getCellByPosition(3, iRow).String = sID
    oSheet.getCellByPosition(4, iRow).String = sTitle
  End Sub

  REM Sortierung des Tabellenblatts zur besseren Lesbarkeit.
  Sub SortCustomers(oSheet, iRows As Integer)
    Dim oRange
    Dim aSortFields(2) As New com.sun.star.util.SortField
    Dim aSortDesc(1) As New com.sun.star.beans.PropertyValue

    REM Sortieroptionen:
    REM (1) Name des Kunden
    REM (2) E-Mail des Kunden
    REM     (für den Fall, dass verschiedene Personen den selben Namen tragen)
    REM (3) Slot-ID
    With aSortFields(0): .Field = 0
                         .SortAscending = True: End With
    With aSortFields(1): .Field = 1
                         .SortAscending = True: End With
    With aSortFields(2): .Field = 2
                         .SortAscending = True: End With

    With aSortDesc(0): .Name = "SortFields"
                       .Value = aSortFields(): End With
    With aSortDesc(1): .Name = "ContainsHeader"
                       .Value = True         : End With

    oRange = oSheet.getCellRangeByPosition(0, 0, 4, iRows)
    oRange.sort(aSortDesc())
  End Sub
```

Das Makro im Listing 488 erstellt zuerst eine frei erfundene XML-Datei aus einer Reihe von einfachen Textzeilen (wie man das aus einem DOM-Dokument-Baum macht, sehen Sie im nächsten Abschnitt) und speichert sie in Ihrem Arbeitsverzeichnis. Danach wird die Funktion BuildDOMDoc (s. Listing 484) aufgerufen, die aus dieser Datei einen DOM-Dokument-Baum aufbaut. Ein neu erzeugtes Calc-Dokument wird dann die von der Subroutine GetData gefilterten Daten aufnehmen. Schließlich wird die originale XML-Datei als Textdatei in ein Writer-Dokument importiert, damit Sie die Operation überprüfen können.

Da wir nur Slots gebrauchen können, die „active“ sind, und wir wissen, dass nur aktive Slots den Knoten „validTo“ haben, ruft die Subroutine GetData zuerst die Methode getElementsByTagName des Wurzelknotens auf, die eine Liste aller „validTo“-Knoten zurückgibt, und iteriert danach mit Hilfe der Methode item() über die enthaltenen Elemente.

Ebenso hätten wir eine Liste der „document“-Knoten machen können, denn die Anzahl der Elemente ist dieselbe für dieselben Slots wie mit den „validTo“-Knoten.

Jedes „validTo“-Element ist ein Kindelement eines „slot“-Elements. Mit der Methode getParentNode erhalten wir Zugriff auf das „slot“-Element, um mit der Funktion IsProperSlot zu überprüfen, ob seine Attribute die erwünschten Werte haben. Wenn es so ist, wird das Datum überprüft, das als Text-

knoten im „validTo“-Element steckt. Da ein Textelement nur das einzige Kindelement sein kann, greifen wir mit der Methode getFirstChild direkt darauf zu. Der Wert eines Textknotens ist der Inhalt des Textknotens (s. Tabelle 228). Man erhält ihn mit der Methode getNodeValue:

```
oVtList.item(i).getFirstChild().getNodeValue()
```

Wenn auch diese Überprüfung erledigt ist, können wir endlich die Daten zusammentragen. Wir benötigen den Dokumenttitel und die Attribute der Kunden. Der direkte Weg setzt getElementsByTagName ein, um jeweils eine Liste der Kindknoten des Slot-Elements zu erhalten:

```
oDocs = oSlotElem.getElementsByTagName("document")
oCusts = oSlotElem.getElementsByTagName("customer")
```

Nachdem nun alle relevanten Daten eines Slots beisammen sind, kann sie GetData mit Hilfe der Subroutine ImportSlot in der Calc-Tabelle speichern. Wenn schließlich alle Slots abgearbeitet sind, wird die Calc-Tabelle zur besseren Lesbarkeit sortiert.

15.12.2. Export einer XML-Datei

Für den Export Ihrer Calc-Tabelle als reine, unformatierte Textdatei werden Sie im Normalfall das CSV-Format wählen. Dem einen oder anderen Adressaten könnte aber eine spezifische XML-Datei besser gefallen. Über das DOM-Interface können Sie das mit Basic bewerkstelligen.

Wenn Sie eine XML-Datei importieren, wird das DOM-Dokument automatisch von den Methoden parse oder parseURI des Service DocumentBuilder aufgebaut. Das können Sie gar nicht beeinflussen. Ihnen bleibt nur die Arbeit, im DOM-Dokument die benötigten Daten zu finden.

Daten als XML-Datei zu exportieren heißt, dass zwei Schritte nötig sind. Zuerst müssen Sie das DOM-Dokument von Grund auf neu aufbauen. Danach müssen Sie den Dokument-Baum auslesen, um aus den Daten einfache Textzeilen zu bilden und auszugeben. Mit anderen Worten, Sie müssen alle Elemente in Winkelklammern setzen, die Attributpaare zusammensetzen, die Kommentare nach den XML-Regeln angeben – und die hierarchische Ausgabe in eine lesbare Form bringen.

Ich werde Ihnen eine einfache Ausgabe eines DOM-Dokument-Baumes zu einer lesbar formatierten XML-Textdatei vorstellen. Wieder einmal lasse ich alle Fragen der Validierung, der Namensräume, der Verarbeitungsanweisungen und der Fehlerbehandlung außen vor. Das Makro ist dazu gedacht, jedes benutzerdefinierte DOM-Dokument auszugeben (mit den gerade erwähnten Einschränkungen). Ihre Aufgabe ist es, den DOM-Baum zu bauen.

Wie Sie aus der Tabelle 228 ersehen können, ist das DOM-Dokument ein Spezialknoten. Er kann die ererbten XNode-Methoden nutzen als auch eine Reihe zusätzlicher Methoden, die Sie in der Tabelle 230 finden.

Tabelle 230. *Methoden im Interface com.sun.star.xml.dom.XDocument.*

Methode	Beschreibung (aus der API-Dokumentation übersetzt)
createAttribute(name)	Erzeugt ein Attribut mit dem angegebenen Namen.
createAttributeNS (namespaceURI, qualifiedName)	Erzeugt ein Attribut mit dem angegebenen qualifizierten Namen und dem URI des Namensraums.
createCDATASection(data)	Erzeugt einen CDATASection-Knoten, dessen Wert der angegebene String ist.
createComment(text)	Erzeugt einen Kommentarknoten mit dem angegebenen String.
createDocumentFragment()	Erzeugt ein leeres DocumentFragment-Objekt.
createElement(tagName)	Erzeugt ein Element mit dem angegebenen Tagnamen.
createElementNS (namespaceURI, qualifiedName)	Erzeugt ein Element mit dem angegebenen qualifizierten Namen und dem URI des Namensraums.
createEntityReference(name)	Erzeugt ein EntityReference-Objekt.
createProcessingInstruction (target, data)	Erzeugt eine Verarbeitungsanweisung mit der angegebenen Zielanwendung und dem Datenstring.

Methode	Beschreibung (aus der API-Dokumentation übersetzt)
createTextNode(text)	Erzeugt einen Textknoten mit dem angegebenen String.
getDoctype()	Die mit diesem Dokument verknüpfte DTD (Document Type Declaration).
getDocumentElement()	Direkter Zugriff auf das Wurzelelement des Dokuments.
getElementById(elementIdstring)	Gibt das Element mit der angegebenen ID zurück.
getElementsByTagName (tagname)	Gibt eine NodeList aller Elemente mit dem angegebenen Tagnamen zurück.
getElementsByTagNameNS (namespaceURI, localName)	Gibt eine NodeList aller Elemente mit dem angegebenen lokalen Tagnamen und dem URI des Namensraums zurück.
getImplementation()	Das DOMImplementation-Objekt, das dieses Dokument steuert.
importNode(importedNode, deep)	Importiert einen Knoten aus einem anderen Dokument in dieses Dokument.

Ich führe nun die einzelnen Operationen Schritt für Schritt auf. Sie beginnen damit, dass Sie das DOM-Dokument-Objekt oDOM, den Wurzelknoten, erzeugen und referenzieren.

```
oDocBuilder = CreateUnoService("com.sun.star.xml.dom.DocumentBuilder")
oDOM = oDocBuilder.newDocument()
```

Als nächstes bevölkern Sie das Dokument mit Kindknoten. Diese müssen erst erzeugt und später eingefügt werden. Knoten werden einzig von den „create...“-Methoden des DOM-Dokuments erzeugt. Im Makro spielt es keine Rolle, in welcher Reihenfolge Sie die Knoten erzeugen. Alle Knotenobjekte unterstützen die Methode appendChild, um abhängige Knoten einzufügen. Wieder spielt es keine Rolle, an welcher Stelle im Makro Sie einem Knoten ein Kind einfügen, aber Sie wollen vielleicht die Reihenfolge beachten. Jedes neue Kind wird am Ende der schon vorhandenen Kindknoten angefügt. Wenn die Kindelemente schließlich in der vorliegenden Reihenfolge ausgegeben werden sollen, so sollten Sie beim Einfügen auf die entstehende Ordnung achten. Allerdings gibt es auch die Methode insertChild(newChild, refChild), mit der Sie einen neuen Kindknoten direkt vor ein schon vorhandenes Kind einschieben können. Mit der Methode createComment erzeugen Sie einen Kommentar mit dem Text des Kommentars als Argument. Mit der Methode createElement erzeugen Sie einen Elementknoten mit dem Tagnamen als Argument.

Sie müssen die XML-Namensregeln beachten: Namen dürfen Buchstaben, Ziffern und andere Zeichen enthalten. Leerzeichen sind nicht erlaubt. Das erste Zeichen darf keine Ziffer und auch kein Satzzeichen sein. Der Name darf nicht mit der Zeichenfolge „xml“ (egal ob klein- oder großgeschrieben) beginnen.

```
REM Dem Wurzelknoten hinzugefügte Kindknoten.
oNComment = oDOM.createComment(sComment)
oNPersons = oDOM.createElement("Personen")
oDOM.appendChild(oNComment)
oDOM.appendChild(oNPersons)

REM Einem anderen Knoten hinzugefügter Kindknoten.
oNPlayer = oDOM.createElement("Spieler")
oNPersons.appendChild(oNPlayer)
```

Attribute werden nicht vom DOM-Dokument erzeugt. Man setzt sie direkt – das heißt, man fügt sie an das Ende der Liste an – mit der XElement-Methode setAttribute(name, value), denn für Attribute ist die Reihenfolge nicht so wichtig. Sie werden in einer NamedNodeMap gelistet, und man kann auf sie über Ihren Namen zugreifen. Die Methode setAttribute benötigt zwei Argumente, den Namen des Attributs und seinen Wert, beide Angaben als String. Die folgende Zeile nimmt den Namen aus dem Spaltenkopf und den Wert aus einer zuvor referenzierten Zelle.

```
oNPlayer.setAttribute(oSheet.getCellByPosition(iCol, 0).String, sCellContent)
```

Mit der Methode createTextNode erzeugen Sie einen Textknoten mit dem Textstring als Argument. Da ein Textknoten keine Attribute und auch keine Kindknoten haben kann, braucht man nur selten eine Variable, die den Knoten referenziert:

```
oNName.appendChild(oDOM.createTextNode(sCellContent))
```

Zur Demonstration des gesamten Prozesses habe ich in einer Calc-Tabelle eine Liste von aktiven und nicht mehr aktiven Tennishelden erstellt (s. Bild 117). Diese Tabelle dient als Datenquelle für den XML-Export.

Listing 489. *Eine Calc-Tabelle als Datenquelle für einen XML-Export.*

```
Sub CreateExampleSheet(oSheet)
  Dim aRecords(6)
  Dim i%, j%

  aRecords(0) = Array("ID", "Geschlecht", "Nation", "Name", "Bilanz", _
                      "Titel", "MaxRang", "Melbourne", _
                      "Paris", "London", "NewYork")
  aRecords(1) = Array(1, "m", "USA", "Andre Agassi", "'870:274", 60, 1, _
                      "1995, 2000, 2001, 2003", "1999", "1992", "1994, 1999")
  aRecords(2) = Array(2, "m", "D", "Boris Becker", "'713:214", 49, 1, _
                      "1991, 1996", "", "1985, 1986, 1989", "1989")
  aRecords(3) = Array(3, "m", "CH", "Roger Federer", "'792:186", 67, 1, _
                      "2004, 2006, 2007, 2010", "2009", "2003-2007, 2009", _
                      "2004-2008")
  aRecords(4) = Array(4, "f", "D", "Steffi Graf", "'902:115", 107, 1, _
                      "1988, 1989, 1990, 1994", _
                      "1987, 1988, 1993, 1995, 1996, 1999", _
                      "1988, 1989, 1991, 1992, 1993, 1995, 1996", _
                      "1988, 1989, 1993, 1995, 1996")
  aRecords(5) = Array(5, "f", "RUS", "Anna Kurnikova", "'209:129", 0, 8, _
                      "", "", "", "")
  aRecords(6) = Array(6, "f", "USA", "Martina Navrátilová", "'1140:213", _
                      167, 1, "1981, 1983, 1985", _
                      "1982, 1984", "1978, 1979, 1982-1987, 1990", _
                      "1983, 1984, 1986, 1987")

  For i = 0 To UBound(aRecords())
    For j = 0 To 10
      oSheet.getCellByPosition(j, i).Formula = aRecords(i)(j)
    Next
  Next
End Sub
```

	A	B	C	D	E	F	G	H	I	J	K
1	ID	Geschlecht	Nation	Name	Bilanz	Titel	MaxRang	Melbourne	Paris	London	NewYork
2	1	m	USA	Andre Agassi	870:274	60	1	1995, 2000, 2001, 2003	1999	1992	1994, 1999
3	2	m	D	Boris Becker	713:214	49	1	1991, 1996		1985, 1986, 1989	1989

Bild 117. *Detail der Quelltabelle für den XML-Export.*

Das Makro im Listing 490 zeigt, wie dieses Tabellenblatt in einen DOM-Dokument-Baum umgewandelt wird.

Listing 490. *Aus der Calc-Tabelle im Bild 117 wird ein DOM-Dokument-Baum erstellt.*

```
REM Konvertiert eine Calc-Datentabelle in einen XML-DOM-Baum.
REM Jede Zeile ist ein Datensatz. In der ersten Zeile sind die Feldkennungen.
REM Jedes Feld (Spalte) entspricht einem entsprechenden Knoten als Text oder Attribut.
```

```
REM Diese Funktion ist auf die spezifischen Tabellenblattdaten abgestimmt.
REM Sie wird am besten in der Dokumentbibliothek platziert.
REM Diese Funktion gibt ein DOM-Dokument zurück.
Function MakeXMLFromSheet(oSheet)
  Dim oDocBuilder              'Das Interface DOM DocumentBuilder.
  Dim oDOM                     'Der DOM-Dokument-Baum.
  Dim iRow&, iCol&
  Dim sCellContent As String   'Alle XML-Werte sind Strings.
  Dim sComment As String       'Ein Kommentar, der nicht Teil der Daten ist.
  Dim oNodeComment             'Der Kommentarknoten.
  Dim oNodePersons             'Das Element "Personen".
  Dim oNodePlayer      'Ein "Spieler"-Element als Kind des "Personen"-Elements.
  Dim oNodeName        'Ein "Name"-Element als Kind eines "Spieler"-Elements.
  Dim oNodeStats       'Ein "Statistik"-Element als Kind eines "Spieler"-Elements.
  Dim oNodeGrandSlams  'Ein "GrandSlams"-Element als Kind eines "Spieler"-Elements.
  Dim oNodeGrandSlam   'Ein "GrandSlam"-Element als Kind eines "GrandSlams"-Elements.

  REM Erzeugt den DOM-Dokument-Baum.
  oDocBuilder = CreateUnoService("com.sun.star.xml.dom.DocumentBuilder")
  oDOM = oDocBuilder.newDocument()

  REM Das DOM-Dokument wird folgende Baumstruktur haben:
  REM <!-- ... -->
  REM <Personen>
  REM   <Spieler ID="..." Geschlecht="..." Nation="...">
  REM     <Name>Der Name des Spielers</Name>
  REM     <Statistik Bilanz="..." Titel="..." MaxRang="..." />
  REM     <GrandSlams>
  REM       <Melbourne>Liste der Siegjahre</Melbourne>
  REM       <Paris>...</Paris>
  REM       ... (weitere Grand-Slam-Knoten)
  REM     </GrandSlams>
  REM   </Spieler>
  REM   ... (weitere Spielerknoten)
  REM </Personen>

  sComment = "Dies ist eine Liste berühmter Tennisspieler," & Chr$(10) & _
             "     die den ATP-Tenniszirkus in der Vergangenheit geprägt haben."
  REM Erzeugt den Kommentarknoten und das "Personen"-Element und fügt beide
  REM am Ende der Liste der Kindelemente des Wurzelknotens ein.
  oNodeComment = oDOM.createComment(sComment)
  oNodePersons = oDOM.createElement("Personen")
  oDOM.appendChild(oNodeComment)
  oDOM.appendChild(oNodePersons)

  REM Der Zugriff auf die Datensatzzellen beginnt mit Zeile 1,
  REM da in der Zeile 0 die Überschriften sind.
  iRow = 1
  Do While oSheet.getCellByPosition(0, iRow).Value <> 0
    REM Mit jeder neuen Zeile wird ein neues "Spieler"-Element erzeugt und
    REM am Ende der Liste der Kindelemente des "Personen"-Elements eingefügt.
    oNodePlayer = oDOM.createElement("Spieler")
    oNodePersons.appendChild(oNodePlayer)
    REM Erzeugt drei Elemente und fügt sie am Ende der Liste
    REM der Kindelemente des "Spieler"-Elements ein.
    oNodeName = oDOM.createElement("Name")
```

```
      oNodeStats = oDOM.createElement("Statistik")
      oNodeGrandSlams = oDOM.createElement("GrandSlams")
      oNodePlayer.appendChild(oNodeName)
      oNodePlayer.appendChild(oNodeStats)
      oNodePlayer.appendChild(oNodeGrandSlams)
      For iCol = 0 To 10
        sCellContent = Trim(oSheet.getCellByPosition(iCol, iRow).String)
        Select Case iCol
          Case 0, 1, 2  'Die Spieler-Attribute ID, Geschlecht und Nation
            REM Erstellt die Attribute mit zwei Argumenten:
            REM Name von der Überschrift, Wert aus der aktuellen Zelle.
            oNodePlayer.setAttribute _
              (oSheet.getCellByPosition(iCol, 0).String, sCellContent)
          Case 3
            REM Fügt einen neuen Textknoten mit dem Spielernamen als Wert
            REM als einziges Kind des "Name"-Elements ein.
            oNodeName.appendChild(oDOM.createTextNode(sCellContent))
          Case 4, 5, 6     'Die Spielerstatistik
            REM Erstellt die Attribute mit zwei Argumenten:
            REM Name von der Überschrift, Wert aus der aktuellen Zelle.
            oNodeStats.setAttribute _
              (oSheet.getCellByPosition(iCol, 0).String, sCellContent)
          Case 7, 8, 9, 10  'Die Grand-Slam-Siege des Spielers.
            REM Erzeugt ein neues Grand-Slam-Element und fügt es am Ende der Liste
            REM der Kindelemente des "GrandSlams"-Elements ein.
            REM Die Überschrift wird als Tagname genommen.
            oNodeGrandSlam = oDOM.createElement _
              (oSheet.getCellByPosition(iCol, 0).String)
            oNodeGrandSlams.appendChild(oNodeGrandSlam)
            REM Fügt einen neuen Textknoten mit den Grand-Slam-Siegjahren als Wert
            REM als einziges Kind des Grand-Slam-Elements ein.
            oNodeGrandSlam.appendChild(oDOM.createTextNode(sCellContent))
        End Select
      Next
      iRow = iRow + 1
    Loop
    MakeXMLFromSheet = oDOM
  End Function
```

Eine schnörkellose Sache. Das Makro erzeugt einen Kommentar und das Container-Element „Personen“ und fügt beide dem Wurzelknoten oDOM hinzu. Dann iteriert es über die Zeilen der Datensätze. Für jede neue Zeile wird ein Element namens „Spieler“ erzeugt und dem Element „Personen“ hinzugefügt. Jeder der „Spieler“-Knoten wird drei Kindelemente mit den Namen „Name“, „Statistik“ und „GrandSlams“ enthalten. Sie werden also erzeugt und dem Element „Spieler“ hinzugefügt.

Nun iteriert das Makro über die Spalten. Die Spalten 0, 1 und 2 sind Attribute des Elements „Spieler“. Die Attribute werden aus dem Zellinhalt als Wert und der Spaltenüberschrift als Name gebildet. Das ist möglich, weil die Namen der Überschriften unter Berücksichtigung der XML-Namensregeln gewählt wurden. Spalte 3 enthält den Namen der Person. Er wird als Textknoten dem Element „Name“ hinzugefügt. Die Spalten 4, 5 und 6 sind Attribute des Elements „Statistik“. Diese Attribute werden genauso gebildet wie die aus den Spalten 0, 1 und 2. Schließlich folgen die Spalten 7 bis 10 mit den Jahren der Siege in den vier Grand-Slam-Turnieren. Für jeden der Turnierorte wird ein neues Element gebildet mit der Überschrift als Tagname und dem Element „GrandSlams“ hinzugefügt. Der Inhalt der aktuellen Zelle wird diesem neuen Grand-Slam-Element als Textknoten hinzugefügt.

All dies spielt sich im Arbeitsspeicher ab. Wie nun ein DOM-Dokument als einfache Textdatei in gültiger und lesbar formatierter XML-Notation ausgegeben wird, können Sie im Listing 493 nachle-

sen. Vorher stelle ich Ihnen aber noch die vom Interface XElement eingesetzten Methoden vor (s. Tabelle 231).

Tabelle 231. *Methoden im Interface com.sun.star.xml.dom.XElement.*

Methode	Beschreibung (aus der API-Dokumentation übersetzt)
getAttribute(name)	Gibt einen Attributwert über seinen Namen zurück.
getAttributeNode(name)	Gibt einen Attributknoten über seinen Namen zurück.
getAttributeNodeNS (namespaceURI, name)	Gibt einen Attributknoten über seinen lokalen Namen und URI des Namensraums zurück.
getAttributeNS(namespaceURI, name)	Gibt einen Attributwert über seinen lokalen Namen und URI des Namensraums zurück.
getElementsByTagName(name)	Gibt eine Liste aller abhängigen Elemente mit dem angegebenen Namen als NodeList zurück.
getElementsByTagNameNS (namespaceURI, name)	Gibt eine Liste aller abhängigen Elemente mit dem angegebenen lokalen Namen und URI des Namensraums als NodeList zurück.
getTagName()	Gibt den Namen des Elements zurück.
hasAttribute(name)	Gibt True zurück, wenn ein Attribut mit dem angegebenen Namen für dieses Element existiert.
hasAttributeNS(namespaceURI, name)	Gibt True zurück, wenn ein Attribut mit dem angegebenen lokalen Namen und URI des Namensraums für dieses Element existiert.
removeAttribute(name)	Löscht ein Attribut über den Namen.
removeAttributeNode(oldAttr)	Löscht den angegebenen Attributknoten.
removeAttributeNS (namespaceURI, name)	Löscht ein Attribut über den lokalen Namen und URI des Namensraums.
setAttribute(name, value)	Fügt ein neues Attribut ein.
setAttributeNode(newAttr)	Fügt einen neuen Attributknoten ein.
setAttributeNodeNS(newAttr)	Fügt ein neues Attribut ein.
setAttributeNS (namespaceURI, qualifiedName, value)	Fügt ein neues Attribut ein.

Tabelle 231 zeigt die vom Spezialknoten-Interface XElement eingesetzten Methoden. Sie ergänzen die Methoden des elterlichen XNode-Interface (s. Tabelle 229).

Die folgenden Subroutinen und Funktionen sind unabhängig von irgendwelchen DOM-Dokumenten. Sie können auf jedes DOM-Dokument angewendet werden (so hoffe ich jedenfalls), das mit der Einschränkung aufgebaut wurde, dass keine Validierung, Namensräume und Verarbeitungsanweisungen eingesetzt werden. Es findet auch keine Fehlerbehandlung statt. Sie können also diese Subroutinen und Funktionen in Ihrer „Meine Makros“-Bibliothek speichern. Sie brauchen nicht einmal zu wissen, wie sie funktionieren.

Die einzige Subroutine, an die Sie Hand anlegen müssen, ist ExportSheetToXML. Die drei kommentierten Zeilen ganz am Anfang der Subroutine könnten für Minimalansprüche ausreichen: entfernen Sie die Kommentar-Anweisung und werfen Sie die restlichen Zeilen weg. Vielleicht wollen Sie aber nicht immer nur das Tabellenblatt 0 der aktiven Komponente als Grundlage für die spezifische MakeXMLFromSheet-Funktion (s. Listing 490) nutzen.

Wenn Sie kein Interesse an der Funktionsweise haben, können Sie den Rest dieses Abschnitts überspringen.

So, Sie sind also interessiert. Der Code inspiziert jeden existierenden Knoten, kümmert sich aber nur um Kommentare, Elemente und Attribute. Er iteriert über Listen von Kindknoten und sucht die Spe-

zialknoten COMMENT_NODE, ELEMENT_NODE und TEXT_NODE (s. Tabelle 228). Die wesentliche Technik ist wie folgt. Sie können statt der If-Else-Konstruktion natürlich auch Select Case nehmen.

Listing 491. *Behandlung von Kindknoten.*

```
If oNode.hasChildNodes() Then
  oChildren = oNode.getChildNodes()
  For i = 0 To oChildren.getLength() - 1
    oChild = oChildren.item(i)
      If oChild.getNodeType() = com.sun.star.xml.dom.NodeType.COMMENT_NODE Then
        …
      ElseIf Child.getNodeType() = com.sun.star.xml.dom.NodeType.ELEMENT_NODE Then
        …
      ElseIf Child.getNodeType() = com.sun.star.xml.dom.NodeType.TEXT_NODE Then
        …
....
```

Im Falle eines Kommentars oder eines Textknotens gibt es keine Attribute und auch keine Kindknoten, nur Text. Dieser Text ist dann der Knotenwert:

```
sText = oChild.getNodeValue()
```

Elementknoten können Attribute haben. Mit der Methode getAttributes erhalten Sie Zugriff auf die Attributknoten der Elemente. Kommentare und Textknoten haben keine Attribute. Die Methode gibt ein NamedNodeMap zurück, eine einerseits enumerierte und andererseits benannte Liste. Wenn keine Attribute existieren, wird Null zurückgegeben. Da der Code die Attributnamen nicht kennt, greift er auf die einzelnen Attribute über ihren Indexwert zu und holt sich Namen und Wert über die Methoden getNodeName und getNodeValue. Die Technik:

Listing 492. *Iteration über jedes Attribut eines Elements.*

```
oAttList = oElement.getAttributes()
If Not IsNull(oAttList) Then
  For i = 0 To oAttList.getLength() - 1
    oAtt = oAttList.item(i)
    sValue = oAtt.getNodeValue()
    sName = oAtt.getNodeName()
....
```

Elementknoten können Kindknoten enthalten, mehrere Kommentare, mehrere Elemente oder einen Textknoten. Der Code inspiziert diese Kindknoten rekursiv. Die Subroutine PrintDom wird für jede Elementebene aufgerufen.

Listing 493. *Makro, das ein DOM-Dokument als XML-Textdatei ausgibt.*

```
REM Exportiert den Inhalt eines Calc-Tabellenblatts in eine XML-Datei.
Sub ExportSheetToXML
  REM Normalerweise ist dies eine sehr kurze Routine:
  'Dim sURL
  'sURL = ChooseAFileName  'Eine Funktion, die Sie im Modul UNO dieser Datei finden.
  'WriteDomToFile(MakeXMLFromSheet(ThisComponent.Sheets(0)), sURL)
  REM Für dieses Beispiel wird ein neues Calc-Dokument erzeugt,
  REM in dessen Tabellenblatt 0 ein paar Daten geschrieben werden.
  REM Schließlich wird die gespeicherte XML-Datei zur Ansicht im Writer geöffnet.

  Dim oCalcDoc        'Das als Quelle dienende Calc-Dokument.
  Dim oSheet          'Die Quelltabelle.
  Dim sURL As String  'Der URL der Ausgabedatei.
  Dim noArgs(0) As New com.sun.star.beans.PropertyValue
  Dim oPathSettings   'Die Pfadeinstellungen der Anwendung.
```

```
  oPathSettings = CreateUnoService("com.sun.star.util.PathSettings")

  sURL = oPathSettings.Work & "/oome_xml_export.xml"
  oCalcDoc = StarDesktop.loadComponentFromURL( _
             "private:factory/scalc", "_blank", 0, Array())
  oSheet = oCalcDoc.Sheets(0)
  REM Füllt die Tabelle mit Daten.
  CreateExampleSheet(oSheet)
  REM Die Subroutine MakeXMLFromSheet muss die spezifischen Tabellendaten verwenden.
  REM Sie liegt daher normalerweise in der Dokument-Bibliothek.
  WriteDomToFile(MakeXMLFromSheet(oSheet), sURL)
  Wait 500             'Lässt dem OS Zeit, die Datei auf der Platte zu speichern.
  REM Öffnet die XML-Datei als Writer-Dokument.
  REM Für LO muss FilterName spezifiziert werden, für AOO nicht.
  noArgs(0).Name = "FilterName"
  noArgs(0).Value = "Text"
  StarDesktop.loadComponentFromURL(sURL, "_blank", 0, noArgs())
End Sub

REM Schreibt die Struktur des aktuellen DOM-Dokuments
REM als gültigen und lesbar formatierten XML-Text in eine Textdatei.
REM Diese Subroutine und die aufgerufenen Subroutinen und Funktionen sind nicht
REM dokumentspezifisch und brauchen nicht in der Dokument-Bibliothek zu sein.
REM Sie können in der "Meine Makros"-Bibliothek gespeichert werden.
Sub WriteDomToFile(oDOM, sFilePath As String)
  REM Autor: Volker Lenhardt
  Dim oSimpleFileAccess 'Der Service SimpleFileAccess.
  Dim oOutputStream     'Von SimpleFileAccess zurückgegebener Stream.
  Dim oTextOutput       'Der Service TextOutputStream.
  Dim oNodes            'Liste der Kindknoten des Wurzelknotens.
  Dim i%
  Dim iIndentLevel As Integer  'Einrückungsebene.
  Dim iIndentSpaces As Integer 'Anzahl der für jede Einrückungsebene
                               'hinzugefügten Leerzeichen.

  On Error Goto Catch

  iIndentSpaces = 2
  REM Der Ausgabestream.
  sFilePath = ConvertToURL(sFilePath)
  oSimpleFileAccess = CreateUnoService("com.sun.star.ucb.SimpleFileAccess")
  With oSimpleFileAccess
    If .exists(sFilePath) Then .kill(sFilePath)
    oOutputStream = .openFileWrite(sFilePath)
  End With

  oTextOutput = CreateUnoService("com.sun.star.io.TextOutputStream")
  With oTextOutput
    .OutputStream = oOutputStream
    .setEncoding("UTF-8")
    REM Die erste Zeile ist eine Verarbeitungsanweisung. Sie ist aber kein Teil
    REM des DOM-Dokuments. Also schreiben wir sie separat.
    .WriteString("<?xml version=""1.0"" encoding=""UTF-8""?>" & Chr$(10))

    REM Ein DOM-Baum kann auf der Wurzelebene null, einen oder mehrere
    REM Kindknoten haben. Diese werden hierarchisch behandelt.
```

```
    If oDOM.hasChildNodes() Then
      oNodes = oDOM.getChildNodes()
      For i = 0 To oNodes.getLength() - 1
        PrintDom(oNodes.item(i), oTextOutput, iIndentLevel, iIndentSpaces)
      Next
    End If
    .closeOutput()
  End With
  oOutputStream.closeOutput()
  Exit Sub

  Catch:
  Print "Fehler " & Err & " (" & Error(Err) & ")"
End Sub

REM Schreibt die Elemente eines DOM-Baums rekursiv Zeile für Zeile
REM in eine Textdatei. Es beginnt mit der hierarchischen Ebene iLevel 0.
REM Niedrigere Ebenen werden durch iIndent Leerzeichen (akkumulierend) eingerückt,
REM außer bei Textknoten: sie werden in derselben Zeile wie die Element-Tags
REM geschrieben, direkt nach dem Starttag und direkt vor dem Endetag.
REM Es wird vorausgesetzt, dass es neben einem Textknoten keine weiteren
REM Kindknoten eines Elements gibt.
Sub PrintDom(oNode, oStream, iLevel As Integer, iIndent As Integer)
  REM Autor: Volker Lenhardt
  Dim oElementChildren
  Dim oChild
  Dim sLine As String
  Dim sAtt As String
  Dim sIndent As String
  Dim i%, iLen%
  Dim sNodeName As String

  sNodeName = oNode.getNodeName()
  sIndent = String(iLevel * iIndent, " ")

  REM Es werden nur Kommentare und Elemente berücksichtigt.
  If oNode.getNodeType() = com.sun.star.xml.dom.NodeType.COMMENT_NODE Then
    REM XML-Kommentar.
    sLine = sIndent & "<!-- " & oNode.getNodeValue() & " -->"
    oStream.writeString(sLine & Chr$(10))
  ElseIf oNode.getNodeType() = _
      com.sun.star.xml.dom.NodeType.ELEMENT_NODE Then
    REM XML-Element.
    sAtt = AttString(oNode.getAttributes())
    REM Prüft, ob das Element Daten enthält. Ansonsten wird es übergangen.
    If oNode.hasChildNodes() Or sAtt <> "" Then
      oElementChildren = oNode.getChildNodes()
      If HasContent(oElementChildren) Then
        sLine = sIndent & "<" & sNodeName & sAtt & ">"  'Die Zeile mit dem Starttag.
        iLen = oElementChildren.getLength()
        If iLen = 1 Then
          REM Ist es ein Textknoten? Dann gibt es keine weiteren Geschwisterknoten.
          oChild = oElementChildren.item(0)
          If oChild.getNodeType() = com.sun.star.xml.dom.NodeType.TEXT_NODE Then
            sLine = sLine & oChild.getNodeValue() & "</" & sNodeName & ">"
            REM In der Zeile stehen nun Starttag plus Textwert plus Endetag.
            oStream.writeString(sLine & Chr$(10))
            Exit Sub
```

```
          End If
        End If
        REM An diesem Punkt gibt es Kindelemente, die keine Textknoten sind.
        REM Die Zeile mit dem Starttag wird geschrieben.
        oStream.writeString(sLine & Chr$(10))
        For i = 0 To iLen - 1
          REM Start der Rekursion, die Einrückung wird erweitert.
          PrintDom(oElementChildren.item(i), oStream, iLevel + 1, iIndent)
        Next
        REM Die Zeile mit dem Endetag wird geschrieben.
        sLine = sIndent & "</" & sNodeName & ">"
        oStream.writeString(sLine & Chr$(10))
      Else
        REM Es gibt keine Kindelemente. Wenn es Attribute gibt,
        REM wird die Kurznotation verwendet.
        REM Falls es auch keine Attribute gibt, wird gar kein Elementtag geschrieben.
        If sAtt <> "" Then
          sLine = sIndent & "<" & sNodeName & sAtt & " />"
          REM Das Element in Kurznotation.
          oStream.writeString(sLine & Chr$(10))
        End If
      End If
    End If
  End If
End Sub

REM Gibt einen String mit allen Attributen eines Elements zurück.
Function AttString(oAttList) As String
  REM Autor: Volker Lenhardt
  Dim oAtt                    'Ein einzelner Attributknoten.
  Dim sValue As String        'Ein einzelner Attributwert.
  Dim sAtts As String         'Der String mit allen Attributen.
  Dim i%

  If Not IsNull(oAttList) Then
    For i = 0 To oAttList.getLength() - 1
      oAtt = oAttList.item(i)
      sValue = oAtt.getNodeValue()
      If sValue <> "" Then
        sAtts = sAtts & " " & oAtt.getNodeName() & "=""" & sValue & """"
      End If
    Next
  End If
  AttString = sAtts
End Function

REM Prüft rekursiv, ob ein Kindknoten mit Inhalt existiert,
REM sei es Text, Kommentar oder ein Attributwert größer als "".
REM Gibt True zurück, wenn im Baum Inhalt gefunden wurde, ansonsten False.
REM Wird von PrintDom aufgerufen, bevor der Starttag des Elements geschrieben wird,
REM so kann PrintDom auf die Ausgabe eines leeren Elements verzichten.
Function HasContent(oElementList) As Boolean
  REM Autor: Volker Lenhardt
  Dim oChild          'Ein einzelner Kindknoten aus oElementList
  Dim oAttributes     'Die Attribute von oChild
  Dim oChildren       'Die Kindknoten von oChild
  Dim i%, j%
```

```
    For i = 0 To oElementList.getLength() - 1
      oChild = oElementList.item(i)
      If oChild.hasAttributes() Then
        oAttributes = oChild.getAttributes()
        For j = 0 To oAttributes.getLength() - 1
          If oAttributes.item(j).getNodeValue() <> "" Then
            HasContent = True
            Exit Function
          End If
        Next
      End If
      If oChild.getNodeType() = com.sun.star.xml.dom.NodeType.TEXT_NODE Then
        If oChild.getNodeValue() <> "" Then
            HasContent = True
            Exit Function
        End If
      ElseIf oChild.getNodeType() = com.sun.star.xml.dom.NodeType.COMMENT_NODE Then
        If oChild.getNodeValue() <> "" Then
            HasContent = True
            Exit Function
        End If
      Else
        oChildren = oChild.getChildNodes()
        If oChildren.getLength() <> 0 Then
          REM Start der Rekursion.
          If HasContent(oChildren) Then
            HasContent = True
            Exit Function
          End If
        End If
      End If
    Next
    HasContent = False
  End Function
```

Die Subroutine WriteDomToFile erwartet zwei Argumente, das DOM-Dokument und den URL der Ausgabedatei. Es errichtet zuerst einen SimpleFileAccess, der eine eventuell existierende Datei mit dem betreffenden Namen überschreibt. Für die UTF-8-kodierte Ausgabe wird ein spezieller TextOutputStream verwendet. Die Daten werden Zeile für Zeile geschrieben.

Die erste Zeile einer XML-Datei ist die XML-Verarbeitungsanweisung, die normalerweise kein Teil des DOM-Dokuments ist, also separat geschrieben werden muss. Danach iteriert das Makro über die Liste der Kindknoten des Wurzelknotens, des DOM-Dokuments. Die Subroutine PrintDom wird gestartet, die jedes Element inspiziert und sich selbst rekursiv aufruft, falls es Kindknoten zu einem Element gibt.

PrintDom erwartet vier Argumente: den aktuellen Knoten, den Ausgabestream, die Einrückungsebene und die Anzahl der Leerzeichen für die Einrückung. Ich habe mich für zwei Leerzeichen entschieden. Die oberste Ebene 0 wird nicht eingerückt, für jede niedrigere Ebene – das heißt, für jeden folgenden Aufruf von PrintDom – werden zwei Leerzeichen hinzugefügt.

Wenn der aktuelle Knoten ein Kommentar ist, schreibt PrintDom den Kommentartext in der XML-Notation <!-- Text -->. Zeilenumbrüche innerhalb des Textes umbrechen auch die Ausgabezeilen.

Wenn der aktuelle Knoten ein Element ist, sucht PrintDom nach vorhandenen Attributen. Die Funktion AttString übernimmt die Liste der Attribute und gibt die zu einer Zeichenfolge verketteten „Name=Wert“-Paare zurück: Att1="1" Att2="2" Att3="3". Falls dieser String leer ist und auch keine

Kindknoten existieren, wird keine Elementzeile geschrieben, und PrintDom ist beendet. Ansonsten wird der Starttag <Name AttString> gebildet.

Ob Starttag und Endetag des Elements in zwei verschiedenen Zeilen geschrieben werden, hängt davon ab, ob es Kindknoten mit Inhalt gibt. Denn wenn Elemente keine Daten enthalten, werden sie ja nicht geschrieben. Die Funktion HasContent sucht rekursiv nach irgendwelchem Inhalt – Kommentare, Attribute, Textknoten – in allen darunter liegenden Ebenen und gibt in dem Moment True zurück, wenn sie auf Inhalt stößt. Wenn keine Daten gefunden werden, wird der Starttag mit Leerzeichen, Schrägstrich, schließende Winkelklammer (/>) abgeschlossen. Ein Endetag ist nicht mehr nötig. Falls dazu noch der Attributstring leer sein sollte, wird gar keine Zeile geschrieben.

Falls es nur einen einzigen Kindknoten gibt, prüft PrintDom, ob es ein Textknoten ist. In diesem Fall wird der Text auf derselben Zeile direkt nach dem Starttag geschrieben, direkt gefolgt vom Endetag: <Name AttString>Text</Name>.

An diesem Punkt weiß PrintDom, dass es Kindknoten gibt, von denen mindestens einer Daten enthält. Also wird die Zeile mit dem Starttag geschrieben und PrintDom ruft sich für jeden Kindknoten selbst auf. Nachdem der PrintDom-Aufruf mit dem letzten Kindknoten beendet ist, wird die Zeile mit dem Endetag geschrieben.

15.13. Diagramme

Diagramme bieten vielfältige Möglichkeiten, doch nur ein kleiner Teil davon soll hier ausreichen, einen Überblick darüber zu bieten, was alles zur Verfügung steht.

Ein Diagramm ist ein Dokument und hat sein eigenes Dokument-Modell. Es ist aber immer in einem anderen OOo-Dokument eingebettet. Objekte, die eingebettete Diagramme unterstützen (Tabellenblätter), besitzen die Methode getCharts des Interface XTableChartsSupplier. Diese Methode gibt den Service TableCharts zurück.

```
ThisComponent.Sheets(0).getCharts()
```

Tabelle 232. *Methoden im Interface com.sun.star.table.XTableCharts.*

Methode	Beschreibung
addNewByName(name, rechteck, daten, bSpaltenkopf, bZeilenkopf)	Erzeugt ein benanntes Diagramm und fügt es ein.
createEnumeration()	Erzeugt eine Enumeration.
getByIndex(i)	Zugriff über den Index.
getByName(name)	Zugriff über den Namen.
getCount()	Anzahl der Diagramme.
getElementNames()	Array der Diagrammnamen.
hasByName(name)	Ermittelt, ob ein bestimmtes benanntes Diagramm existiert.
hasElements()	Ermittelt, ob überhaupt ein Diagramm existiert.
removeByName(name)	Entfernt ein benanntes Diagramm.

Um Diagramme zu untersuchen, benötigen wir erst einmal Daten.

Listing 494. *Erzeugung von Beispieldaten für Diagramme.*

```
Function CreateCalcForChart
  Dim oCalcDoc        'Das Calc-Tabellendokument
  Dim oSheet          'Das Tabellenblatt
  Dim sDataRng$       'Adresse des Datenbereichs
  oCalcDoc = StarDesktop.loadComponentFromURL("private:factory/scalc", _
                                              "_blank", 0, Array())
```

```
    oSheet = oCalcDoc.Sheets(0)
    sDataRng = "A1:D8"
    Dim oData
    oData = Array(Array("", "6 Uhr", "12 Uhr", "18 Uhr"), _
                  Array("Montag", 12, 23, 29), _
                  Array("Dienstag", 14, 24, 30), _
                  Array("Mittwoch", 15, 25, 31), _
                  Array("Donnerstag", 11, 20, 26), _
                  Array("Freitag", 19, 27, 34), _
                  Array("Samstag", 20, 28, 35), _
                  Array("Sonntag", 16, 23, 28))
    oSheet.getCellRangeByName(sDataRng).setDataArray(oData)
    CreateCalcForChart = oCalcDoc
End Function
```

Der Tabellenausschnitt mit den Beispieldaten sieht so aus:

Tabelle 233. *Beispieldaten für Diagramme.*

	A	B	C	D
1		6 Uhr	12 Uhr	18 Uhr
2	Montag	12	23	29
3	Dienstag	14	24	30
4	Mittwoch	15	25	31
5	Donnerstag	11	20	26
6	Freitag	19	27	34
7	Samstag	20	28	35
8	Sonntag	16	23	28

Das Beispiel unten erzeugt ein einfaches Diagramm. Zuerst wird ein Calc-Dokument mit einfachen Daten erzeugt (s. Listing 494 und Tabelle 233). Wenn es das Diagramm noch nicht gibt, wird es mit Abstand von 1 cm vom linken Rand und 5 cm vom oberen Rand erzeugt. Die Grafik hat eine Größe von 20 cm x 10 cm. Der Methode addNewByName wird als drittes Argument der Adressstring eines Zellbereichs übergeben, in dem die Daten für das Diagramm definiert sind.

Listing 495. *Erzeugt ein einfaches Diagramm.*

```
Sub CreateCalcWithSimpleChart
  Dim oSheet     'Tabellenblatt, das das Diagramm enthält
  Dim oRect      'Position und Größe des Diagramms als Rechteck
  Dim oCharts    'Die im Tabellenblatt enthaltenen Diagramme
  Dim oChart     'Das neu erzeugte Diagramm
  Dim aAddresses 'Array der Datenadresse
  Dim sName$     'Diagrammname
  Dim oChartDoc  'Eingebettetes Diagrammdokument
  Dim oTitle     'Diagrammtitel-Objekt
  Dim oLegend    'Legende-Objekt des Diagrammdokuments
  Dim oArea      'Fläche-Objekt des Diagrammdokuments
  Dim oDiagram   'Diagrammtyp-Objekt
  Dim oWall      'Diagrammwand-Objekt
  Dim oYAxis     'Y-Achse-Objekt
  Dim oYMainGrid 'Hauptgitterlinien-Objekt
  Dim oYHelpGrid 'Hilfsgitterlinien-Objekt
  Dim sDataRng$  'Adressstring der darzustellenden Daten
  Dim sCategRng$ 'Adressstring der Zeilenbeschriftung
  Dim oCalcDoc
```

```
oCalcDoc = CreateCalcForChart()

sName = "Temperaturen 1"
sCategRng = "A2:A8"
sDataRng = "B1:D8"

oSheet = oCalcDoc.Sheets(0)
aAddresses = Array(oSheet.getCellRangeByName(sCategRng).getRangeAddress(), _
                   oSheet.getCellRangeByName(sDataRng).getRangeAddress())
oCharts = oSheet.getCharts()
If Not oCharts.hasByName(sName) Then
  'Die Masseinheit für ein Rechteck ist 1/100 mm.
  oRect       = CreateObject("com.sun.star.awt.Rectangle")
  oRect.X     = 1000     'Position von links
  oRect.Y     = 5000     'Position von oben
  oRect.Width = 20000    'Breite
  oRect.Height= 10000    'Höhe

  '1. Argument: Name des Diagramms.
  '2. Argument: Diagrammfläche: Position und Größe
  '3. Argument: Array der Datenbereichsadressen.
  '             Die Daten können aus getrennten Zellbereichen stammen.
  '4. Argument: Spaltenbeschriftung.
  '             True: Die Werte werden der ersten Datenzeile entnommen.
  '             False: Die Werte werden automatisch bestimmt.
  '5. Argument: Zeilenbeschriftung.
  '             True: Die Werte werden der ersten Datenspalte entnommen.
  '             False: Die Werte werden automatisch bestimmt.
  oCharts.addNewByName(sName, oRect, aAddresses, True, True)
End If
oChart = oCharts.getByName(sName)
'Das Diagrammdokument wird erzeugt.
oChartDoc = oChart.getEmbeddedObject()
oTitle = oChartDoc.getTitle()
oTitle.String = "Temperaturen der Woche"
oChartDoc.HasLegend = True
'Die Legende soll weiß sein und keinen Rahmen haben.
oLegend = oChartDoc.getLegend()
With oLegend
  .FillColor = RGB(255, 255, 255)
  .LineWidth = 0
  .LineStyle = com.sun.star.drawing.LineStyle.NONE
End With
'Die Diagrammfläche soll einen blauen Rahmen haben.
oArea = oChartDoc.getArea()
With oArea
  .LineColor = RGB(52, 101, 164)
  .LineStyle = com.sun.star.drawing.LineStyle.SOLID
  .LineWidth = 100
End With

'Erzeugt einen Diagrammtyp.
oDiagram = oChartDoc.createInstance("com.sun.star.chart.LineDiagram")
oChartDoc.setDiagram(oDiagram)
oDiagram.DataCaption = com.sun.star.chart.ChartDataCaption.VALUE
oDiagram.DataRowSource = com.sun.star.chart.ChartDataRowSource.COLUMNS
```

```
    'Die Diagrammwand soll hellbelb mit grauem Rahmen sein.
    oWall = oDiagram.getWall()
    With oWall
      .FillStyle = com.sun.star.drawing.FillStyle.SOLID
      .FillColor = RGB(255, 255, 204)
      .LineStyle = com.sun.star.drawing.LineStyle.SOLID
      .LineColor = RGB(179, 179, 179)
    End With
    'Das Diagramm hat waagerechte Gitterlinien für jeden Intervallschritt
    'und Hilfslinien dazwischen.
    oDiagram.HasYAxisGrid = True
    oDiagram.HasYAxisHelpGrid = True
    'Die Skalierungen der Y-Achse.
    oYAxis = oDiagram.getYAxis()
    With oYAxis
      .Min = 10            'Minimum
      .StepMain = 4        'Hauptintervall
      .StepHelpCount = 2   'Hilfsintervall Schritte
    End With
    'Die Hauptgitterlinien sind mittelgrau.
    oYMainGrid = oDiagram.getYMainGrid()
    With oYMainGrid
      .LineStyle = com.sun.star.drawing.LineStyle.SOLID
      .LineColor = RGB(179, 179, 179)
    End With
    'Die Hilfsgitterlinien sind heller grau.
    oYHelpGrid = oDiagram.getYHelpGrid()
    With oYHelpGrid
      .LineStyle = com.sun.star.drawing.LineStyle.SOLID
      .LineColor = RGB(221, 221, 221)
    End With
End Sub
```

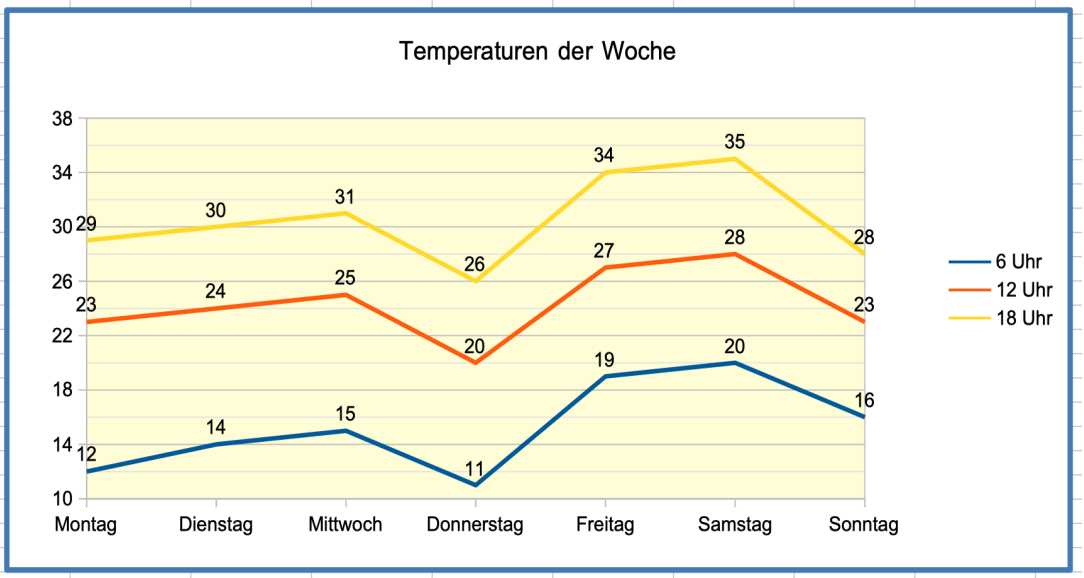


Bild 118. *Einfaches Diagramm mit Rahmen.*

Ein Diagramm innerhalb eines Dokuments kann modifiziert werden. Das folgende Makro ändert das eingefügte Diagramm dahin, dass drei Zellbereiche verwendet werden, die nicht verbunden sind.

Listing 496. *Modifiziert ein existierendes Diagramm.*

```
Sub ModifyChart
  Dim sName$, oSheet, oCharts, oChart, oData
  sName = "Temperaturen 1"

  oSheet = ThisComponent.Sheets(0)
  oCharts = oSheet.getCharts()
  If oCharts.hasByName(sName) Then
    oChart = oCharts.getByName(sName)
    oData = Array(oSheet.getCellRangeByName("A2:A8").getRangeAddress(), _
        oSheet.getCellRangeByName("B1:B8").getRangeAddress(), _
        oSheet.getCellRangeByName("D1:D8").getRangeAddress())
    oChart.setRanges(oData)
  End If
End Sub
```

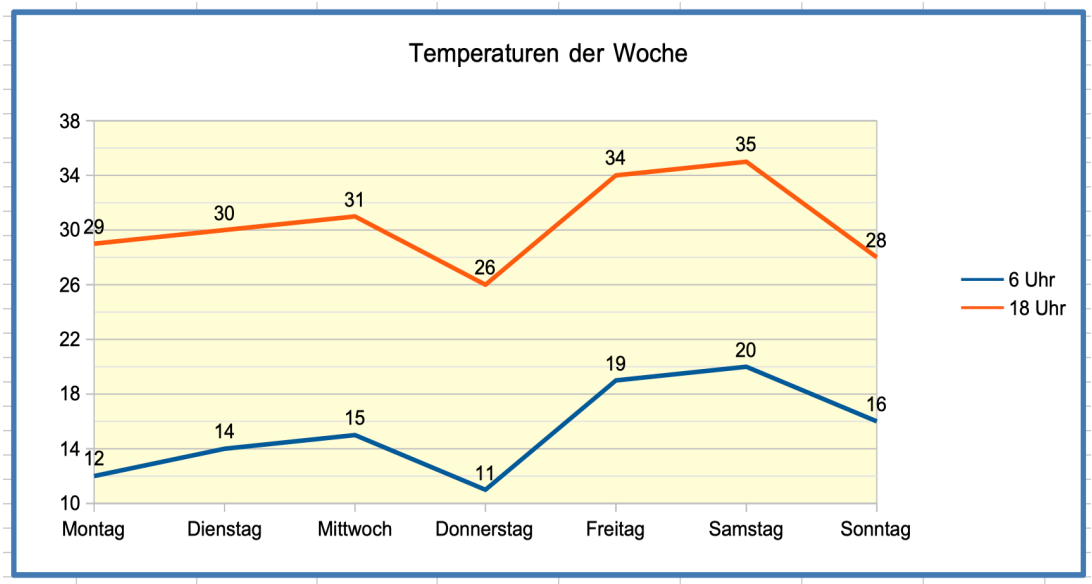


Bild 119. *Modifiziertes Diagramm.*

Das Diagramm unterstützt den Service com.sun.star.table.TableChart. Die im Wesentlichen interessanten Methoden finden Sie in der Tabelle 234. Mit ihnen können Sie die Einstellungen modifizieren, die Sie beim Anlegen eines neuen Diagramms als Argumente angegeben haben. Ein TableChart-Objekt setzt auch Methoden zum Hinzufügen und Entfernen von Listeners ein.

Tabelle 234. *Die wesentlichen Methoden zur Kontrolle eines Tabellendiagramms.*

Methode	Beschreibung
getEmbeddedObject()	Referenz auf das Diagramm-Modell.
getHasColumnHeaders()	Zeigt, ob die Zellen der obersten Zeile der Datenquelle als Spaltenüberschriften verwendet werden.
getHasRowHeaders()	Zeigt, ob die Zellen der ganz linken Spalte der Datenquelle als Zeilenüberschriften verwendet werden.
getName()	Gibt den Diagrammnamen zurück.
getRanges()	Gibt die benutzten Datenquellbereiche zurück (Array).
setHasColumnHeaders(boole)	Legt fest, ob die Zellen der obersten Zeile der Datenquelle als Spaltenüberschriften verwendet werden.

Methode	Beschreibung
setHasRowHeaders(boole)	Legt fest, ob die Zellen der ganz linken Spalte der Datenquelle als Zeilenüberschriften verwendet werden.
setName(name)	Legt den Diagrammnamen fest.
setRanges(ranges)	Legt die benutzten Datenquellbereiche fest (Array).

Das Diagramm-Modell ist von erheblicher Komplexität – Sie erinnern sich, dass es ein eingebettetes Dokument ist. Es gibt mehrere Methoden für den Umgang mit Diagrammtiteln. Das Titelobjekt unterstützt Hintergrund, Bitmaps, Textrotation, Absatzformate und ein Array von formatierten Textobjekten als Repräsentation des Titeltextes. In unserem Beispiel hat der Titel nur ein einziges Textelement, das auf folgende Weise geändert werden kann:

Listing 497. *Mit getTitleObject() den Diagrammtitel setzen.*

```
oChartDoc = oChart.getEmbeddedObject()
Dim txtArray
txtArray = oChartDoc.getTitleObject().getText()
txtArray(0).setString("Neuer Titeltext")
```

Formatierte Strings besitzen Eigenschaften, die das Darstellungsformat festlegen, wie CharColor, CharEmphasis, CharFontName und CharFontStyleName. Ich empfehle die Objektinspizierung, um herauszufinden, was alles unterstützt wird. Das Objekt oChartDoc enthält auch die Methode getTitle(), die den Titel als einfachen String und Textformateigenschaften ähnlich wie oben anbietet. Das von getTitle zurückgegebene Objekt bindet das Interface XShape ein, so dass man mit ihm die Position und Größe ändern kann.

Listing 498. *Mit getTitle() den Diagrammtitel setzen.*

```
oChartDoc.getTitle().String = "eins zwei drei"
```

Auch der Untertitel ist ein Service com.sun.star.comp.chart.Title, so dass alle Methoden, die sich auf den Titel beziehen, auch für den Untertitel gelten. Man erhält ihn mit getSubtitle().

Listing 499. *Den Untertitel setzen.*

```
oChartDoc.getSubTitle().String = "Ich bin der Untertitel"
```

Zur Legende (com.sun.star.comp.chart.Legend) gelangen Sie mit getLegend(). Damit können Sie Schriftfont, Größe, Position und andere Formateigenschaften einstellen. Das Legende-Objekt gestattet allerdings nicht den Zugriff auf den Legendentext.

Zum Ändern des Diagrammtyps gehört, vom Diagrammdokument einen geeigneten Typ anzufordern und ihn dann einzurichten (s. Listing 495). Sie können an dieser Stelle auch das Makro TypesDocCanCreate im Listing 209 nutzen.

Listing 500. *Auflistung der Services, die ein Diagrammdokument erstellen kann.*

```
Dim oWriteDoc
Dim oText
Dim sn
oWriteDoc = StarDesktop.loadComponentFromURL("private:factory/swriter", _
                                              "_blank", 0, Array())
oText = oWriteDoc.getText()
sn = oChartDoc.getAvailableServiceNames()
MsgBox Join(sn, Chr$(10))
oText.insertString (oText.End, Join(sn, Chr$(10)), False)
```

Von LO 5.3 wird folgende Liste ausgewiesen:

1. com.sun.star.chart.AreaDiagram
2. com.sun.star.chart.BarDiagram

3. com.sun.star.chart.BubbleDiagram
4. com.sun.star.chart.DonutDiagram
5. com.sun.star.chart.FilledNetDiagram
6. com.sun.star.chart.GL3DBarDiagram
7. com.sun.star.chart.LineDiagram
8. com.sun.star.chart.NetDiagram
9. com.sun.star.chart.PieDiagram
10. com.sun.star.chart.StockDiagram
11. com.sun.star.chart.XYDiagram
12. com.sun.star.document.ExportGraphicObjectResolver
13. com.sun.star.document.ImportGraphicObjectResolver
14. com.sun.star.drawing.BitmapTable
15. com.sun.star.drawing.DashTable
16. com.sun.star.drawing.GradientTable
17. com.sun.star.drawing.HatchTable
18. com.sun.star.drawing.MarkerTable
19. com.sun.star.drawing.TransparencyGradientTable
20. com.sun.star.xml.NamespaceMap

Die ersten 11 sind Diagrammtypen. Den Diagrammtyp zu ändern ist trivial.

Listing 501. *Ein Kreisdiagramm als Diagrammtyp.*

```
oChartDoc = oChart.getEmbeddedObject()
oDiagram = oChartDoc.createInstance("com.sun.star.chart.PieDiagram")
oChartDoc.setDiagram(oDiagram)
```

Die Diagrammfläche (com.sun.star.comp.chart.Area) erhält man mit getArea(). Über das Area-Objekt gestalten Sie solche Darstellungseigenschaften wie Farben, Größe und Position.

Die eigentliche Darstellung gehört zum Diagrammtyp, zu erreichen mit getDiagram(). Um ein Gefühl für einige der vom Diagrammtyp-Objekt unterstützten Eigenschaften zu erhalten, schauen Sie sich Tabelle 235 an. Es kommen noch zahlreiche andere Eigenschaften und Methoden hinzu, die ich aus Platzgründen weggelassen habe. Eine Objektinspizierung wird Ihnen die Eigenschaften der Achse, der Farben, der Bitmaps und der Gitter zeigen.

Tabelle 235. *Eigenschaften von Y-Fehlerbalken.*

Eigenschaft	Beschreibung
ConstantErrorLow	Untere Grenze des Fehlerbereichs einer Datenreihe.
ConstantErrorHigh	Obere Grenze des Fehlerbereichs einer Datenreihe.
ErrorBarStyle	Mit den Konstanten com.sun.star.chart.ErrorBarStyle setzen Sie die Kategorie des Y-Fehlerbalkens: NONE, VARIANCE, STANDARD_DEVIATION, ABSOLUTE, RELATIVE, ERROR_MARGIN, STANDARD_ERROR oder FROM_DATA.
PercentageError	Der Prozentsatz zur Darstellung des Y-Fehlerbalkens.
ErrorMargin	Der Prozentsatz für die Fehlerbandbreite.

Eigenschaft	Beschreibung
ErrorIndicator	Mit der Enumeration com.sun.star.chart.ChartErrorIndicatorTyp legen Sie fest, wie der Fehlerwert angezeigt wird: NONE, TOP_AND_BOTTOM, UPPER oder LOWER.
ErrorBarRangePositive	Adressstring des Zellbereichs für positive Fehlerbalken – unter der Voraussetzung, dass ErrorBarStyle auf FROM_DATA gesetzt ist.
ErrorBarRangeNegative	Adressstring des Zellbereichs für negative Fehlerbalken – unter der Voraussetzung, dass ErrorBarStyle auf FROM_DATA gesetzt ist.
MeanValue	Schalter zur Darstellung von Mittelwertlinien.
RegressionCurves	Mit der Enumeration com.sun.star.chart.ChartRegressionCurveType wird der Typ der Regressionskurven für die Werte der Datenreihen bestimmt: NONE, LINEAR, LOGARITHM, EXPONENTIAL oder POWER.
DataCaption	Mit den Konstanten com.sun.star.chart.ChartDataCaption bestimmen Sie die Form der Datenbeschriftung: NONE, VALUE, PERCENT, TEXT oder SYMBOL. Im Bild 119 ist VALUE gesetzt, so dass an jedem Datenpunkt der numerische Wert angezeigt wird.
DataRowSource	Mit der Enumeration com.sun.star.chart.ChartDataRowSource legen Sie fest, ob die Daten in Zeilen oder Spalten vorliegen: ROWS oder COLUMNS.

15.14. Fazit

Calc-Dokumente sind facettenreich und unterstützen eine breite Vielfalt von Fähigkeiten. Meiner Meinung nach bieten Calc-Dokumente mehr Funktionalitäten als andere OOo-Dokumenttypen. Dieses Kapitel ist daher nur der Einstieg in die Erkundung der wunderbaren Sachen, die Sie mit Calc-Dokumenten anstellen können.

16. Zeichnungs- und Präsentationsdokumente

In diesem Kapitel werden Methoden vorgestellt, den Inhalt von Zeichnungs- und Präsentationsdokumenten zu bearbeiten und zu modifizieren. Die Zeichnungsfunktionalitäten sind zwar für Draw und Impress identisch, doch Impress enthält zusätzliches Potenzial speziell für Präsentationen.

Draw und Impress sind vektorbasierte graphische Anwendungen. Sie können zwar auch Bitmaps darstellen, ihre Stärke liegt jedoch nicht gerade in der Fotobearbeitung. In vektorbasierten Anwendungen werden viele grafische Darstellungen als Objekte und nicht als gerasterte Bilder repräsentiert. Linien, Kreise, Rechtecke und Text zum Beispiel sind jeweils spezielle Objekte. Einer der Vorteile der Vektorgrafik liegt darin, dass man mehrere übereinander liegende Elemente unabhängig voneinander bearbeiten und umbilden kann, ohne sich um die Auflösung oder die Pixel kümmern zu müssen.

Grafikprogramme zur Fotobearbeitung sind im allgemeinen zur Darstellung und Gestaltung von Bildern als Bitmaps befasst. Höhe und Breite eines Bitmap-Bildes werden in Pixel angegeben. Laut Wikipedia ist ein Pixel „ein Kunstwort aus den Abkürzungen der englischen Wörter pictures (umgangssprachlich verkürzt ‚pix') und element". Es repräsentiert einen einzelnen, farbigen Punkt im Bild. Die grafischen Fähigkeiten von OpenOffice beschränken sich jedoch auf Vektoroperationen.

Jedes Draw-Dokument unterstützt den Service com.sun.star.drawing.DrawingDocument, und jedes Impress-Dokument unterstützt den Service com.sun.star.presentation.PresentationDocument. Wenn ich ein benutzerfreundliches Makro schreibe, das einen bestimmten Dokumenttyp benötigt, vergewissere ich mich über die Objektmethode supportsService, ob das Dokument vom richtigen Typ ist (s. Listing 502).

Listing 502. *Erst testen, ob das Dokument ein Impress-Dokument ist, danach, ob es ein Draw-Dokument ist.*

```
REM Falls es wichtig ist, sollten Sie den Dokumenttyp überprüfen,
REM um einen Laufzeitfehler zu vermeiden.
sDraw$ = "com.sun.star.drawing.DrawingDocument"
sImpress$ = "com.sun.star.presentation.PresentationDocument"
If ThisComponent.supportsService(sImpress$) Then
  MsgBox "Das aktuelle Dokument ist ein Impress-Dokument", 0, "Impress-Dokument"
ElseIf ThisComponent.supportsService(sDraw$) Then
  MsgBox "Das aktuelle Dokument ist ein Draw-Dokument", 0, "Draw-Dokument"
Else
  MsgBox "Das aktuelle Dokument ist nicht der richtige Typ", 48, "Fehler"
  Exit Sub
End If
```

Achtung Der Service PresentationDocument bindet den Service DrawingDocument ein. Das bedeutet, dass jedes Präsentationsdokument wie ein Zeichnungsdokument aussieht. Um zwischen den beiden Dokumenttypen zu unterscheiden, müssen Sie zuerst auf Vorhandensein eines Präsentations-(Impress-)Dokuments testen, erst danach auf Vorhandensein eines Zeichnungs-(Draw-)Dokuments.

Folgende einfache Funktion gibt ein neues Dokument zurück. Sie wird in anderen Makros in diesem Kapitel verwendet.

Listing 503. *Erstellt ein neues Dokument.*

```
REM Erstellt ein neues leeres Dokument.
REM Mögliche Dokumenttypen: scalc, swriter, sdraw, smath, und simpress.
Function LoadEmptyDocument(docType$)
  Dim noArgs()          'Ein leeres Array für die Argumente.
  Dim sURL As String    'URL des zu öffnenden Dokuments.
  sURL = "private:factory/" & docType
```

```
    LoadEmptyDocument = StarDesktop.loadComponentFromUrl(sURL, "_blank", 0, noArgs())
End Function
```

16.1. Draw-Seiten

Hauptsächlich bestehen Draw- und Impress-Dokumente aus grafischen Daten, die in Draw-Seiten, auch Folien genannt, gespeichert sind. Die zentrale Folienfunktionalität wird von dem Service GenericDrawPage eingebunden. Es gibt zwei Folientypen: MasterPage und DrawPage. Beide Folientypen binden den Service GenericDrawPage ein und können daher denselben Bestand enthalten und bearbeiten.

Eine Masterfolie wirkt als gemeinsamer Hintergrund für null oder mehr normale Folien. Jede normale Folie kann mit einer Masterfolie verknüpft sein. Jede Masterfolie hat folgende Beschränkungen:

- Eine Masterfolie darf im Gegensatz zu einer normalen Folie nicht mit einer Masterfolie verknüpft sein.
- Eine Masterfolie darf nicht aus einem Dokument entfernt werden, wenn noch irgendeine normale Folie mit ihr verknüpft ist.
- Änderungen an einer Masterfolie sind unmittelbar auf jeder normalen Folie sichtbar, die mit dieser Masterfolie verknüpft ist.

Die Methode getMasterPages() gibt die Liste der Masterfolien des Dokuments zurück. Die Methode getDrawPages() gibt die Liste der normalen Folien des Dokuments zurück. Beide Methoden geben denselben Objekttyp zurück. Der Unterschied liegt nur darin, wie die Inhalte verwendet werden (s. Tabelle 236).

Tabelle 236. *Methoden im Interface com.sun.star.drawing.XDrawPages.*

Methode	Beschreibung
insertNewByIndex(Long)	Erzeugt eine neue Folie und fügt sie an der angegebenen Stelle ein.
hasByName(String)	Gibt True zurück, wenn die genannte Folie existiert.
hasElements()	Gibt True zurück, wenn überhaupt eine Folie existiert.
remove(DrawPage)	Entfernt eine bestimmte Folie.
getCount()	Gibt die Anzahl der enthaltenen Objekte als Long Integer zurück.
getByIndex(Long)	Gibt die Folie anhand ihres Indexwerts zurück.
getByName(String)	Gibt die Folie anhand ihres Namens zurück.
duplicate(DrawPage)	Dupliziert die Folie und gibt die neue Folie zurück.

Jede Folie hat einen Namen, den man mit der Methode getName() holen bzw. mit der Methode setName() vergeben kann. Die Masterfolie einer Folie, die selbst keine Masterfolie ist, wird mit getMasterPage() geholt bzw. mit setMasterPage() festgelegt. Listing 504 zeigt die Auflistung der Folien des Dokuments und die zu ihnen gehörenden Masterfolien.

Listing 504. *Ausgabe einer Liste von Foliennamen mit der jeweils zugehörigen Masterfolie.*

```
Sub GetPages()
  Dim s$
  Dim oDoc

  oDoc = LoadEmptyDocument("simpress")
  s = s & GetPagesInfo(oDoc.getDrawPages(), "normale Folien") & Chr$(10)
  s = s & GetPagesInfo(oDoc.getMasterPages(), "Masterfolien")
  MsgBox s, 0, "Folien"
End Sub

Function GetPagesInfo(oDPages, sType$) As String
  REM oDPages: entweder die Liste der normalen oder der Master-Folien
```

```
  REM sType: Typ der Folien im Plural
  Dim i%, s$
  Dim oDPage         'DrawPage
  Dim oMPage         'MasterPage
  Dim sTypeNumerus$ 'Ausgabestring für den Typ der Folien, Singular oder Plural
  Dim iPagesCount%  'Anzahl der Folien
  iPagesCount = oDPages.getCount()
  sTypeNumerus = sType
  If iPagesCount = 1 Then sTypeNumerus = Left(sType, Len(sType) - 1) 'Singular

  s = "*** Es gibt " & iPagesCount & " " & sTypeNumerus & Chr$(10)
  For i = 0 To iPagesCount - 1
    oDPage = oDPages.getByIndex(i)
    s = s & "Folie " & i & " = '" & oDPage.getName() & "'"
    REM Im Fall einer normalen Folie wird zusätzlich der Name
    REM der verknüpften Masterfolie ausgegeben.
    If Not oDPage.supportsService("com.sun.star.drawing.MasterPage") Then
      oMPage = oDPage.getMasterPage()
      s = s & " : Master = "
      If Not IsNull(oMPage) And Not IsEmpty(oMPage) Then
        s = s & "'" & oMPage.getName() & "'"
      End If
    End If
    s = s & Chr$(10)
  Next
  GetPagesInfo = s
End Function
```

Sie können eine Folie zwar mit ihrem Namen ansprechen, allerdings können mehrere Folien mit demselben Namen vorhanden sein. Wenn mehrere Folien denselben Namen verwenden und Sie die Folie mit diesem Namen ansprechen, können Sie sich nicht sicher sein, welche Folie zurückgegeben wird. Das Makro im Listing 505, das nach einer Folie mit einem bestimmten Namen sucht, wird vielfach in diesem Kapitel verwendet.

Listing 505. *Erstellt eine neue Folie mit einem Namen, der noch nicht vorhanden ist.*

```
Function CreateDrawPage(oDoc, sName$, bForceNew As Boolean) As Variant
  REM bForceNew = True: eine unter dem Namen vorhandene Folie soll gelöscht werden.
  Dim oPages 'Alle normalen Folien
  Dim oPage  'Eine einzelne Folie
  Dim i%     'Indexvariable
  oPages = oDoc.getDrawPages()
  If oPages.hasByName(sName) Then
    REM Wenn wir auf jeden Fall eine neue Folie wünschen,
    REM dann wird die Folie gelöscht.
    If bForceNew Then
      oPages.remove(oPages.getByName(sName))
    Else
      REM Ein neue Folie wird nicht benötigt, also wird die gefundene Folie
      REM zurückgegeben und die Funktion verlassen.
      CreateDrawPage = oPages.getByName(sName)
      Exit Function
    End If
  End If

  REM Entweder gibt es die Folie noch nicht oder sie wurde gefunden und entfernt.
  REM Eine neue Folie wird erzeugt, mit Namen versehen und zurückgegeben.
```

```
    oPages.insertNewByIndex(oPages.getCount())
    oPage = oPages.getByIndex(oPages.getCount() - 1)
    oPage.setName(sName)
    CreateDrawPage = oPage
End Function
```

16.1.1. Die eigentliche Folienseite

Sowohl normale Folien als auch Masterfolien unterstützen den Service GenericDrawPage. Er stellt, wie der Name schon andeutet, die grundlegende Zeichnungsfunktionalität zur Verfügung. In den Komponenten Writer und Calc sorgen spezielle Vorlagen für die Formatierung ganzer Seiten. Draw nutzt jedoch die in der Tabelle 237 genannten Eigenschaften.

Tabelle 237. *Eigenschaften im Service com.sun.star.drawing.GenericDrawPage.*

Eigenschaft	Beschreibung
BorderBottom	Unterer Rand in 1/100 mm, als Long Integer.
BorderLeft	Linker Rand in 1/100 mm, als Long Integer.
BorderRight	Rechter Rand in 1/100 mm, als Long Integer.
BorderTop	Oberer Rand in 1/100 mm, als Long Integer.
Height	Höhe in 1/100 mm, als Long Integer.
IsBackgroundDark	True = Die durchschnittliche Leuchtkraft der Hintergrundfüllfarbe liegt unter einem festgelegten Schwellwert.
NavigationOrder	Liste der Zeichnungsobjekte der obersten Ebene der Folie in der Reihenfolge, mit der sie mit der Tab-Taste ausgewählt werden: für den indexierten Zugriff. Als Standard ist die Indexreihenfolge der Folie selbst eingestellt, wodurch die Z-Ordnung zur Standard-Navigationsordnung für Zeichnungsobjekte der obersten Ebene wird, also referenziert oPage.NavigationOrder.getByIndex(2) dieselbe Form wie oPage.getByIndex(2).
Number	Foliennummer als Short Integer. Nur lesbar, startet mit 1 für die erste Seite.
Orientation	Seitenausrichtung als Enumeration com.sun.star.view.PaperOrientation: PORTRAIT (Hochformat) und LANDSCAPE (Querformat).
UserDefinedAttributes	Benutzerdefinierte XML-Attribute.
Width	Breite in 1/100 mm, als Long Integer.

Der Hauptzweck einer Zeichnungsfolie besteht darin, Zeichnungsobjekte (Formen) aufzunehmen. Mit der Methode addShape(Shape) fügt man einem Dokument eine Form hinzu, und mit removeShape(Shape) wird sie entfernt. Bevor eine Form hinzugefügt werden kann, muss sie zuerst vom Dokument erzeugt werden. Jede der vom Listing 506 produzierten und im Bild 120 dargestellten Linien ist eine eigenständige, unverbundene Form. Jede Form kann unabhängig von anderen umgestaltet werden.

Listing 506. *Zeichnet 21 Linien in einem Draw- oder Impress-Dokument.*

```
Function Draw21Lines()
  Dim oPage  'Folie für die Zeichnung
  Dim oShape 'Einzufügende Form
  Dim oPoint 'Startpunkt der Linie
  Dim oSize  'Höhe und Breite der Linie
  Dim i%     'Indexvariable
  Dim n%     'Anzahl der Iterationen
  Dim nShift%'Schiebt die Grafik nach unten
  Dim oDoc

  oDoc = LoadEmptyDocument("sdraw")
  Draw21Lines = oDoc
  oPage = CreateDrawPage(oDoc, "Draw-Test", True)
```

```
    nShift = oPage.Height / 4
    n = 20
    For i = 0 To n
      'Erzeugt eine Linie.
      oShape = oDoc.createInstance("com.sun.star.drawing.LineShape")
      oShape.LineColor = RGB(255, 0, i + 20)        'Farbe
      oShape.LineWidth = 20                         'Breite
      'Kopie des Position-Structs.
      'Es positioniert die linke obere Ecke des Rechtecks, das die Linie umschreibt.
      oPoint = oShape.Position
      oPoint.X = oPage.Width / 4                    'Abstand von links: 1/4 der Breite
      oPoint.Y = i * oPage.Height / n / 4 + nShift  'Abstand von oben
      'Kopiert das Struct zurück in die oShape-Eigenschaft Position.
      oShape.Position = oPoint
      'Kopie des Size-Structs. Es ist das Rechteck, das die Linie umschreibt.
      oSize = oShape.Size
      oSize.Height = (oPage.Height - 2 * i * oPage.Height / n) / 4  'Höhe
      oSize.Width = oPage.Width / 2                                 'Breite
      'Kopiert das Struct zurück in die oShape-Eigenschaft Size.
      oShape.Size = oSize
      'Fügt die Linie in die Folie ein.
      oPage.add(oShape)
    Next

    'Aktiviert die Folie.
    oDoc.CurrentController.setCurrentPage(oPage)
End Function
```

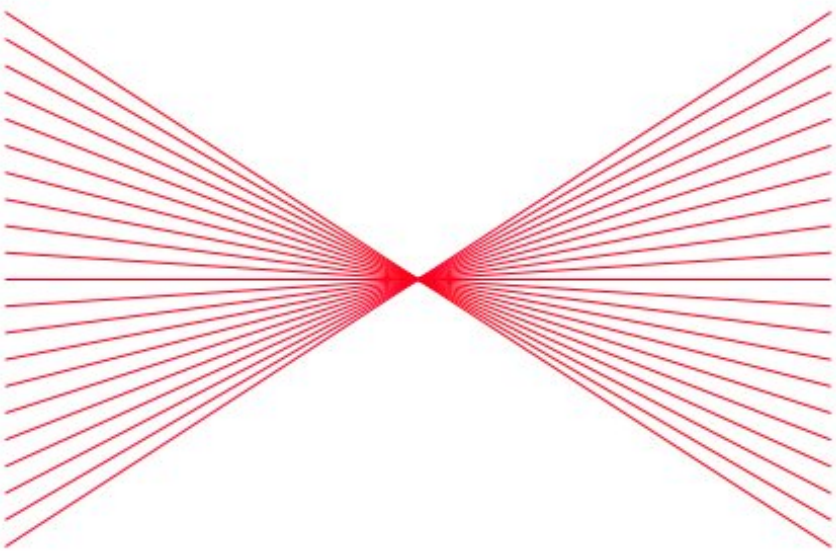


Bild 120. *Einundzwanzig Linien in einem Draw-Dokument.*

Das Makro im Listing 506 funktioniert sowohl mit Draw als auch mit Impress. Die 21 roten Linien werden auf einer Folie mit dem Namen „Draw-Test“ gezeichnet. Diese Folie wird neu erzeugt, falls sie noch nicht existiert.

16.1.2. Auf Formen zugreifen (A. Heier)

Es gibt zwei Möglichkeiten, auf Formen zuzugreifen: über den Index und den Namen. Grundsätzlich können Sie nur über den Index auf eine Form zugreifen. Erst im zweiten Schritt können Sie den Namen bei jedem indizierten Zeichnungsobjekt vergleichen und so eine bestimmte Form identifizieren.

Achtung Leider ist die Identifikation einer Form nur über den Index nicht so einfach. Sie können per Makro oder per Hand angelegte Formen so nicht sicher unterscheiden. Um das Risiko zu verringern, können Sie einen eindeutigen Namen vergeben oder andere Eigenschaften einer Form so parametrieren, dass diese eindeutig zu bestimmen ist. Dazu müssen Sie jedoch dann über alle Formen einer Seite per Schleife prüfen, ob die Kriterien erfüllt sind. Es kann jedoch immer noch das Risiko einer doppelt vorkommenden gleichen Benennung geben.

Der folgende Code in Listing 507 fügt in ein Draw-Dokument auf der ersten Folie zwei Rechtecke ein, wie auf Bild 121 gezeigt. Es werden darin die beiden Routinen aus Listing 515 benutzt.

Listing 507. *Zeichnet zwei Rechtecke auf eine Folienseite und benennt das zweite.*

```
Sub RectangleShapes
  Dim oDoc, oPage, oRectangle

  oDoc = ThisComponent
  oPage = oDoc.DrawPages(0)
  For i = 0 To 1
    oRectangle = oDoc.createInstance("com.sun.star.drawing.RectangleShape")
    With oRectangle
      'Maßeinheit 1/100 mm
      .Position = CreatePoint(2000, 2000+(i*2000))
      .Size = CreateSize(1200, 1200)
      'um die unterschiedlichen Namen im Navigator zu zeigen,
      'wird das zweite Rechteck hier benannt.
      If i = 1 Then
        .Name = "Rechteck_" & i
      End If
    End With
    oPage.add(oRectangle)
  Next
End Sub
```

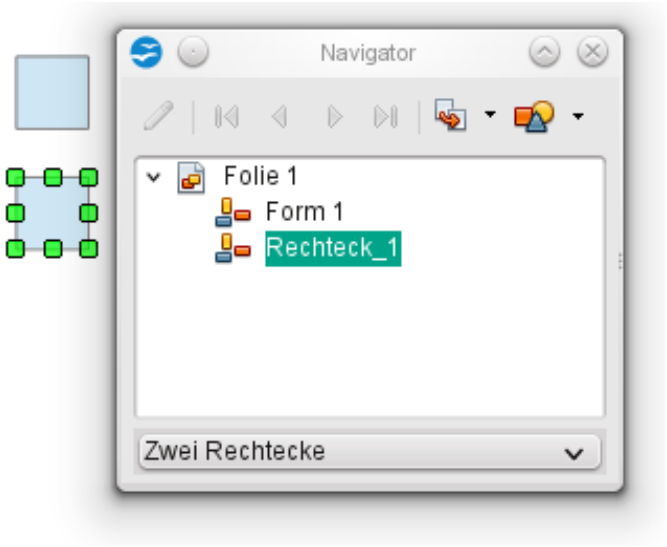


Bild 121. *Individuell benannte Form.*

Sie können nachträglich per Hand Formen individuell benennen (über das Menü **Ändern| Name…**, bzw. Kontextmenü **Name…**). Das Bild 121 zeigt die benannte Form im Navigator. Weiter lässt sich natürlich das vorher erzeugte Rechteck wieder über seinen Namen identifizieren.

Listing 508. *Eine Form über den Namen identifizieren.*

```
Sub GetRectangle(ByVal sName As String)
  Dim oDoc, oPage, oShape
  Dim nNumOfShapes As Integer   'Anzahl der in der Folie enthaltenen Formen
  Dim i As Integer

  oDoc = ThisComponent
  oPage = oDoc.DrawPages(0)
  nNumOfShapes = oPage.getCount()
  For i = 0 To nNumOfShapes - 1
    'das Shape aus dem Container durch den Index festlegen
    oShape = oPage.getByIndex(i)
    'den Namen mit dem übergebenen Parameter vergleichen
    If oShape.Name = "Rechteck_1" Then
      MsgBox oShape.getName() 'das gleiche wie oShape.Name
    End If
  Next
End Sub
```

16.1.3. Formen kombinieren

Das Makro im Listing 506 erzeugt 21 unabhängige Linien. Man kann Formen gruppieren und sie so als einzelne Form gestalten. Die Methode group(XShapes) akzeptiert eine Kollektion von Formen und macht aus ihnen eine einzelne Gruppe: ein XShapeObject entsteht. Am Anfang des Makros im Listing 509 steht der Aufruf von Listing 506. Danach werden alle Linien zu einer Gruppe zusammengefasst.

Listing 509. *Gruppierung vieler Formen zu einem einzigen Objekt.*

```
Sub GroupShapes
  Dim oDoc    'Das Dokument mit 21 Linien
  Dim oPage   'Zeichnungsfolie
  Dim oShapes 'Container für Formen
  Dim i%      'Indexvariable

  REM Erzeugt das Dokument mit den Formen.
  oDoc = Draw21Lines()
  oPage = oDoc.getDrawPages().getByName("Draw-Test")

  REM Erzeugt den Container und fügt alle vorhandenen Formen hinzu.
  oShapes = CreateUnoService("com.sun.star.drawing.ShapeCollection")
  For i = 0 To oPage.getCount() - 1
    oShapes.add(oPage.getByIndex(i))
  Next
  'Gruppiert die Formen, die im Container sind.
  oPage.group(oShapes)
End Sub
```

Werden mehrere Formen mit der Methode group() gruppiert, wird der Folie die gesamte Gruppe als eine einzige Form hinzugefügt. Sie können sie über oPage.getByIndex() ansprechen. Es ist nicht mehr möglich, eine einzelne Linie auszuwählen und gesondert zu bearbeiten. Man kann mit der Methode ungroup(XShapeObject) eine Gruppe aber auch wieder in unabhängige Formen zurückverwan-

deln. Diese Methode entfernt die Objekte aus der Gruppe und fügt sie wieder als individuelle Objekte in die Folie ein (s. Listing 510).

Listing 510. *Wandelt eine Zeichnungsgruppe zurück in einzelne Formen.*

```
Sub UnGroupShapes
  Dim oPage   'Zeichnungsfolie
  Dim oShape  'Einzelne Form
  Dim i%      'Indexvariable

  oPage = ThisComponent.getDrawPages().getByName("Draw-Test")
  For i = 0 To oPage.getCount() - 1
    oShape = oPage.getByIndex(i)
    If oShape.supportsService("com.sun.star.drawing.GroupShape") Then
      oPage.ungroup(oShape)
    End If
  Next
End Sub
```

Obwohl zu einer Gruppe zusammengefasste Formen als ein einziges Objekt bearbeitet werden, sind sie dennoch eine Sammlung von Formen. Im Gegensatz dazu konvertiert die Methode combine(XShapes) jede Form in ein Polygon (PolyPolygonShape) und setzt dann alle zu einem einzigen Polygon zusammen. Die neue Form wird in die Folie eingefügt, daraufhin werden die ursprünglichen Formen entfernt und gelöscht. Die Methode split(XShape) konvertiert die Form in ein Polygon (falls sie es noch nicht ist), und dann wird die Form in mehrere Formen des Typs PolyPolygonShape aufgesplittet. Die neuen Formen werden in die Folie eingefügt, daraufhin wird die ursprüngliche Form entfernt und gelöscht.

Listing 511. *Kombiniert alle Formen zu einer ShapeCollection.*

```
Sub CombineShapes
  Dim oDoc      'Das Dokument mit 21 Linien
  Dim oPage     'Folie
  Dim oShapes   'Container für Formen
  Dim i%        'Indexvariable

  REM Erzeugt das Dokument mit den Formen.
  oDoc = Draw21Lines()
  oPage = oDoc.getDrawPages().getByName("Draw-Test")

  'Erzeugt den Container und fügt alle vorhandenen Formen hinzu.
  oShapes = CreateUnoService("com.sun.star.drawing.ShapeCollection")
  For i = 0 To oPage.getCount() - 1
    oShapes.add(oPage.getByIndex(i))
  Next
  'Kombiniert die Formen, die im Container sind.
  oPage.combine(oShapes)
End Sub
```

Eine Kombination von Formen verändert nicht das Erscheinungsbild gegenüber den einzelnen Formen. Somit ist die Darstellung aus dem Listing 511 dieselbe wie aus dem Listing 506. Beim Verbinden von Formen, wie im Listing 512 zu sehen ist, werden die Linien jedoch mit einer Bézierkurve zusammengefügt.

Listing 512. *Verbindet Formen.*

```
Sub BindShapes()
  Dim oDoc      'Das Dokument mit 21 Linien
  Dim oPage     'Folie
  Dim oShapes   'Container für Formen
  Dim i%        'Indexvariable
```

```
    REM Erzeugt das Dokument mit den Formen.
    oDoc = Draw21Lines()
    oPage = oDoc.getDrawPages().getByName("Draw-Test")

    'Erzeugt den Container und fügt alle vorhandenen Formen hinzu.
    oShapes = CreateUnoService("com.sun.star.drawing.ShapeCollection")
    For i = 0 To oPage.getCount() - 1
      oShapes.add(oPage.getByIndex(i))
    Next
    'Verbindet die Formen.
    oPage.bind(oShapes)
End Sub
```

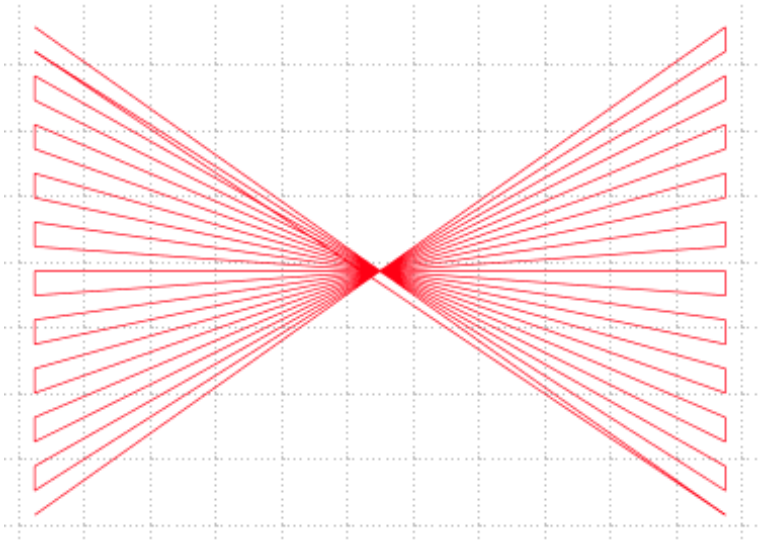


Bild 122. *Bind verbindet die Linien mit einer Bézierkurve.*

Die Methode unbind(XShape) wandelt die Form um in ein Polygon (PolyPolygonShape) (falls sie es nicht schon ist), und dann wird jedes Liniensegment in ein neues Polygon konvertiert. Die ursprüngliche Form wird aus der Folie entfernt und gelöscht. Die neu erzeugte Form verbindet 21 Linien zu einer einzigen Form. Wenn man auf diese Form unbind() anwendet, ändert sich das optische Ergebnis nicht, aber jede Linie ist nun eine eigene Form, es sind also nun 41 Formen entstanden: die ursprünglichen 21 Formen und weitere 20 Linien, mit denen die ursprünglichen 21 verbunden sind.

Tipp	Die Methoden group() und ungroup() wirken wie ein gegenseitiges „Rückgängig“. Die Methoden bind() und combine() sind jedoch kein „Rückgängig“ für unbind() und split(), hauptsächlich weil jede Form in ein Polygon (PolyPolygonShape) umgewandelt wird.

16.1.4. Z-Ordnung (V. Lenhardt)

Alles auf dem Bildschirm ist zweidimensional, alles hat eine Breite und eine Höhe. Jeder in eine Folie eingefügten Form ist eine bestimmte Fläche an einer bestimmten Stelle der Folie zugewiesen. Da in einer Folie aber mehrere Formen existieren können, ist es möglich, dass sich deren Flächen teilweise überlappen. Die Formen werden grafisch nacheinander auf den Bildschirm gezeichnet, so dass früher gezeichnete von später gezeichneten verdeckt sein können. Es ist wie bei einem 3D-Drucker. Eine dritte Dimension kommt ins Spiel. Im mathematischen Koordinatensystem werden zwei Dimensionen durch die Achsen x und y dargestellt. Die dritte Dimension wird üblicherweise durch die z-Achse wiedergegeben, die lotrecht auf den beiden anderen steht.

Die Z-Ordnung ist also die Anordnung der Formen in der Reihenfolge, wie sie gezeichnet werden, beginnend ganz unten mit 0. Jede Form besitzt die Eigenschaft ZOrder, ein Long-Integer als Indexwert für die Position in dieser Reihe. Dieser Wert kann geändert werden. Das Folienobjekt reagiert direkt darauf und passt die ZOrder-Werte der anderen Formen entsprechend an. Damit werden auch die Indexzuordnungen geändert.

```
Dim oDoc, oPage, oShape1, oShape2, oShape3
oPage = oDoc.DrawPages(0)           'Die erste Folie enthält 3 Formen
oShape1 = oPage.getByIndex(0)
oShape2 = oPage.getByIndex(1)
oShape3 = oPage.getByIndex(2)
Print oShape1.ZOrder, oShape2.ZOrder, oShape3.ZOrder          '0  1  2
oPage.getByIndex(1)                 'oShape2
oShape2.ZOrder = 0                  'Ganz nach hinten
oPage.getByIndex(1)                 'oShape1
```

Manchmal möchten Sie die Z-Ordnung der Formen ändern. Im GUI geht das über das Menü **Ändern | Anordnung**. Sie haben die Auswahl „Ganz nach vorn“, „Nach vorn“, „Nach hinten“, „Ganz nach hinten“, „Vor das Objekt“ und „Hinter das Objekt“. In der API gab es einmal diese Optionen im Interface com.sun.star.drawing.XShapeArranger. Mittlerweile ist es als veraltet gekennzeichnet.

Änderungen der Z-Ordnung einer Form durch ein Makro sind jedoch ganz simpel.

- **Ganz nach vorn**: ZOrder wird auf den höchsten Indexwert der enthaltenen Formen gesetzt:

  ```
  oShape.ZOrder = oPage.getCount() - 1
  ```

- **Nach vorn**: ZOrder plus 1. Keine Sorge vor dem Überschreiten des höchsten Indexwerts: Er wird ohne Fehlermeldung ignoriert:

  ```
  oShape.ZOrder = oShape.ZOrder + 1
  ```

- **Nach hinten**: ZOrder minus 1. Keine Sorge vor einem negativen Indexwert: Er wird ohne Fehlermeldung ignoriert:

  ```
  oShape.ZOrder = oShape.ZOrder - 1
  ```

- **Ganz nach hinten**: ZOrder 0:

  ```
  oShape.ZOrder = 0
  ```

- **Vor ein Objekt**: Hängt davon ab, welche ZOrder größer ist:

  ```
  If oShape1.ZOrder < oShape2.ZOrder Then
    oShape1.ZOrder = oShape2.ZOrder
  Else
    oShape1.ZOrder = oShape2.ZOrder + 1
  End If
  ```

- **Hinter ein Objekt**: Hängt davon ab, welche ZOrder größer ist:

  ```
  If oShape1.ZOrder > oShape2.ZOrder Then
    oShape1.ZOrder = oShape2.ZOrder
  Else
    oShape1.ZOrder = oShape2.ZOrder - 1
  End If
  ```

- **Zwei Objekte tauschen**: ZOrder zwischenspeichern und über Kreuz neu zuweisen:

  ```
  z1% = oShape1.ZOrder
  z2% = oShape2.ZOrder
  oShape1.ZOrder = z2
  oShape2.ZOrder = z1
  ```

16.2. Formen

Grafischer Inhalt wird als Shape-Objekt realisiert. Shape-Objekte werden vom Dokument erzeugt und dann der Folie hinzugefügt. Starten Sie mit einem Draw-Dokument das Makro TypesDocCanCreate (s. Listing 209), um schnell herauszufinden, welche Objekte ein Draw-Dokument erzeugen kann. Über einen entsprechenden Filter können Sie die Ausgabe auf die Formtypen beschränken.

Listing 513. *Von Draw unterstützte Shape-Services.*

```
Sub SupportedDrawDocShapes
  Dim oDrawDoc
  ' Erzeugt ein temporäres Draw-Dokument.
  oDrawDoc = LoadEmptyDocument("sdraw")
  ' Produziert die Liste.
  TypesDocCanCreate(oDrawDoc, "shape")
  ' Schließt das temporäre Draw-Dokument.
  oDrawDoc.close(True)
End Sub
```

Auf dieselbe Art finden Sie die von Impress unterstützten Formen.

Listing 514. *Von Impress unterstützte Shape-Services.*

```
Sub SupportedImpressDocShapes
  Dim oDrawDoc
  ' Erzeugt ein temporäres Impress-Dokument.
  oDrawDoc = LoadEmptyDocument("simpress")
  ' Produziert die Liste.
  TypesDocCanCreate(oDrawDoc, "shape")
  ' Schließt das temporäre Impress-Dokument.
  oDrawDoc.close(True)
End Sub
```

In der Tabelle 238 finden Sie die allgemeinen Zeichnungsformen. Diejenigen, die spezifisch für ein Präsentationsdokument sind, zeigt die Tabelle 239. Viele der Formen sind entweder nicht „veröffentlicht“ oder nicht dokumentiert. Wenn ein Service (oder ein Interface) nicht veröffentlicht ist, erwarten die Entwickler, dass er in der Zukunft geändert, entfernt oder aus irgendwelchen Gründen für den normalen Gebrauch ungeeignet sein wird. Ein nicht dokumentierter Service bedeutet, dass eine Dokumentation nicht leicht zu finden ist und der Service somit wahrscheinlich nicht veröffentlicht ist. Verwenden Sie undokumentierte und unveröffentlichte Objekte auf Ihre eigene Verantwortung, denn sie könnten zukünftig anders aussehen. Bedauerlicherweise können sogar manche dokumentierten Formen nicht besonders gut dokumentiert sein.

Tipp	Wenn ein Service (oder ein Interface) nicht veröffentlicht ist, erwarten die Entwickler, dass er in der Zukunft geändert, entfernt oder aus irgendwelchen Gründen für den normalen Gebrauch ungeeignet sein wird.

Manche Formtypen werden nicht direkt vom Dokument erzeugt, zum Beispiel das PolyPolygonBezierShape. Leider sind diese Typen auch schlecht dokumentiert. Einige davon können indirekt erzeugt werden. Testweise habe ich aus einem Impress-Dokument eine Instanz eines CalcShape erzeugt, aber das zurückgegebene Objekt unterstützte den Service CalcShape nicht. Ich habe den erzeugten Service in das Impress-Dokument eingefügt, und die Form wurde zu einem OLE2-Service – und unterstützte immer noch nicht den CalcShape-Service.

Tabelle 238. *Draw-Formen (com.sun.star.drawing).*

Formtyp	Beschreibung
AppletShape	Behälter für ein Java-Applet.

Formtyp	Beschreibung
CaptionShape	Rechteckige mehrzeilige Form, die als Beschriftung für einen festen Punkt innerhalb einer Zeichnung dienen kann.
ClosedBezierShape	Eine Reihe von geschlossenen Bézierkurven.
ClosedFreeHandShape	Über das GUI gezeichnete geschlossene Freihandform.
ConnectorShape	Verbinder zwischen Formen oder Klebepunkten.
ControlShape	Steuerelement, zum Beispiel eine Schaltfläche.
CustomShape	Benutzerdefinierte Form. Nicht veröffentlicht.
EllipseShape	Kreis, Ellipse oder Kreisbogen.
FrameShape	Nicht dokumentiert, scheint aber von anderen Dingen als Rahmen genutzt zu werden.
GraphicObjectShape	Behälter für ein grafisches Objekt wie zum Beispiel eine Bitmap. Für Präsentations- und Zeichnungsobjekte gibt es getrennte Typen.
GroupShape	Behälter für mehrere Formen, verhält sich wie eine einzelne Form.
LineShape	Eine einzelne Linie.
MeasureShape	Maßlinie.
MediaShape	Nicht dokumentiert.
OLE2Shape	OLE-Objekt. Für Präsentations- und Zeichnungsobjekte gibt es getrennte Typen.
OpenBezierShape	Eine Reihe von Bézierkurven.
OpenFreeHandShape	Über das GUI gezeichnete Freihandform.
PageShape	Vorschau einer anderen Seite. Für Präsentations- und Zeichnungsobjekte gibt es getrennte Typen.
PluginShape	Behälter für einen nicht direkt unterstützten Medientyp.
PolyLinePathShape	Nicht dokumentiert.
PolyLineShape	Eine Reihe von verbundenen geraden Linien.
PolyPolygonPathShape	Nicht dokumentiert.
PolyPolygonShape	Eine Reihe von geraden Linie mit Verbindung zwischen dem ersten und dem letzten Punkt.
RectangleShape	Rechteck.
Shape3DCubeObject	Nicht dokumentiert.
Shape3DExtrudeObject	Nicht dokumentiert.
Shape3DLatheObject	Nicht dokumentiert.
Shape3DPolygonObject	Nicht dokumentiert.
Shape3DSceneObject	Nicht dokumentiert.
Shape3DSphereObject	Nicht dokumentiert.
TableShape	Tabelle in einem Draw-Dokument.
TextShape	Textbox.

Tabelle 239. *Impress-Formen (com.sun.star.presentation).*

Formtyp	Beschreibung
CalcShape	Calc-Tabellenblatt in einer Folie.
ChartShape	Diagramm in einem Präsentationsdokument.
DateTimeShape	Nicht veröffentlicht.
FooterShape	Nicht veröffentlicht.
GraphicObjectShape	Behälter für ein grafisches Objekt wie zum Beispiel eine Bitmap. Für Präsentations- und Zeichnungsobjekte gibt es getrennte Typen.

Formtyp	Beschreibung
HandoutShape	Handzettel in einem Präsentationsdokument.
HeaderShape	Nicht veröffentlicht.
MediaShape	Nicht dokumentiert.
NotesShape	Textbox (TextShape) für Notizen in einem Präsentationsdokument.
OLE2Shape	OLE-Objekt. Für Präsentations- und Zeichnungsobjekte gibt es getrennte Typen.
OrgChartShape	Nicht dokumentiert.
OutlinerShape	Textbox (TextShape) für Gliederungen in einem Präsentationsdokument.
PageShape	Vorschau einer anderen Seite. Für Präsentations- und Zeichnungsobjekte gibt es getrennte Typen.
SlideNumberShape	Nicht veröffentlicht.
SubtitleShape	Textbox (TextShape) für Untertitel in einem Präsentationsdokument.
TableShape	Tabelle in einem Impress-Dokument.
TitleTextShape	Textbox (TextShape) für Titel in einem Präsentationsdokument.

16.2.1. Gemeinsame Attribute

Die Lage des eine Form umschreibenden Rechtecks wird in einem Struct com.sun.star.awt.Point gespeichert. Es enthält zwei Long-Integer-Werte, X und Y, für die linke obere Ecke in der Maßeinheit 1/100 mm. Die Flächengröße des eine Form umschreibenden Rechtecks findet sich im Struct com.sun.star.awt.Size, mit den Long-Integer-Werten Width und Height, ebenso in der Maßeinheit 1/100 mm.

Tabelle 240. *Von Shape-Objekten unterstützte Methoden.*

Methode	Beschreibung
getPosition()	Gibt die aktuelle Lage einer Form in 1/100 mm zurück.
setPosition(Point)	Legt die Lage einer Form in 1/100 mm fest.
getSize()	Gibt die aktuelle Größe einer Form in 1/100 mm zurück.
setSize(Size)	Legt die Größe einer Form in 1/100 mm fest.
getGluePoints()	Gibt ein Objekt zurück, das einen indexierten Zugriff auf einen Satz von Klebepunkten bietet, die vom Objekt intern verwendet werden. Jeder Klebepunkt ist ein Struct com.sun.star.drawing.GluePoint2 (s. Tabelle 254).
getShapeType()	Der Typ der Form als String.

Makros, die mit Shape-Objekten umgehen, benötigen häufig die Structs Size und Point. Das Listing 515 stellt Ihnen zwei einfache Methoden zur Erstellung und Modifizierung dieser Structs vor.

Listing 515. *Erstellt ein Point- oder Size-Struct und gibt es zurück.*

```
Function CreatePoint(ByVal x As Long,ByVal y As Long) As com.sun.star.awt.Point
  Dim oPoint
  oPoint = CreateUnoStruct("com.sun.star.awt.Point")
  oPoint.X = x : oPoint.Y = y
  CreatePoint = oPoint
End Function

Function CreateSize(ByVal x As Long,ByVal y As Long) As com.sun.star.awt.Size
  Dim oSize
  oSize = CreateUnoStruct("com.sun.star.awt.Size")
  oSize.Width = x : oSize.Height = y
```

```
    CreateSize = oSize
End Function
```

Interfaces definieren Methoden, und sie können von anderen Interfaces abstammen. Services hingegen binden Interfaces und andere Services ein. Sie definieren auch Eigenschaften. Einige Services sind genau dafür konzipiert, eine Gruppe von verwandten Eigenschaften zu definieren. Die vom Service Shape definierten Eigenschaften sind ganz allgemein auf die meisten Formtypen anwendbar (s. Tabelle 241).

Tabelle 241. *Eigenschaften im Service com.sun.star.drawing.Shape.*

Eigenschaft	Version	Beschreibung
ZOrder	AOO/LO	Repräsentiert die Z-Ordnung dieser Form. Die Z-Ordnung kontrolliert die Zeichnungsreihenfolge von Objekten und damit ihre überlappende Anordnung nach vorn oder hinten. Long Integer.
LayerID	AOO/LO	Index der Ebene, zu der die Form gehört. Short Integer.
LayerName	AOO/LO	Name der Ebene, zu der die Form gehört.
Visible	AOO/LO	False = Die Form ist auf dem Bildschirm nicht sichtbar, kann aber druckbar sein, s.unten.
Printable	AOO/LO	True = Die Form erscheint in der Druckausgabe, kann aber auf dem Bildschirm nicht sichtbar sein, s.oben.
MoveProtect	AOO/LO	True = Der Benutzer darf die Form interaktiv nicht bewegen.
Name	AOO/LO	Name der Form als String.
SizeProtect	AOO/LO	True = Der Benutzer darf die Größe der Form interaktiv nicht ändern.
Style	AOO/LO	Formvorlage als Objekt com.sun.star.style.XStyle.
Transformation	AOO/LO	Transformationsmatrix vom Typ com.sun.star.drawing.HomogenMatrix3, die Verschiebung, Rotation, Scherung und Skalierung enthalten kann.
ShapeUserDefinedAttributes	AOO/LO	Speichert und stellt XML-Attribute wieder her aus automatischen Vorlagen in XML-Dateien.
NavigationOrder	AOO/LO	Die Position in der Reihenfolge, in der die Formen mit der Tab-Taste ausgewählt werden. Ein negativer Wert steht für die Position entsprechend der Z-Ordnung.
Hyperlink	LO	Linkadresse.
InteropGrabBag	LO ab 4.2	Liste als Grabbelsack für Eigenschaften von com.sun.star.beans.PropertyValue-Werten für interoperatible Zwecke. Wird nicht vom ODF-Filter berücksichtigt.
RelativeHeight	LO ab 4.3	Relative Höhe des Objekts. Nur als positiver Wert wirksam.
RelativeWidth	LO ab 4.3	Relative Breite des Objekts. Nur als positiver Wert wirksam.
RelativeHeightRelation	LO ab 4.3	Das Verhältnis der relativen Höhe des Objekts. Nur mit einem positiven Wert von RelativeHeight wirksam.
RelativeWidthRelation	LO ab 4.3	Das Verhältnis der relativen Breite des Objekts. Nur mit einem positiven Wert von RelativeWidth wirksam.

OOo definiert besondere Services mit Eigenschaften und Methoden, die speziell für Linien, Text, Schatten, Formrotationen und Flächenfüllungen gelten. Nicht alle Formtypen unterstützen alle diese Services. Beispielsweise ist es für eine Linienform nicht sehr sinnvoll, Eigenschaften und Methoden für Flächenfüllungen zu unterstützen.

Dies sind die gerade erwähnten Services für gemeinsame Eigenschaften:

- com.sun.star.drawing.Text
- com.sun.star.drawing.LineProperties

- com.sun.star.drawing.FillProperties
- com.sun.star.drawing.ShadowProperties
- com.sun.star.drawing.RotationDescriptor

Ich habe ein Makro geschrieben, um auf die Schnelle herauszufinden, welche Formen welche gemeinsamen Services anbieten. Zuerst einmal benötigt man eine Liste der Services, die den Namen „shape“ enthalten.

Listing 516. *Rückgabe einer gefilterten Liste der von einem Objekt unterstützten Services.*

```
Function GetListCanCreate(oObj, nameFilter$)
  Dim allNames   ' Liste aller Namen.
  Dim s$
  Dim i%

  s = ""
  allNames = oObj.getAvailableServiceNames()

  For i = LBound(allNames) To UBound(allNames)
    If (InStr(allNames(i), nameFilter) > 0) Then
      If Len(s) > 0 Then
        s = s & "," & allNames(i)
      Else
        s = allNames(i)
      End If
    End If
  Next
  GetListCanCreate = Split(s, ",")
End Function
```

Das folgende Makro erstellt ein Impress-Dokument und fragt es, welche Services es erzeugen kann, die das Wort „shape“ enthalten. Dann wird jede Form daraufhin untersucht, ob sie die Kategorien unterstützt, die hier von Interesse sind. Leider werden diese Services erst dann von der Form unterstützt, wenn sie in das Dokument eingefügt ist. Daher fügt das Makro jede Form in das Impress-Dokument ein, prüft, welche Services unterstützt werden, und entfernt sie danach wieder.

Tipp Wenn eine Form erzeugt wurde, ist sie noch nicht vollständig mit Leben erfüllt. Das geschieht erst, wenn sie in ein Dokument eingefügt ist.

Listing 517. *Kategorien der Formtypen.*

```
Sub CategorizeShapeTypes
  Dim oSize As New com.sun.star.awt.Size
  Dim oPos As New com.sun.star.awt.Point

  Dim aCategories ' Die Servicenamen von Interesse
  Dim iRow%       ' Iteration über die Datenzeilen
  DIm iName%      ' Iteration über die unterstützten Servicenamen

  Dim oShape     ' Die jeweils neu erzeugte Form
  Dim oPage      ' Folie, in die die Formen eingefügt werden

  Dim oCalc      ' Calc-Dokument für die Ergebnisse
  Dim oImpress   ' Impress-Dokument für die Tests
  Dim oData      ' Die temporären Zellinhalte
  Dim oCellRange ' Zellbereich in Calc für die Ergebnisse
```

```
Dim shapeNames ' Liste der unterstützten Formnamen
Dim oSheet     ' Calc-Tabelle mit den Ergebnissen
Dim oRow       ' Eine Zeile des Arrays oData

REM Erstellt ein Impress- und ein Calc-Dokument.
oImpress = LoadEmptyDocument("simpress")
oCalc = LoadEmptyDocument("scalc")

REM Holt die Liste der unterstützten Formnamen.
shapeNames = GetListCanCreate(oImpress, "shape")

REM Verwendet für jede Form dieselbe Größe und Position.
oSize.Width = 6000
oSize.Height = 6100
oPos.X = 6000
oPos.Y = 5000

REM Die erste Folie zur Aufnahme der erzeugten Formen.
oPage = oImpress.DrawPages.getByIndex(0)

REM Die Kategorien von Interesse.
aCategories = Array("com.sun.star.drawing.Text", _
   "com.sun.star.drawing.LineProperties", _
   "com.sun.star.drawing.FillProperties", _
   "com.sun.star.drawing.ShadowProperties", _
   "com.sun.star.drawing.RotationDescriptor")

REM Der Zellbereich für die Ergebnisse.
oSheet = oCalc.getSheets().getByIndex(0)
oCellRange = oSheet.getCellRangeByPosition(0, 0, _
          UBound(aCategories) - LBound(aCategories) + 1, _
          UBound(shapeNames) - LBound(shapeNames) + 1)

oData = oCellRange.getDataArray()

REM Die Zeile mit den Spaltentiteln.
oRow = oData(0)
For iRow = 1 To UBound(oRow)
  oRow(iRow) = aCategories(iRow - 1)
Next

REM Für jeden Formnamen ...
For iName = 0 To UBound(shapeNames)
  REM ... eine eigene Zeile mit dem Namen in der ersten Spalte
  oRow = oData(iName + 1)
  oRow(0) = shapeNames(iName)

  REM Die Form wird erzeugt und der Folie hinzugefügt.
  oShape = oImpress.CreateInstance(shapeNames(iName))
  oPage.add(oShape)
  oShape.setSize(oSize)
  oShape.setPosition(oPos)

  REM Welche Services werden von der aktuellen Form unterstützt?
  For iRow = LBound(aCategories) To UBound(aCategories)
    If oShape.supportsService(aCategories(iRow)) Then
      oRow(iRow + 1) = "X"
    End If
```

```
    Next

    REM Die erzeugte Form wird entsorgt.
    oShape.dispose()
  Next

  REM Fügt die Ergebnisse in das Calc-Dokument ein.
  oCellRange.setDataArray(oData)
End Sub
```

Das Calc-Dokument aus dem Listing 517 ist nicht sortiert, und es werden die vollen Servicenamen verwendet. Die Daten finden Sie in den beiden folgenden Tabellen aufgelistet: die Formen von com.sun.star.drawing in der Tabelle 242 und die von com.sun.star.presentation in der Tabelle 243.

Tabelle 242. *Welche Zeichnungsformen unterstützen welche Services.*

Form	Text	Line Properties	Fill Properties	Shadow Properties	Rotation Descriptor
AppletShape					
CaptionShape	X	X	X	X	X
ClosedBezierShape	X	X	X	X	X
ClosedFreeHandShape	X	X	X	X	X
ConnectorShape	X	X		X	X
ControlShape					
CustomShape	X	X	X		
EllipseShape	X	X	X	X	X
FrameShape					
GraphicObjectShape	X			X	X
GroupShape					
LineShape	X	X		X	X
MeasureShape	X	X		X	X
MediaShape					
OLE2Shape					
OpenBezierShape	X	X	X	X	X
OpenFreeHandShape	X	X	X	X	X
PageShape					
PluginShape					
PolyLinePathShape	X	X		X	X
PolyLineShape	X	X		X	X
PolyPolygonPathShape	X	X	X	X	X
PolyPolygonShape	X	X	X	X	X
RectangleShape	X	X	X	X	X
Shape3DCubeObject					
Shape3DExtrudeObject					
Shape3DLatheObject					
Shape3DPolygonObject					
Shape3DSceneObject					

Form	Text	Line Properties	Fill Properties	Shadow Properties	Rotation Descriptor
Shape3DSphereObject					
TableShape					
TextShape	X	X	X	X	X

Tabelle 243. *Welche Impress-Formen unterstützen welche Services.*

Form	Text	Line Properties	Fill Properties	Shadow Properties	Rotation Descriptor
CalcShape					
ChartShape					
DateTimeShape	X	X	X	X	X
FooterShape	X	X	X	X	X
GraphicObjectShape	X			X	X
HandoutShape					
HeaderShape	X	X	X	X	X
MediaShape					
NotesShape	X	X	X	X	X
OLE2Shape					
OrgChartShape					
OutlinerShape	X	X	X	X	X
PageShape					
SlideNumberShape	X	X	X	X	X
SubtitleShape	X	X	X	X	X
TableShape					
TitleTextShape	X	X	X	X	X

Der Zeichnungsservice Text

Jede Form, die den Service com.sun.star.drawing.Text unterstützt, besitzt die Fähigkeit zur Textgestaltung. Dieser Service unterstützt das Standardinterface com.sun.star.text.XText und einen für Zeichnungen speziellen Satz von Texteigenschaften. Neben den Zeichen- und Absatzeigenschaften definiert der Service Eigenschaften, die speziell für Zeichnungsobjekte konzipiert sind (s. Tabelle 244).

Tabelle 244. *Eigenschaften im Service com.sun.star.drawing.TextProperties.*

Eigenschaft	Beschreibung
IsNumbering	True = Die Nummerierung des Textes in dieser Form ist eingeschaltet.
NumberingRules	Die Nummerierungsebenen als eine Reihe von com.sun.star.style.NumberingRule.
TextAnimationAmount	Schrittweite in Pixel bei jeder Stufe der Lauftextanimation.
TextAnimationCount	Anzahl der Durchläufe der Lauftextanimation.
TextAnimationDelay	Schrittverzögerung zwischen jeder Stufe der Lauftextanimation, in 1/1000 Sekunden.
TextAnimationDirection	Laufrichtung der Lauftextanimation: Enumeration com.sun.star.drawing.TextAnimationDirection: LEFT, RIGHT, UP und DOWN.

Eigenschaft	Beschreibung
TextAnimationKind	Effekt der Lauftextanimation: Enumeration com.sun.star.drawing.TextAnimationKind: • NONE – Kein Effekt. • BLINK – Beständiger Wechsel zwischen sichtbarem und unsichtbarem Text. • SCROLL – Durchlaufen. Scrollt den Text. • ALTERNATE – Hin- und zurücklaufen. • SLIDE – Reinlaufen. Scrollt den Text von einer Seite zur Endposition.
TextAnimationStartInside	True = Der Text ist sichtbar beim Start der Lauftextanimation.
TextAnimationStopInside	True = Der Text ist sichtbar beim Beenden der Lauftextanimation.
TextAutoGrowHeight	True = Die Höhe der Form wird an den Text angepasst.
TextAutoGrowWidth	True = Die Breite der Form wird an den Text angepasst.
TextContourFrame	Konturfluss. True = Der Text wird am Rahmen ausgerichtet..
TextFitToSize	Textgrößenanpassung als Enumeration com.sun.star.drawing.TextFitToSizeType: • NONE – Die Textgröße wird durch die Fontattribute definiert. • PROPORTIONAL – Der Text wird skaliert, wenn die Form skaliert wird. • ALLLINES – Wie PROPORTIONAL, aber auch Skalierung der Breite jeder Zeile. • RESIZEATTR – Bei einer Skalierung der Form werden die Fontattribute skaliert.
TextHorizontalAdjust	Textausrichtung horizontal: Enumeration com.sun.star.drawing.TextHorizontalAdjust: • LEFT – Linksbündig am Rahmen. • CENTER – Zentriert zwischen dem linken und rechten Rand des Rahmens. • RIGHT – Rechtsbündig am Rahmen. • BLOCK – Gestreckt zwischen dem linken und rechten Rand des Rahmens.
TextLeftDistance	Abstand zwischen Text und linkem Rand der Form. Long Integer.
TextLowerDistance	Abstand zwischen Text und unterem Rand der Form. Long Integer.
TextMaximumFrameHeight	Obere Grenze der Höhe einer Form bei automatischer Anpassung durch Texteingabe.
TextMaximumFrameWidth	Obere Grenze der Breite einer Form bei automatischer Anpassung durch Texteingabe.
TextMinimumFrameHeight	Untere Grenze der Höhe einer Form bei automatischer Anpassung durch Texteingabe.
TextMinimumFrameWidth	Untere Grenze der Breite einer Form bei automatischer Anpassung durch Texteingabe.
TextRightDistance	Abstand zwischen Text und rechtem Rand der Form. Long Integer.
TextUpperDistance	Abstand zwischen Text und oberem Rand der Form. Long Integer.
TextVerticalAdjust	Textausrichtung vertikal: Enumeration com.sun.star.drawing.TextVerticalAdjust: • TOP – Mit dem oberen Textrand am oberen Rand des Rahmens. • CENTER – Zentriert zwischen dem oberen und unteren Rand des Rahmens. • BOTTOM – Mit dem unteren Textrand am unteren Rand des Rahmens. • BLOCK – Gestreckt vom oberen zum unteren Rand des Rahmens.
TextWritingMode	Textlaufrichtung: Enumeration com.sun.star.text.TextWritingMode: • LR_TB – Von links nach rechts, zeilenweise von oben nach unten. • RL_TB – Von rechts nach links, zeilenweise von oben nach unten. • TB_RL – Von oben nach unten, zeilenweise von rechts nach links.

Maßlinie

Eine Maßlinie (MeasureShape, s. Bild 123) ist eine Linie mit zugehörigem Text, der die Länge eben dieser Linie anzeigt. Das Makro im Listing 518 erzeugt zwei Maßlinien und ändert den Text der ei-

nen auf „Breite“. Als Beispiel für das Setzen einer Eigenschaft wird in diesem Makro die Eigenschaft TextAnimationKind auf SCROLL (s. Tabelle 244) gesetzt, so dass der Text von rechts nach links läuft, allerdings nur in einem Impress-Dokument.

Listing 518. *Zeichnung einer Maßlinie.*

```
Sub DrawMeasureShape()
  Dim oPage     'Zeichnungsfolie.
  Dim oShape    'Einzufügende Form.
  Dim oStart As New com.sun.star.awt.Point
  Dim oEnd   As New com.sun.star.awt.Point
  Dim oDrawDoc  'Temporäres Draw-Dokument.

  oDrawDoc = LoadEmptyDocument("sdraw")

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.MeasureShape")
  oPage.add(oShape)

  REM Die folgenden Werte müssen gesetzt werden, NACHDEM das Objekt eingefügt wurde.
  oStart.X = oPage.Width / 4    : oEnd.X = oPage.Width / 2
  oStart.Y = oPage.Height / 4   : oEnd.Y = oPage.Height / 4
  oShape.StartPosition = oStart
  oShape.EndPosition = oEnd
  oShape.setString("Breite")
  oShape.TextAnimationKind = com.sun.star.drawing.TextAnimationKind.SCROLL

  oShape = oDrawDoc.createInstance("com.sun.star.drawing.MeasureShape")
  oPage.add(oShape)
  oStart.X = oPage.Width / 5    : oEnd.X = oPage.Width / 5
  oStart.Y = oPage.Height / 4   : oEnd.Y = oPage.Height / 2.5
  oShape.StartPosition = oStart
  oShape.EndPosition = oEnd
  'Aktiviert die Folie.
  oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

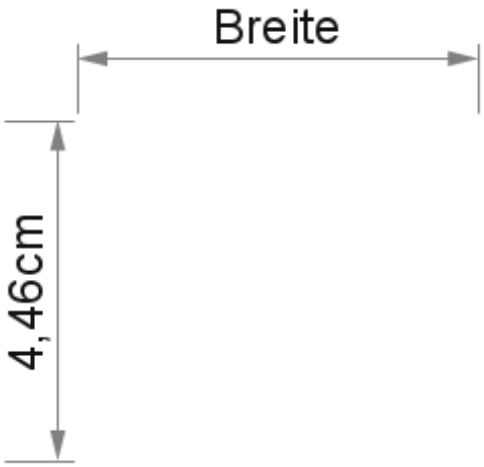


Bild 123. *Standardmäßig zeigen Maßlinien die Größe der Linie an. Sie können das überschreiben.*

Linieneigenschaften

Formen, die den Service com.sun.star.drawing.LineProperties unterstützen, können Einfluss darauf nehmen, wie Linien gezeichnet werden. Die meisten Formen unterstützen Linieneigenschaften, denn

die meisten Formen enthalten irgendwelche Linien. Die Eigenschaften, die für Ausgangs- und Endpunkte von Linien gelten, werden allerdings nur von Formen mit offenen Enden unterstützt.

Tabelle 245. *Eigenschaften im Service com.sun.star.drawing.LineProperties.*

Eigenschaft	Beschreibung
LineColor	Linienfarbe. Long Integer.
LineCap	Die Gestalt des Endes dicker Linien als Enumeration cm.sun.star.drawing.LineCap: BUTT = abruptes Ende ohne Gestaltung ROUND = hinzugefügter Halbkreis SQUARE = hinzugefügtes halbes Quadrat
LineDash	Gestaltung eines Glieds einer durchbrochenen Linie. Enumeration com.sun.star.drawing.LineDash: • Style – Enumeration com.sun.star.drawing.DashStyle: RECT – Rechteckig. ROUND – Rund. RECTRELATIVE – Rechteckig, Größe abhängig von der Linienlänge. ROUNDRELATIVE – Rund, Größe abhängig von der Linienlänge. • Dots – Anzahl der Punkte. Long Integer. • DotLen – Länge eines Punkts. Long Integer. • Dashes – Anzahl der Striche. Long Integer. • DashLen – Länge eines Strichs. Long Integer. • Distance – Abstand zwischen den Punkten/Strichen. Long Integer.
LineDashName	Der Name der durchbrochenen Linie.
LineEnd	Linienende als Polygonkoordinaten in der Form PolyPolygonBezierCoords.
LineEndCenter	True = Die Linie endet in der Mitte des Polygons.
LineEndName	Name des Linienende-Polygons.
LineEndWidth	Breite des Linienende-Polygons.
LineJoint	Eckenstil: Enumeration com.sun.star.drawing.LineJoint: • NONE – Nicht durchgezogener Winkel. • MIDDLE – Winkelform als Mittelwert der Anschlüsse. • BEVEL – Abgeschrägter Winkel. • MITER – Gehrungswinkel. • ROUND – Abgerundeter Winkel.
LineStart	Linienbeginn als Polygonkoordinaten in der Form PolyPolygonBezierCoords.
LineStartCenter	True = Die Linie beginnt in der Mitte des Polygons.
LineStartName	Name des Linienbeginn-Polygons.
LineStartWidth	Breite des Linienbeginn-Polygons.
LineStyle	Linienstil: Enumeration com.sun.star.drawing.LineStyle: • NONE – Die Linie ist unsichtbar. • SOLID – Die Linie ist durchgängig. • DASH – Die Linie besteht aus einer Reihe von Linien und/oder Punkten.
LineTransparence	Transparenz der Linie als prozentuale Angabe: Short Integer.
LineWidth	Linienbreite in 1/100 mm. Long Integer.

Flächenfüllung am Beispiel einer geschlossenen Bézierform

In Formen, die den Service com.sun.star.drawing.FillProperties unterstützen, kann die freie Fläche innerhalb der Form kontrolliert gefüllt werden. Ganz allgemein: wenn die Form geschlossen ist, kann sie gefüllt werden.

Tabelle 246. *Eigenschaften im Service com.sun.star.drawing.FillProperties.*

Eigenschaft	Beschreibung
FillBackground	True = Der transparente Hintergrund einer schraffierten Fläche erhält die aktuelle Hintergrundfarbe.
FillBitmap	Das verwendete Bitmapmuster, wenn FillStyle BITMAP ist.
FillBitmapLogicalSize	Angabe, ob die Größe eines Bitmapmusters prozentual (True) oder absolut (False) angegeben ist.
FillBitmapMode	Modus, wie eine Fläche mit einem Bitmapmuster gefüllt wird. Abhängig vom Wert werden die Eigenschaften FillBitmapStretch und FillBitmapTile gesetzt. Enumeration com.sun.star.drawing.BitmapMode: • REPEAT – Über die gesamte Fläche wiederholt. (FillBitmapStretch = False, FillBitmapTile = True) • STRETCH – Über die gesamte Fläche ausgedehnt. (FillBitmapStretch = True, FillBitmapTile = False) • NO_REPEAT – In originaler oder angegebener Größe. (FillBitmapStretch = False, FillBitmapTile = False
FillBitmapName	Der Name des Bitmapmusters, wenn FillStyle BITMAP ist.
FillBitmapOffsetX	Der horizontale Offset, an dem das Muster startet. Prozentuale Angabe bezogen auf die Breite des Musters.
FillBitmapOffsetY	Der vertikale Offset, an dem das Muster startet. Prozentuale Angabe bezogen auf die Höhe des Musters.
FillBitmapPositionOffsetX	Jede zweite Zeile der Muster wird um den angegebenen Prozentsatz der Musterbreite verschoben.
FillBitmapPositionOffsetY	Jede zweite Zeile der Muster wird um den angegebenen Prozentsatz der Musterhöhe verschoben.
FillBitmapRectanglePoint	Die Position innerhalb des Musters, die als obere linke Ecke in der Form dargestellt wird: Enumeration com.sun.star.drawing.RectanglePoint: LEFT_TOP, MIDDLE_TOP, RIGHT_TOP, LEFT_MIDDLE, MIDDLE_MIDDLE, RIGHT_MIDDLE, LEFT_BOTTOM, MIDDLE_BOTTOM, RIGHT_BOTTOM.
FillBitmapSizeX	Die Breite des Füllmusters, absolut oder prozentual (s. FillBitmapLogicalSize).
FillBitmapSizeY	Die Höhe des Füllmusters, absolut oder prozentual (s. FillBitmapLogicalSize).
FillBitmapURL	Der URL des Musters, wenn FillStyle BITMAP ist.
FillColor	Füllfarbe, wenn FillStyle SOLID ist.
FillGradient	Der Farbverlauf, wenn FillStyle GRADIENT ist: Struct com.sun.star.awt.Gradient (s. Tabelle 247).
FillGradientName	Der Name des Farbverlaufs, wenn FillStyle GRADIENT ist.
FillHatch	Die Schraffur, wenn FillStyle HATCH ist.
FillHatchName	Der Name der Schraffur, wenn FillStyle HATCH ist.

Eigenschaft	Beschreibung
FillStyle	Art der Füllung: Enumeration com.sun.star.drawing.FillStyle: • NONE – Keine Füllung. • SOLID – Farbfüllung. • GRADIENT – Farbverlauf. • HATCH – Schraffur. • BITMAP – Bild als Füllung, zum Beispiel JPG.
FillTransparence	Prozentuale Transparenz, wenn FillStyle SOLID ist.
FillTransparenceGradient	Transparenzverlauf. Struct com.sun.star.awt.Gradient (s. Tabelle 247).
FillTransparenceGradientName	Der Name des Farbverlaufs. Kann auch leer sein.
GraphicCrop [ab LO 4.3]	Beschnitt des grafischen Objekts. Negative Werte schneiden einen Teil des sichtbaren Bereichs ab, positive Werte erweitern den sichtbaren Bereich durch die Hintergrundfarbe. Der gesamte Höhen-/Breite-Beschnitt darf die Abmessungen der Grafik nicht überschreiten. Als Struct com.sun.star.text.GraphicCrop mit den Elementen (jeweils als Long): Top (oben), Bottom (unten), Left (links), Right (rechts).

Farbverläufe werden wie folgt definiert:

Tabelle 247. *Eigenschaften des Structs com.sun.star.awt.Gradient.*

Eigenschaft	Beschreibung
Style	Enumeration com.sun.star.awt.GradientStyle: • LINEAR – Linearer Verlauf. • AXIAL – Axialer Verlauf. • RADIAL – Radialer Verlauf. • ELLIPTICAL – Verlauf in der Form einer Ellipse. • SQUARE – Verlauf in der Form eines Quadrats. • RECT – Verlauf in der Form eines Rechtecks.
StartColor	Startfarbe.
EndColor	Endfarbe.
Angle	Winkel des Verlaufs in 1/10 Grad.
Border	Prozentsatz der gesamten Breite, wo nur die Startfarbe erscheint.
XOffset	X-Koordinate, wo der Verlauf beginnt.
YOffset	Y-Koordinate, wo der Verlauf beginnt.
StartIntensity	Farbintensität beim Start.
EndIntensity	Farbintensität am Ende.
StepCount	Anzahl der Farbänderungen.

Das Makro im Listing 519 zeichnet eine geschlossene Bézierform. Als Füllung ist ein Farbverlauf gewählt, wodurch die Helligkeit der Form kontinuierlich zunimmt. Das Ergebnis ist eine Zeichnung, die schmale Bänder jeder Farbe oder Helligkeitsstufe zeigt. Sie können das Aussehen des Verlaufs mit Hilfe der Eigenschaft FillTransparenceGradient (s. Tabelle 246) glätten.

Listing 519. *Eine geschlossene Bézierkurve mit einer Verlaufsfüllung.*

```
Sub DrawClosedBezierShape
  Dim oPage     'Zeichnungsfolie
  Dim oShape    'Einzufügende Form
```

```
    Dim oCoords   'Koordinaten des einzufügenden Polygons
    Dim oDrawDoc  'Temporäres Draw-Dokument
    Dim oPlgFlags 'Spart Schreibarbeit

    oPlgFlags = com.sun.star.drawing.PolygonFlags
    oDrawDoc = LoadEmptyDocument("sdraw")

    oCoords = CreateUnoStruct("com.sun.star.drawing.PolyPolygonBezierCoords")

    REM Array der Koordinaten. Der erste wie der letzte Punkt
    REM sind normale Punkte. Die mittleren Punkte sind Bézier-Kontrollpunkte.
    oCoords.Coordinates = Array(Array(CreatePoint(1000, 1000),_
                                      CreatePoint(3000, 4000),_
                                      CreatePoint(3000, 4000),_
                                      CreatePoint(5000, 1000)))
    oCoords.Flags = Array(Array(oPlgFlags.NORMAL,_
                                oPlgFlags.CONTROL,_
                                oPlgFlags.CONTROL,_
                                oPlgFlags.NORMAL))

    oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
    oShape = oDrawDoc.createInstance("com.sun.star.drawing.ClosedBezierShape")
    oPage.add(oShape)
    oShape.FillStyle = com.sun.star.drawing.FillStyle.GRADIENT  'Farbverlauf
    oShape.PolyPolygonBezier = oCoords
    'Aktiviert die Folie.
    oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Bild 124. *Bézierform mit Verlaufsfüllung.*

Schatten und Rechteck

Formen, die den Service ShadowProperties unterstützen, können mit einem Schatten gezeichnet werden. Lage und Farbe des Schattens werden über die Eigenschaften in der Tabelle 248 gesteuert.

Tabelle 248. *Eigenschaften im Service com.sun.star.drawing.ShadowProperties.*

Eigenschaft	Beschreibung
Shadow	True = Die Form hat einen Schatten.
ShadowColor	Farbe des Schattens. Long Integer.
ShadowTransparence	Transparenz des Schattens als Prozentsatz.
ShadowXDistance	Horizontaler Abstand des linken Formrandes vom Schatten.
ShadowYDistance	Vertikaler Abstand des oberen Formrandes vom Schatten.

Üblicherweise werden Schatten so gezeichnet, dass zuerst die Form in der Schattenfarbe in einem bestimmten Verschiebeabstand gezeichnet wird und danach die Form selbst (s. Bild 125). Mit dieser Hintergrundinformation sind die Eigenschaften ShadowXDistance und ShadowYDistance als der Abstand erklärlich, um den das „Schattenobjekt“ beim Zeichnen verschoben wird. Die Standardwerte für ShadowXDistance und ShadowYDistance sind positiv, wodurch sich eine Verschiebung nach

rechts und nach unten ergibt. Negative Werte verschieben den Schatten nach links und nach oben. Das Makro im Listing 520 zeichnet zwei Rechtecke, das erste mit einem Standardschatten nach rechts unten, das zweite mit einem Schatten nach links unten (s. Bild 125).

Listing 520. *Rechtecke mit Text, abgerundeten Ecken und Schatten.*

```
Sub DrawRectangleWithShadow()
  Dim oPage    'Zeichnungsfolie
  Dim oShape   'Einzufügende Form
  Dim oDrawDoc 'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(CreatePoint(1000, 1000))
  oShape.setSize(CreateSize(4000, 1000))
  oShape.setString("Box 1")
  oShape.Shadow = True

  oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(CreatePoint(6000, 1000))
  oShape.setSize(CreateSize(4000, 1000))
  oShape.setString("Box 2")
  oShape.Shadow = True
  oShape.ShadowXDistance = -150   'Schattenversatz nach links
  oShape.CornerRadius = 100       'Runde Ecken
  'Aktiviert die Folie.
  oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Bild 125. *Boxen mit unterschiedlichen Schatten und runden Ecken bei der zweiten Box.*

Rotation und Scherung

Der Service com.sun.star.drawing.RotationDescriptor leistet über die Eigenschaften RotateAngle und ShearAngle das Drehen und Scheren einer Form. Die Eigenschaften werden als Long Integer in der Maßeinheit 1/100 Grad angegeben. Die Rotation erfolgt gegen den Uhrzeigersinn um die Mitte des die Form einschließenden Rechtecks. Eine Form zu scheren bedeutet, dass sie gestreckt wird und dass zum Beispiel aus einem Rechteck ein Parallelogramm wird. Die Scherung erfolgt im Gegensatz zur Rotation jedoch im Uhrzeigersinn um die Mitte des die Form einschließenden Rechtecks.

Das Makro im Listing 521 rotiert ein Rechteck um 20 Grad gegen den Uhrzeigersinn und schert ein Rechteck um 25 Grad im Uhrzeigersinn. Es zeichnet auch ein normales Rechteck ohne jede Rotation oder Scherung, um Ihnen einen Eindruck der Effekte zu ermöglichen (s. Bild 126).

Listing 521. *Rotation und Scherung von Rechtecken mit Text.*

```
Sub DrawRotateRectangle()
  Dim oPage    'Zeichnungsfolie
  Dim oShape   'Einzufügende Form
  Dim oDrawDoc 'Temporäres Draw-Dokument
```

```
    oDrawDoc = LoadEmptyDocument("sdraw")

    oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
    oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
    oPage.add(oShape)
    oShape.setPosition(CreatePoint(1000, 1000))
    oShape.setSize(CreateSize(4000, 1500))
    oShape.setString("Box 1")
    oShape.RotateAngle = 2000  'Um 20 Grad gedreht

    oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
    oPage.add(oShape)
    oShape.setPosition(CreatePoint(1000, 1000))
    oShape.setSize(CreateSize(4000, 1500))
    oShape.FillStyle = com.sun.star.drawing.FillStyle.NONE   'Keine Flächenfüllung
    oShape.LineStyle = com.sun.star.drawing.LineStyle.DASH   'Gestrichelte Linien

    oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
    oPage.add(oShape)
    oShape.setPosition(CreatePoint(6000, 1000))
    oShape.setSize(CreateSize(4000, 1500))
    oShape.setString("Box 2")
    oShape.ShearAngle = 2500   'Um 25 Grad geschert

    oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
    oPage.add(oShape)
    oShape.setPosition(CreatePoint(6000, 1000))
    oShape.setSize(CreateSize(4000, 1500))
    oShape.FillStyle = com.sun.star.drawing.FillStyle.NONE   'Keine Flächenfüllung
    oShape.LineStyle = com.sun.star.drawing.LineStyle.DASH   'Gestrichelte Linien
    'Aktiviert die Folie.
    oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

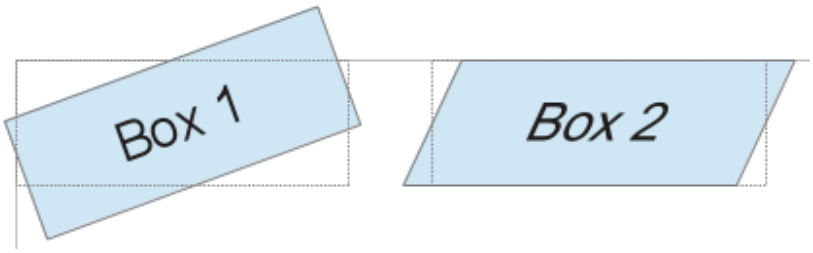


Bild 126. *Rotiertes und geschertes Rechteck, mit den originalen Rechtecken in gestrichelten Linien.*

16.2.2. Formtypen

OOo unterstützt viele verschiedene Formtypen, die aufeinander aufbauen. Die meisten dieser Typen erkennt man an ihren Namen. Zum Beispiel ist ein LineShape eine Linie. Zu Anfang war ich jedoch ein wenig verwirrt von der verschwenderischen Verwendung des Wortes „Poly“ in solchen Namen wie PolyLineShape und PolyPolygonShape. Die Vorsilbe „Poly“ kommt aus dem Griechischen und bedeutet „viele“. Demnach ist ein Polygon in OOo eine Figur, die viele Winkel enthält., denn im Griechischen heißt Winkel „gonia“. Ein PolyLineShape enthält viele Linien, und ein PolyPolygon-Shape enthält viele Polygone.

Einfache Linien

Der Service LineShape hat die Aufgabe, eine einfache Linie zu zeichnen. Ein LineShape benötigt eine Startposition (setPosition) und eine Größe (setSize). Das Makro im Listing 522 zeichnet eine Linie vom Punkt (1000, 1000) zum Punkt (1999, 1999). Der Endpunkt einer Linie wird durch die Größe der Form bestimmt.

Listing 522. *Darstellung einer Linie.*

```
Sub SimpleLine
  Dim oPage    'Zeichnungsfolie
  Dim oShape   'Einzufügende Form
  Dim oDrawDoc 'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.LineShape")
  oPage.add(oShape)
  oShape.setPosition(CreatePoint(1000, 1000))
  oShape.setSize(CreateSize(1000, 1000))
  'Aktiviert die Folie.
  oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Ich habe es zwar nie in einer Anwendung gesehen, aber der Service LineShape unterstützt den Service PolyPolygonDescriptor (s. Tabelle 249). Implizit werden einfache Linien als offene Polygone mit nur einer Linie repräsentiert. In anderen Services findet sich ebenfalls der PolyPolygonDescriptor.

Tabelle 249. *Eigenschaften im Service com.sun.star.drawing.PolyPolygonDescriptor.*

Eigenschaft	Beschreibung
PolygonKind	Diese nur lesbare Eigenschaft identifiziert den Polygontyp (s. Tabelle 250).
PolyPolygon	Referenzpunkte für dieses Polygon als Array von Arrays. Jedes enthaltene Array ist ein Array von com.sun.star.awt.Point-Structs. Aus diesen Punkten wird das Polygon gezeichnet. Sie könnten auch durch eine Rotation oder eine andere Transformation umgewandelt sein.
Geometry	Die Punkte des Polygons ohne Transformation.

Die Enumeration PolygonKind kennzeichnet den Typ des Polygons (s. Tabelle 250). PolygonKind ist eine nur lesbare Eigenschaft im Service PolyPolygonDescriptor (s. Tabelle 249). Man kann also erfahren, um welchen Typ es sich handelt, man kann ihn aber nicht ändern.

Tabelle 250. *Die Enumeration com.sun.star.drawing.PolygonKind.*

Wert	Beschreibung
LINE	LineShape = einfache Linie.
POLY	PolyPolygonShape = geschlossenes Polygon.
PLIN	PolyLineShape = offenes Polygon.
PATHLINE	OpenBezierShape = offene Bézierkurve.
PATHFILL	ClosedBezierShape = geschlossene Bézierkurve.
FREELINE	OpenFreeHandShape = offene Freihandzeichnung.
FREEFILL	ClosedFreeHandShape = geschlossene Freihandzeichnung.
PATHPOLY	PolyPolygonPathShape = geschlossene Polygonkontur.

Wert	Beschreibung
PATHPLIN	PolyLinePathShape = offene Polygonkontur.

Über die Eigenschaft PolyPolygon in der Tabelle 249 können Sie direkt die Punkte inspizieren, aus denen die Gestalt der Linie entsteht. Der Code im Listing 523 setzt voraus, dass oShape ein LineShape-Objekt enthält, und gibt die beiden Punkte der Linie aus.

Listing 523. *Überprüfung der Punkte in einem LineShape-Objekt.*

```
x = oShape.PolyPolygon(0)
MsgBox "" & x(0).X & " und " & x(0).Y
MsgBox "" & x(1).X & " und " & x(1).Y
```

Offenes Polygon (PolyLineShape)

Der Service LineShape definiert eine einzelne Linie. Der Service PolyLineShape definiert eine Reihe von Linien. Ein LineShape wird durch seine Position und seine Größe bestimmt. Ein PolyLineShape wird jedoch vom PolyPolygonDescriptor (s. Tabelle 249) definiert. Wenn man weiß wie, ist es einfach, ein PolyLineShape zu erzeugen, viele verstehen es aber nicht.

Tipp Die Eigenschaft PolyPolygon ist ein Array von Arrays, die Punkte enthalten.

Die Linien im PolyLineShape werden von der Eigenschaft PolyPolygon definiert, die ihrerseits ein Array ist, das ein oder mehrere Arrays von Punkten enthält. Jedes Punktearray wird als Reihe von verbundenen Linien gezeichnet, aber die einzelnen Arrays sind nicht eigens miteinander verbunden. Das Makro im Listing 524 erzeugt zwei Punktearrays (oPoints_1 und oPoints_2). Dann werden die Arrays in einem weiteren Array gespeichert.

Listing 524. *Zeichnung eines einfachen PolyLineShape.*

```
Sub SimplePolyLineShape
  Dim oPage     'Zeichnungsfolie
  Dim oShape    'Einzufügende Form
  Dim oPoints_1 'Erster Satz von Zeichnungspunkten
  Dim oPoints_2 'Zweiter Satz von Zeichnungspunkten
  Dim oDrawDoc  'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  oPoints_1 = Array(CreatePoint(1000, 1000),_
                    CreatePoint(3000, 2000),_
                    CreatePoint(1000, 2000),_
                    CreatePoint(3000, 1000))

  oPoints_2 = Array(CreatePoint(4000, 1200),_
                    CreatePoint(4000, 2000),_
                    CreatePoint(5000, 2000),_
                    CreatePoint(5000, 1200))

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.PolyLineShape")
  oPage.add(oShape)
  'PolyPolygon erwartet ein Array von Arrays.
  oShape.PolyPolygon = Array(oPoints_1, oPoints_2)
  oShape.LineWidth = 50
  'Aktiviert die Folie.
  oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Bild 127. *Ein einfaches PolyLineShape erzeugt zwei unverbundende Formen.*

Tipp	Zuerst wird die Form der Folie hinzugefügt, danach erst werden die Punkte übergeben.

Die Eigenschaft PolyPolygon ist ein Array von Arrays. Sie können das Makro im Listing 524 auch mit nur einem Punktesatz einsetzen, dennoch muss sich dieses einzige Punktearray immer noch in einem zweiten Array befinden..

Listing 525. *Die Eigenschaft PolyPolygon ist ein Array von Punktarrays.*

```
oShape.PolyPolygon = Array(oPoints_1)
```

Geschlossenes Polygon (PolyPolygonShape)

Der Service PolyPolygonShape definiert eine Reihe von geschlossenen Polygonen, die nicht miteinander verbunden sind (s. Bild 128). Eigentlich ist der Service eine Version des PolyLineShape in geschlossener Form. Weil er geschlossene Formen produziert, unterstützt er die Fülleigenschaften.

Das Makro im Listing 526 nutzt denselben Punktesatz wie das Makro im Listing 524. Ich habe die Dinge aber etwas anders aufgebaut, um Ihnen eine weitere Methode zur Erzeugung von Arrays von Punktearrays zu demonstrieren. Allerdings erzeugen beide Makros zuerst die Form und fügen sie der Folie zu, dann erst werden die Eigenschaften gesetzt.

Listing 526. *Zeichnung eines einfachen gefüllten PolyPolygonShape.*

```
Sub SimplePolyPolygonShapeFilled
  Dim oPage     'Zeichnungsfolie
  Dim oShape    'Einzufügende Form
  Dim oDrawDoc  'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.PolyPolygonShape")
  oPage.add(oShape)
  oShape.PolyPolygon = Array(Array(CreatePoint(1000, 1000),_
                                   CreatePoint(3000, 2000),_
                                   CreatePoint(1000, 2000),_
                                   CreatePoint(3000, 1000)),_
                             Array(CreatePoint(4000, 1200),_
                                   CreatePoint(4000, 2000),_
                                   CreatePoint(5000, 2000),_
                                   CreatePoint(5000, 1200)))
  oShape.LineWidth = 50
  'Aktiviert die Folie.
  oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Bild 128. *Das PolyPolygonShape erzeugt eine geschlossene Version des PolyLineShape.*

Rechteck und Textrahmen

Nach außen sind RectangleShape und TextShape praktisch identisch. Beide Typen unterstützen denselben Satz von Services (mit Ausnahme des definierenden Service natürlich) und können so konfiguriert werden, dass sie auf dieselbe Art dargestellt werden. Der Hauptunterschied liegt in ihren Standardwerten für die Bildschirmausgabe. Prinzipiell können die Eigenschaften aus den Standardwerten so angepasst werden, dass beide Typen die eine wie die andere Ausgabe produzieren. Das Makro im Listing 527 erzeugt ein Rechteck neben einem Text (s. Bild 129).

Listing 527. *Zeichnung eines Rechtecks und eines Textobjekts.*

```
Sub SimpleRectangleShape
  Dim oPage     'Zeichnungsfolie
  Dim oShape    'Einzufügende Form
  Dim oDrawDoc  'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  'Die Rechteck-Form
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)

  oShape.setPosition(CreatePoint(1000, 1000))
  oShape.setSize(CreateSize(6000, 1000))
  oShape.setString("Rechteck")
  oShape.Shadow = True

  'Die Text-Form
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.TextShape")
  oPage.add(oShape)

  oShape.setPosition(CreatePoint(8000, 1000))
  oShape.setSize(CreateSize(10000, 1000))
  oShape.setString("Text")
  oShape.Shadow = True
  'Aktiviert die Folie.
  oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

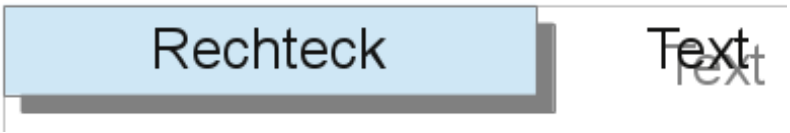


Bild 129. *Die nahezu identischen Formtypen RectangleShape und TextShape werden standardmäßig unterschiedlich dargestellt.*

Beide Formen RectangleShape und TextShape unterstützen die Eigenschaft CornerRadius. Der Eckenradius, ein Long Integer, ist der Radius des Kreises, mit dem runde Ecken produziert werden. Ein Beispiel dafür liefert das Listing 520, zu sehen im Bild 125.

Ellipse

Ein Mathematiker würde von einer Ellipse sagen, sie sei eine geschlossene Kurve mit zwei Brennpunkten, in der die Summe der Entfernungen von jedem Punkt der Kurve zu den zwei Brennpunkten konstant ist. Wenn die beiden Brennpunkte in einem Punkt liegen, ist die Ellipse ein Kreis. Vereinfacht gesagt ist ein Kreis eine Ellipse und eine Ellipse ein gequetschter Kreis.

Beim Zeichnen eines Rechtecks wird die Position von der linken oberen Ecke des Rechtecks bestimmt, und die Größe wird dann durch die Breite und die Höhe definiert. Wenn dieselben Punkt- und Größenangaben für eine Ellipse verwendet werden, wird sie innerhalb des Rechtecks liegen und dessen vier Seiten nur gerade eben berühren. Mathematisch gesehen sind die Rechteckseiten Tangenten der Ellipse zu ihren Hauptachsen, dem größten und dem kleinsten Durchmesser durch den Ellipsenmittelpunkt. Das Makro im Listing 528 beginnt mit dem Zeichnen von vier elliptischen Formen. Die letzte Ellipse wird um 30 Grad gedreht. Dann zeichnet das Makro ein Rechteck mit derselben Position, derselben Lage und derselben Rotation wie die letzte Ellipse (s. Bild 130). Dieses letzte Rechteck dient der Illustration des Zusammenhangs zwischen Rechteck und Ellipse.

Listing 528. *Zeichnung elliptischer Formen.*

```
Sub SimpleEllipseShapes
  Dim oPage     'Zeichnungsfolie
  Dim oShape    'Einzufügende Form
  Dim i%        'Schleifenvariable
  Dim x         'Eine Lageangabe
  Dim nLocs     'Array der Lageangaben
  Dim oDrawDoc  'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  nLocs = Array(Array(CreatePoint(1000, 1000), CreateSize(1000, 1000)),_
                Array(CreatePoint(3000, 1000), CreateSize(1000, 1500)),_
                Array(CreatePoint(5000, 1000), CreateSize(1500, 1000)),_
                Array(CreatePoint(7000, 1000), CreateSize(1500, 1000)))

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  For i = LBound(nLocs) To UBound(nLocs)
    oShape = oDrawDoc.createInstance("com.sun.star.drawing.EllipseShape")
    oPage.add(oShape)
    x = nLocs(i)
    oShape.setPosition(x(0))
    oShape.setSize(x(1))
    oShape.setString(i)
  Next
  'Die letzte Ellipse wird um 30° rotiert.
  oShape.RotateAngle = 3000

  REM Nun wird ein Rechteck an derselben Stelle und von derselben Größe
  REM und Rotation wie die letzte Ellipse gezeichnet.
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape")
  oPage.add(oShape)
  oShape.setPosition(x(0))
  oShape.setSize(x(1))
  oShape.RotateAngle = 3000
  oShape.FillStyle = com.sun.star.drawing.FillStyle.NONE  'Ohne Flächenfüllung
  'Aktiviert die Folie.
  oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Bild 130. *Die Größenparameter bestimmen die Gestalt der Formen, andere Parameter bestimmen die Position und die Ausrichtung.*

Der Service EllipseShape enthält eine Eigenschaft vom Typ CircleKind, mit der festgelegt wird, ob die gesamte Ellipse oder nur ein Teil von ihr gezeichnet wird (s. Tabelle 251). Man kann also einen Ellipsenbogen zeichnen. Die Eigenschaften CircleStartAngle und CircleEndAngle definieren den Beginn und das Ende des Bogens. Jede der Ellipsen im Bild 130 hat als CircleKind FULL.

Tabelle 251. *Die Enumeration com.sun.star.drawing.CircleKind.*

Wert	Beschreibung
FULL	Eine voll durchgezogene Ellipse.
SECTION	Eine Ellipse mit einem durch zwei Linien verbundenen Ausschnitt (Sektor).
CUT	Eine Ellipse mit einem durch eine Linie verbundenen Ausschnitt (Segment).
ARC	Eine Ellipse mit einem offenen Ausschnitt (Bogen).

Achtung Achtung. Sowohl bei AOO als auch bei LO sind in der API-Dokumentation die Beschreibungen von SECTION und CUT vertauscht. Tatsächlich aber ist SECTION ein Sektor und CUT ein Segment (s.a. Listing 529 und Bild 131).

Die vier verschiedenen Ellipsenformen werden durch das Makro im Listing 529 gezeichnet und im Bild 131 gezeigt.

Listing 529. *Zeichnung von Ellipsenbögen.*

```
Sub ArcEllipseShapes
  Dim oPage     'Zeichnungsfolie
  Dim oShape    'Einzufügende Form
  Dim nKinds    'Array der Ellipsendarstellungen
  Dim oDrawDoc  'Temporäres Draw-Dokument
  Dim oCclKind  'Kürzer als com.sun.star.drawing.CircleKind
  Dim i%

  oDrawDoc = LoadEmptyDocument("sdraw")
  oCclKind = com.sun.star.drawing.CircleKind
  'Ellipsendarstellungen: Voll, Sektor, Segment, Bogen
  nKinds = Array(oCclKind.FULL,_
                 oCclKind.SECTION,_
                 oCclKind.CUT,_
                 oCclKind.ARC)

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
  '4 Ellipsen unterschiedlicher Darstellung
  For i = LBound(nKinds) To UBound(nKinds)
    oShape = oDrawDoc.createInstance("com.sun.star.drawing.EllipseShape")
    oPage.add(oShape)
    oShape.setPosition(CreatePoint((i + 1) * 2000, 1000))
    oShape.setSize(CreateSize(1000, 700))
    oShape.setString(i)
    oShape.CircleStartAngle = 9000
    oShape.CircleEndAngle = 36000
    oShape.CircleKind = nKinds(i)
```

```
    Next
    'Aktiviert die Folie.
    oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Bild 131. *Ellipsenformen in der Reihenfolge der CircleKind-Werte.*

Bézierkurven

Eine Bézierkurve ist eine regelmäßige Kurve, die über eine Anzahl von Punkten geregelt wird. Bézierkurven verbinden den ersten mit dem letzten Punkt, ihre Form wird aber über die anderen Punkte beeinflusst. Mathematiker lieben Bézierkurven, denn sie sind gleichbleibend unter affinen Abbildungen (jede beliebige Kombination von Verschiebung oder Rotation). Professionelle Computergrafiker lieben Bézierkurven, denn man kann sie auf einfache Weise ändern und transformieren.

Bézierkurven werden durch einen PolyPolygonBezierDescriptor gesteuert (s. Tabelle 252), der beinahe identisch mit dem PolyPolygonDescriptor der Tabelle 249 ist. Der Unterschied zwischen den beiden Deskriptoren liegt darin, dass jeder Punkt in einer Bézierkurve zu einer bestimmten Kategorie gehört, die regelt, wie der Punkt die Kurve beeinflusst.

Tabelle 252. *Eigenschaften im Service com.sun.star.drawing.PolyPolygonBezierDescriptor.*

Eigenschaft	Beschreibung
PolygonKind	Diese nur lesbare Eigenschaft identifiziert den Polygontyp (s. Tabelle 250).
PolyPolygonBezier	Referenzpunkte für diese Bézierkurve. Es ist ein Struct PolyPolygonBezierCoords, das ein Array von Punkten enthält sowie ein Array von Flags für die Kategorie jedes Punktes in Bezug auf seine Funktion für die Kurve.
Geometry	Die Punkte der PolyPolygonBezierCoords ohne Transformation.

Die Eigenschaft PolyPolygonBezier (s. Tabelle 252) ist ein Struct PolyPolygonBezierCoords, das zwei Eigenschaften enthält, Coordinates und Flags. Die Eigenschaft Coordinates ist ein Array von Arrays von Kontrollpunkten für die Bézierkurve. Die Eigenschaft Flags ist ein Array von Arrays von PolygonFlags (s. Tabelle 253) mit der Angabe, auf welche Art die Kurve von den Punkten beeinflusst wird.

Tabelle 253. *Die Enumeration com.sun.star.drawing.PolygonFlags.*

Wert	Beschreibung
NORMAL	Normaler Punkt (aus der Sicht der Kurvendiskussion).
SMOOTH	Glatter Punkt (erste Ableitung der Kurvendiskussion).
CONTROL	Kontrollpunkt, der die Kurve von der Benutzerschnittstelle her beeinflusst.
SYMMETRIC	Symmetrischer Punkt (zweite Ableitung der Kurvendiskussion).

Das Makro im Listing 530 zeichnet einen kleinen Kreis an jedem Punkt des Arrays, um anschaulich verständlich zu machen, wie die verschiedenen Punkte auf die Gestalt einer Bézierkurve wirken.

Listing 530. *Zeichnung von Kreisen an jedem Punkt eines Arrays.*

```
Sub DrawControlPoints(oCoords, oPage, oDoc, nWidth As Long)
  REM oCoords = Koordinaten des einzufügenden Polygons
  REM nWidth  = Kreisdurchmesser

  Dim oPoints  'Unterarray von Punkten
  Dim oPoint   'Ein Punkt
```

```
  Dim oFlags   'Unterarray von Flags
  Dim oShape   'Der zu zeichnende Kreis
  Dim nShape%  'Index zu den oCoords-Arrays
  Dim i%       'Allgemeine Indexvariable

  For nShape = LBound(oCoords.Coordinates) To UBound (oCoords.Coordinates)
    oPoints = oCoords.Coordinates(nShape)
    oFlags  = oCoords.Flags(nShape)
    For i = LBound(oPoints) To UBound(oPoints)
      oShape = oDoc.createInstance("com.sun.star.drawing.EllipseShape")
      oPage.add(oShape)
      oPoint = oPoints(i)
      REM Um den Kreis am Mittelpunkt auszurichten, muss ich die Position
      REM um den halben Durchmesser nach links und nach oben setzen.
      oShape.setPosition(CreatePoint(oPoint.X - nWidth / 2, _
                                     oPoint.Y - nWidth / 2))
      oShape.setSize(CreateSize(nWidth, nWidth))
    Next
  Next
End Sub
```

Listing 531 zeichnet zwei unverbundene Bézierkurven (s.Bild 132). Die zweite Kurve setzt zwei Kontrollpunkte an dieselbe Stelle. Das Makro DrawControlPoints im Listing 530 zeichnet die Kontrollpunkte zusätzlich als sichtbare Kreise.

Listing 531. *Zeichnung einer offenen Bézierkurve.*

```
Sub DrawOpenBezierCurves()
  Dim oPage     'Zeichnungsfolie
  Dim oShape    'Einzufügende Form
  Dim oCoords   'Koordinaten des einzufügenden Polygons
  Dim oDrawDoc  'Temporäres Draw-Dokument
  Dim oPlgFlags 'Kürzer als com.sun.star.drawing.PolygonFlags
  Dim i%

  oDrawDoc = LoadEmptyDocument("sdraw")
  oPlgFlags = com.sun.star.drawing.PolygonFlags
  oCoords = CreateUnoStruct("com.sun.star.drawing.PolyPolygonBezierCoords")

  REM Angabe der verwendeten Koordinaten. Der erste wie der letzte Punkt
  REM sind normale Punkte, die mittleren sind Bézierkontrollpunkte.
  oCoords.Coordinates = Array(Array(CreatePoint(1000, 1000),_
                                    CreatePoint(2000, 3000),_
                                    CreatePoint(3000, 0500),_
                                    CreatePoint(4000, 1000)),_
                              Array(CreatePoint(5000, 1000),_
                                    CreatePoint(6500, 0200),_
                                    CreatePoint(6500, 0200),_
                                    CreatePoint(8000, 1000)))
  oCoords.Flags = Array(Array(oPlgFlags.NORMAL,_
                              oPlgFlags.CONTROL,_
                              oPlgFlags.CONTROL,_
                              oPlgFlags.NORMAL),_
                        Array(oPlgFlags.NORMAL,_
                              oPlgFlags.CONTROL,_
                              oPlgFlags.CONTROL,_
                              oPlgFlags.NORMAL))

  oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
```

```
    oShape = oDrawDoc.createInstance("com.sun.star.drawing.OpenBezierShape")
    oPage.add(oShape)
    oShape.PolyPolygonBezier = oCoords
    DrawControlPoints(oCoords, oPage, oDrawDoc, 100)
    'Aktiviert die Folie.
    oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

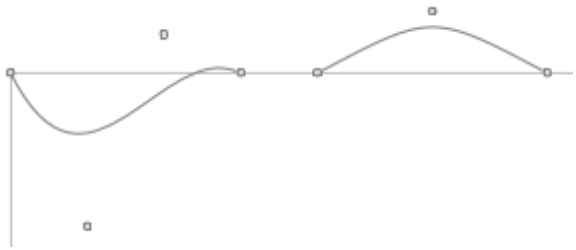


Bild 132. *Der Einfluss von Kontrollpunkten auf eine Bézierkurve.*

Nicht alle Kombinationen von Punkten und Flags sind gültig. Eine vollständige Erörterung gültiger Kombinationen von Punkten und Flags geht über den Rahmen dieses Buches hinaus. Wenn Sie eine falsche Anzahl von Punkten oder eine nicht unterstützte Folge von Kontrollflags einsetzen, erhalten Sie einen Laufzeitfehler.

Verbinder

Mit der Form ConnectorShape gestalten Sie eine Verbindung zwischen zwei Formen. Ein „Klebepunkt“ ist eine Stelle innerhalb einer Form, an der der Endpunkt eines Verbinders andocken kann. Jeder Klebepunkt wird durch das Struct GluePoint2 definiert (s. Tabelle 254).

Tabelle 254. *Eigenschaften des Structs com.sun.star.drawing.GluePoint2.*

Eigenschaft	Beschreibung
Position	Position des Klebepunkts als Struct Point.
IsRelative	True = Angabe der Position in 1/100 Prozent, False = Angabe der Position in 1/100 mm.
PositionAlignment	Falls die Position nicht relativ ist, wird die horizontale und vertikale Ausrichtung des Klebepunkts angegeben: Enumeration com.sun.star.drawing.Alignment: TOP_LEFT (oben links), TOP (oben), TOP_RIGHT (oben rechts), LEFT (links), CENTER (Zentrum), RIGHT (rechts), BOTTOM_LEFT (unten links), BOTTOM (unten) und BOTTOM_RIGHT (unten rechts).
Escape	Richtung, in die der Verbinder aus einem Klebepunkt hinausgeht: Enumeration com.sun.star.drawing.EscapeDirection: SMART = Selbständig. LEFT = Nach links, der Text ist linksbündig am linken Rand der Form angeordnet. RIGHT = Nach rechts, der Text ist rechtsbündig am rechten Rand der Form angeordnet. UP = Nach oben. DOWN = Nach unten. HORIZONTAL = Spiegelt die Horizontalachse. VERTICAL = Spiegelt die Vertikalachse.
IsUserDefined	False = Standardklebepunkt, True = Benutzerdefinierter Klebepunkt.

Jede Form enthält standardmäßig je einen Klebepunkt oben, rechts, unten und links an der Form. Mit der Methode getGluePoints() (s. Tabelle 240) erhalten Sie die Klebepunkte der Form. Die Indexwerte der Standardklebepunkte sind 0 (oben), 1 (rechts), 2 (unten) und 3 (links). Listing 532 zeigt Ihnen, wie man den Standardpunkten weitere Klebepunkte hinzufügen kann.

Verbinder enthalten die Eigenschaften StartPosition und EndPosition, nämlich die Ausgangs- und die Zielposition des Verbinders. Diese Positionen werden nur dann gesetzt, wenn die Eigenschaften StartShape und EndShape leer sind. Wenn diese nicht leer sind, verbindet sich ein Verbinder mit einem Klebepunkt der entsprechenden Form. Verbinder beziehen sich auf die Klebepunkte anderer Formen über die Indexwerte der Eigenschaften StartGluePointIndex und EndGluePointIndex.

Tabelle 255. *Eigenschaften im Service com.sun.star.drawing.ConnectorShape.*

Eigenschaft	Beschreibung
StartShape	Ausgangsform, oder leer, wenn der Ausgangspunkt nicht mit einer Form verbunden ist.
StartGluePointIndex	Indexwert des Klebepunkts in der Ausgangsform.
StartPosition	Anfangspunkt in 1/100 mm. Dieser Punkt kann nur gesetzt werden, wenn StartShape leer ist. Gelesen werden kann er in jedem Fall.
EndShape	Zielform, oder leer, wenn der Ausgangspunkt nicht mit einer Form verbunden ist.
EndPosition	Zielpunkt in 1/100 mm. Dieser Punkt kann nur gesetzt werden, wenn EndShape leer ist. Gelesen werden kann er in jedem Fall.
EndGluePointIndex	Indexwert des Klebepunkts in der Zielform.
EdgeLine1Delta	Länge der Linie 1.
EdgeLine2Delta	Länge der Linie 2.
EdgeLine3Delta	Länge der Linie 3.
EdgeKind	Typ des Verbinders (s. Tabelle 256).

Vier Verbindertypen werden unterstützt (s. Tabelle 256). Der Verbindertyp bestimmt die Gestaltung der Linie, die zwischen zwei Punkten gezogen wird. Der Typ STANDARD bevorzugt drei Linien zur Verbindung zwischen den Formen, bei Bedarf können es aber auch mehr sein.

Tabelle 256. *Die Enumeration com.sun.star.drawing.ConnectorType.*

Wert	Beschreibung
STANDARD	Der Verbinder wird mit drei Linien gezogen, die mittlere senkrecht zu den anderen beiden.
CURVE	Der Verbinder wird als Kurve gezogen.
LINE	Der Verbinder wird als eine gerade Linie gezogen.
LINES	Der Verbinder wird mit drei Linien gezogen.

Das Makro im Listing 532 zeichnet vier Rechtecke und verbindet sie dann mit je einem Verbinder (ConnectorShape). Das Makro legt den Ausgangsklebepunkt fest, überlässt aber OOo die automatische Wahl des Zielpunkts. Würde der Ausgangsklebepunkt nicht gezielt gesetzt, dann würde auch er automatisch festgelegt. Wird ein Klebepunkt automatisch gewählt, geschieht das auf intelligente Weise, wie Sie im Bild 133 sehen können.

Listing 532. *Illustration von vier Verbindertypen.*

```
Sub DrawConnectorShapeNormal
  DrawConnectorShape(False, False)
End Sub

Sub DrawConnectorShape(doCustom As Boolean, useArrows As Boolean)
  REM doCustom  = benutzerdefinierte Klebepunkte (True),
  REM             Standardklebepunkte (False)
  REM useArrows = Verbinder mit oder ohne Pfeilenden

  Dim oPage     'Zeichnungsfolie
  Dim oShapes   'Einzufügende Formen
  Dim oShape    'Einzelne Form
  Dim nConTypes 'Array von Verbindertypen
```

```
Dim oGlue     'Benutzerdefinierter Klebepunkt
Dim oDrawDoc  'Temporäres Draw-Dokument
Dim oStyles   'Bildvorlagen
Dim oConnType 'Kürzer als com.sun.star.drawing.ConnectorType
Dim i%

oDrawDoc = LoadEmptyDocument("sdraw")
oConnType = com.sun.star.drawing.ConnectorType

nConTypes = Array(oConnType.STANDARD,_
                  oConnType.CURVE,_
                  oConnType.LINE,_
                  oConnType.LINES)

oShapes = Array(oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape"),_
                oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape"),_
                oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape"),_
                oDrawDoc.createInstance("com.sun.star.drawing.RectangleShape"))

REM Die Folie wird erzeugt, dann werden die Formen eingefügt,
REM danach erst werden sie manipuliert.
oPage = CreateDrawPage(oDrawDoc, "Draw-Test", True)
For i = 0 To 3
  oPage.add(oShapes(i))
  oShapes(i).setSize(CreateSize(1300, 1000))
Next

oShapes(0).setPosition(CreatePoint(3000, 3500))
oShapes(1).setPosition(CreatePoint(6000, 3000))
oShapes(2).setPosition(CreatePoint(9000, 2500))
oShapes(3).setPosition(CreatePoint(6000, 4500))

For i = 0 To 3
  oShapes(i).setString(i)
  oShape = oDrawDoc.createInstance("com.sun.star.drawing.ConnectorShape")
  oPage.add(oShape)
  oShape.StartShape = oShapes(i)
  oShape.StartGluePointIndex = i  '0 = oben, 1 = rechts, 2 = unten, 3 = links
  oShape.EndShape = oShapes((i + 1) Mod 4)  '0 -> 1 -> 2 -> 3 -> 0
  oShape.EdgeKind = nConTypes(i)

  If doCustom Then
    REM Benutzerdefinierte Klebepunkte im Mittelpunkt der Form
    oGlue = CreateUnoStruct("com.sun.star.drawing.GluePoint2")
    oGlue.IsRelative = False
    oGlue.Escape = com.sun.star.drawing.EscapeDirection.SMART
    'Positionierung in der Mitte der Form
    oGlue.PositionAlignment = com.sun.star.drawing.Alignment.CENTER
    oGlue.Position.X = 0     'Keine horizontale Abweichung von der Mitte
    oGlue.Position.Y = 0     'Keine vertikale Abweichung von der Mitte
    oGlue.IsUserDefined = True
    oShape.StartGluePointIndex = oShapes(i).getGluePoints().insert(oGlue)
  End If

  If useArrows Then
    REM Optional: Verbinder mit Pfeilenden
```

```
        oStyles = oDrawDoc.getStyleFamilies().getByName("graphics")
        'Standard-Pfeilstil in älteren AO- Und LO-Versionen: objectwitharrow
        'In neueren LO-Versionen: Arrow Line
        If oStyles.hasByName("objectwitharrow") Then
          oShape.Style = oStyles.getByName("objectwitharrow")
        Else
          oShape.Style = oStyles.getByName("Arrow Line")
        End If
      End If
    Next
    'Aktiviert die Folie.
    oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

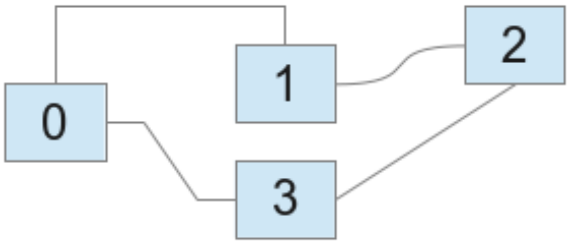


Bild 133. *Unterschiedliche Typen von Verbindern zwischen Formen.*

Tipp	Viele der Eigenschaften einer Form werden zurückgesetzt, wenn die Form in eine Folie eingefügt wird. Sie sollten also die meisten Eigenschaften erst nach dem Einfügen setzen. Es ist zudem wichtig, in welcher Reihenfolge Sie die Eigenschaften einrichten, denn das Setzen mancher Eigenschaften bewirkt, dass andere zurückgesetzt werden. Wenn Sie beispielsweise die Ausgangsform (StartShape) setzen, wird zugleich der StartGluePointIndex zurückgesetzt.

Eigene Klebepunkte erzeugen

Wenn Sie den Verbinder an einem selbst definierten Klebepunkt andocken lassen wollen, müssen Sie ein GluePoint2-Struct erzeugen und es der gewünschten Form hinzufügen. Unmittelbar nach der Festlegung der Eigenschaft EdgeKind erstellen Sie einen Klebepunkt beispielsweise im Zentrum des Rechtecks und wählen diesen Punkt als Ausgangspunkt. Tabelle 254 enthält eine Beschreibung des Structs GluePoint2.

Listing 533. *Verbinder aus dem Zentrum der Form heraus.*

```
Sub DrawConnectorShapeCenter
  DrawConnectorShape(True, False)
End Sub
```

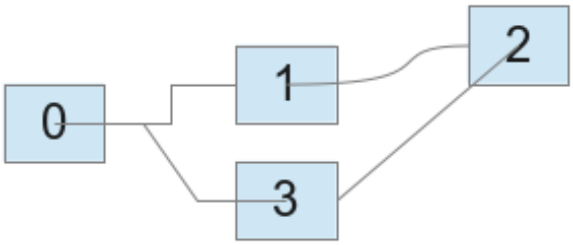


Bild 134. *Benutzerdefinierte Klebepunkte: Verbinder gehen von der Mitte der Rechtecke aus.*

Pfeile über Vorlagen hinzufügen

Viele Eigenschaften einer Form können über eine Vorlage gesetzt werden. Wenn Sie häufig bestimmte Füllungen und Schatten nutzen, erstellen Sie eine eigene Vorlage, um nach Bedarf schnell

die Objekte ändern zu können. Listing 534 gibt die von einem Draw-Dokument genutzten Vorlagenfamilien und Grafikvorlagen aus. Bei AOO und LO sind die Familien identisch, aber die Grafikvorlagen unterscheiden sich erheblich (s Bild 136).

Listing 534. *Ausgabe der unterstützten Grafikvorlagen.*

```
Sub PrintGraphicsStyles
  Dim oStyleFamilies   'Vorlagenfamilien
  Dim oStyles          'Vorlagen in einer Familie
  Dim oDrawDoc         'Temporäres Draw-Dokument.

  oDrawDoc = LoadEmptyDocument("sdraw")

  oStyleFamilies = oDrawDoc.getStyleFamilies()
  MsgBox Join(oStyleFamilies.getElementNames(), Chr$(10)), 0, "Familien"

  oStyles = oStyleFamilies.getByName("graphics")
  MsgBox Join(oStyles.getElementNames(), Chr$(10)), 0, "Grafikvorlagen"
  oDrawDoc.close(True)
End Sub
```

Bild 135. *Von Draw-Dokumenten unterstützte Vorlagenfamilien.*

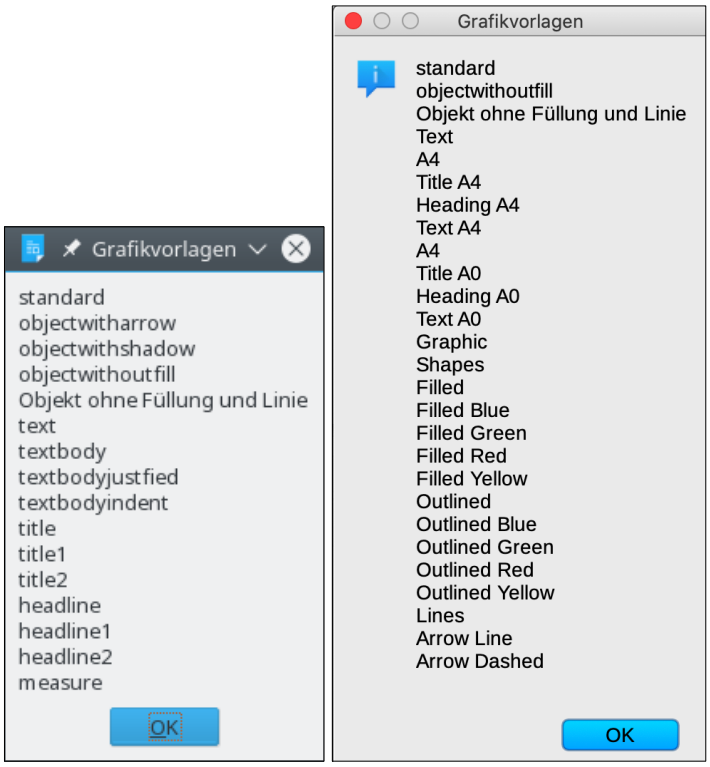


Bild 136. *Von Draw-Dokumenten unterstützte Grafikvorlagen, links AOO, rechts LO.*

Sie können einer Form Pfeile hinzufügen, indem Sie der Form die Vorlage „Arrow Line“ zuweisen (s. Bild 137), wenn sie denn als Vorlage zur Verfügung steht. Es könnte je nach AO- oder LO-Version auch die Vorlage „objectwitharrow“ sein. Sollten Ihnen die Standardwerte der Vorlagen nicht zusagen, so erstellen Sie eine eigene und nutzen diese.

Listing 535. *Zeichnet Verbinder mit Pfeilen.*

```
Sub DrawConnectorShapeWithArrows
  DrawConnectorShape(False, True)
End Sub
```

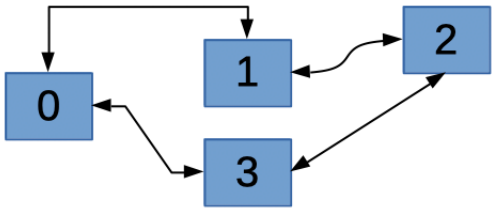


Bild 137. *Pfeile von der Zielform zur Ausgangsform mit der Pfeilvorlage „Arrow Line“.*

Eine Tabelle einfügen

Das folgende Makro fügt in die erste Folie eine Tabelle ein und schreibt einen Wert in eine einzelne Zelle. Schließlich erhält eine andere Zelle eine Grafik als Hintergrundfüllung.

Listing 536. *Eine Tabelle in einem Draw- oder Impress-Dokument.*

```
Sub DrawTableShape
  Dim oSize As New com.sun.star.awt.Size
  Dim oPos As New com.sun.star.awt.Point
  Dim oPage     'Zeichnungsfolie
  Dim oTable    'Einzufügende Tabelle
  Dim oCell     'Eine Zelle, die einen Wert erhalten soll
  Dim oDrawDoc  'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  'Größe und Position der Tabelle.
  oSize.Width = 6000    : oSize.Height = 6100
  oPos.X = 6000         : oPos.Y = 5000

  'Tabelle in der ersten Folie.
  oPage = oDrawDoc.DrawPages.getByIndex(0)
  oTable = oDrawDoc.createInstance("com.sun.star.drawing.TableShape")
  oPage.add(oTable)

  '5 Zeilen und 5 Spalten.
  oTable.Model.Rows.InsertByIndex(1, 4)
  oTable.Model.Columns.InsertByIndex(1, 4)
  oTable.setSize(oSize)
  oTable.setPosition(oPos)

  'Die Zelle in der ersten Spalte und der zweiten Zeile erhält den Text "X".
  oCell = oTable.Model.getCellByPosition(0, 1)
  oCell.getText().setString("X")

  'Die Zelle in der dritten Zeile und der dritten Spalte.
  oCell = oTable.Model.getCellByPosition(2, 2)
```

```
    'Füllung der Zelle mit der Bitmap eines Bildes.
    oCell.FillStyle = com.sun.star.drawing.FillStyle.BITMAP
    oCell.FillBitmapURL = _
            "http://www.openoffice.org/images/AOO_logos/OOo_Website_v2_copy.png"
End Sub
```

16.3. Formulare

Ein in ein Dokument eingefügtes Steuerelement wird als Formular gespeichert. Folien können Formulare enthalten, denn sie binden das Interface com.sun.star.form.XFormsSupplier ein. Der sichtbare Teil des Steuerelements wird in der Folie in der Form eines ControlShape gespeichert. Das Datenmodell für das Steuerelement wird in einem Formular gespeichert und vom ControlShape referenziert. Die Methode getForms() gibt ein Objekt zurück, das die Formulare für die Folie enthält (s. Tabelle 257).

Tabelle 257. *Einige über den Service com.sun.star.form.Forms verfügbare Methoden.*

Methode	Beschreibung
createEnumeration()	Erstellt eine Enumeration der Formulare.
getByIndex(Long)	Zugriff auf ein Formular über den Index.
getByName(String)	Zugriff auf ein Formular über den Namen.
getCount()	Anzahl der Formulare.
hasByName(String)	True = Ein Formular mit diesem Namen existiert.
hasElements()	True = Die Seite enthält wenigstens ein Formular.
insertByIndex(Long, Form)	Einfügung eines Formulars über den Index.
insertByName(String, Form)	Einfügung eines Formulars über den Namen.
removeByIndex(Long)	Entfernung eines Formulars über den Index.
removeByName(String)	Entfernung eines Formulars über den Namen.
replaceByIndex(Long, Form)	Ersetzung eines Formulars über den Index.
replaceByName(String, Form)	Ersetzung eines Formulars über den Namen.

Das Makro im Listing 537 soll zeigen, wie einer Folie ein Formular hinzugefügt wird. Formulare sind uninteressant, wenn kein Steuerelement dazugehört. Daher bietet Listing 537 eine Aufklappliste zum Auswählen aus einigen Werten an. Bei meinen Tests mit AOO erschien das Formular im Entwurfsmodus. Ich musste erst den Entwurfsmodus ausschalten, bevor ich das Steuerelement testen konnte.

Listing 537. *Ein Formular mit einem Steuerelement in der ersten Folie.*

```
Sub AddAForm
  Dim oPage         'Zeichnungsfolie
  Dim oShape        'Einzufügende Form
  Dim oForm         'Individuelles Formular
  Dim oControlModel 'Modell für ein Steuerelement
  Dim s(0 To 5) As String
  Dim oDrawDoc      'Temporäres Draw-Dokument

  oDrawDoc = LoadEmptyDocument("sdraw")

  REM Daten für die Combobox.
  s(0) = "Null"  : s(1) = "Eins"  : s(2) = "Zwei"
  s(3) = "Drei"  : s(4) = "Vier"  : s(5) = "Fünf"
```

```
    oPage = oDrawDoc.DrawPages.getByIndex(0)

    REM Eine neue Form für das Steuerelement.
    oShape = oDrawDoc.createInstance("com.sun.star.drawing.ControlShape")
    oShape.Position = CreatePoint(3000, 4500)
    oShape.Size = CreateSize(2500, 800)

    REM Ein neues Combobox-Modell.
    oControlModel = oDrawDoc.createInstance("com.sun.star.form.component.ComboBox")
    oControlModel.Name = "NumberSelection"
    oControlModel.Text = "Null"
    oControlModel.Dropdown = True
    oControlModel.StringItemList = s()

    REM Das Modell wird der Form zugewiesen.
    oShape.Control = oControlModel

    oForm = oDrawDoc.createInstance("com.sun.star.form.component.Form")
    oForm.Name = "NumberForm"
    oPage.Forms.insertByIndex(0, oForm)

    REM Das Modell wird dem ersten Formular in der Sammlung hinzugefügt.
    oForm.insertByIndex(0, oControlModel)
    oPage.add(oShape)
    'Aktiviert die Folie.
    oDrawDoc.CurrentController.setCurrentPage(oPage)
End Sub
```

Normale Formulare, solche wie vom Listing 537 produziert, bilden Gruppen von Formularkomponenten. Ein Datenformular kann jedoch mit einer Datenbank verknüpft sein und die Ergebnisse von SQL-Abfragen ausgeben. Ein HTML-Formular wiederum enthält Kontrollelemente, die für HTML-Seiten von Belang sind.

Tabelle 258. *In einem Formular benutzbare Kontrollkomponenten.*

Komponente	Beschreibung
CheckBox	Markierfeld.
ComboBox	Kombinationsfeld. Bietet Texteingabe oder Auswahl aus einer Liste von Textwerten.
CommandButton	Anklickbare Schaltfläche.
CurrencyField	Editierbares Währungsfeld.
DatabaseCheckBox	Datensensitives Markierfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseComboBox	Datensensitives Kombinationsfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseCurrencyField	Datensensitives editierbares Währungsfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseDateField	Datensensitives editierbares Datumsfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseFormattedField	Datensensitives formatiertes Textfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseImageControl	Feld zur Darstellung von Bildern, die in einer Datenbank gespeichert sind.
DatabaseListBox	Datensensitives Listenfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseNumericField	Datensensitives numerisches Editierfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabasePatternField	Datensensitives maskiertes Feld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseRadioButton	Datensensitives Optionsfeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseTextField	Datensensitives Texteingabefeld, das an ein Datenbankfeld gekoppelt sein kann.
DatabaseTimeField	Datensensitives Uhrzeitfeld, das an ein Datenbankfeld gekoppelt sein kann.
DateField	Editierbares Datumsfeld.

Komponente	Beschreibung
FileControl	Editierbare Dateiauswahl
FixedText	Textfeld, das vom Benutzer nicht editiert werden kann.
FormattedField	Formatiertes editierbares Textfeld.
GridControl	Tabellen-Kontrollfeld. Ausgabe von Daten in Tabellenform.
GroupBox	Gruppierungsrahmen zum visuellen Gruppieren von Kontrollelementen.
HiddenControl	Verborgenes Kontrollelement.
ImageButton	Grafik als klickbare Schaltfläche.
ListBox	Listenfeld. Bietet eine Auswahl aus einer Liste von Werten.
NumericField	Numerisches Editierfeld.
PatternField	Maskiertes Feld. Editierfeld mit Text, der zu einem Muster passt.
RadioButton	Optionsfeld.
TextField	Mehrzeiliges Texteingabefeld.
TimeField	Editierbares Uhrzeitfeld.

16.4. Präsentationen

Der Service com.sun.star.presentation.Presentation, der von einem Präsentationsdokument bereitgestellt wird, enthält Methoden (s. Tabelle 259) und Eigenschaften (s. Tabelle 260) zur Steuerung einer bestimmten Präsentation.

Tabelle 259. *Methoden im Service com.sun.star.presentation.Presentation.*

Methode	Beschreibung
start()	Die Präsentation startet von Anfang an im Vollbildmodus.
end()	Die Präsentation stoppt, der Vollbildmodus endet.
rehearseTimings()	Die Präsentation startet von Anfang an zusammen mit einer laufenden Uhr.

Sie können mehrere Präsentationsobjekte für unterschiedliche Präsentationstypen erstellen. Sie wünschen beispielsweise eine, die kontinuierlich zu einer Verkaufsmesse läuft, und eine andere, die manuell bedient wird, zum Beispiel für den Besuch bei einem Kunden. Die Methode getPresentation() des Dokuments gibt ein Präsentationsobjekt zurück. Nachdem Sie dessen Eigenschaften (Tabelle 260) festgelegt haben, nutzen Sie die Methoden start() und end() zum Starten und Beenden einer Präsentation. Die Methode rehearseTimings() startet eine Präsentation zusammen mit einer laufenden Uhr, um Ihnen zu helfen, die Dauer Ihrer Präsentation zu planen.

Tabelle 260. *Eigenschaften im Service com.sun.star.presentation.Presentation.*

Eigenschaft	Beschreibung
AllowAnimations	True = Animationen sind zugelassen.
CustomShow	Name einer individuellen Abfolge dieser Präsentation. Ein leerer String ist erlaubt.
FirstPage	Name der ersten Folie in der Präsentation. Ein leerer String ist erlaubt.
IsAlwaysOnTop	True = Das Präsentationsfenster ist immer im Vordergrund.
IsAutomatic	True = Folienwechsel erfolgen automatisch.
IsEndless	True = Die Präsentation läuft endlos als Schleife.
IsFullScreen	True = Die Präsentation läuft im Vollbildmodus.
IsLivePresentation	True = Die Präsentation läuft im Live-Modus.

Eigenschaft	Beschreibung
IsMouseVisible	True = Während der Präsentation ist der Mauszeiger sichtbar.
Pause	Dauer der Anzeige des schwarzen Bildschirms nach dem Ende der Präsentation. Long Integer.
StartWithNavigator	Der Navigator öffnet sich beim Start der Präsentation (True) oder nicht (False).
UsePen	True = Ein Zeichenstift erscheint während der Präsentation. Damit kann man auf dem Bildschirm zeichnen.

Listing 538. *Start der aktuellen Präsentation.*

```
Sub SimplePresentation()
  Dim oPres
  oPres = ThisComponent.getPresentation()
  oPres.UsePen = True
  REM Folgende Anweisung wird die Präsentation starten.
  REM Sie müssen die Leertaste zum Aufrufen der jeweils nächsten Folie drücken.
  oPres.start()
End Sub
```

Eine individuelle Präsentation kann die einzelnen Folien in jeder beliebigen Reihenfolge anzeigen. Folien können mehrfach oder auch gar nicht erscheinen. Die Methode getCustomPresentations() gibt ein Objekt zurück, das alle individuellen Präsentationen enthält (s. Tabelle 261).

Tabelle 261. *Einige Methoden im Interface XCustomPresentationSupplier.*

Methode	Beschreibung
createInstance()	Erzeugt eine individuelle Präsentation.
getByName(String)	Zugriff auf eine individuelle Präsentation über ihren Namen.
getElementNames()	Array der Namen der individuellen Präsentationen.
hasByName(String)	True = Die individuelle Präsentation mit diesem Namen existiert.
hasElements()	True = Die Seite enthält mindestens eine individuelle Präsentation.
insertByName(String, CustomPresentation)	Fügt eine individuelle Präsentation über ihren Namen ein.
removeByName(String)	Entfernt die genannte individuelle Präsentation.
replaceByName(String, CustomPresentation)	Ersetzt die genannte individuelle Präsentation.

Eine individuelle Präsentation ist ein Behälter für Folien, der die Interfaces XNamed und XIndexedAccess bereitstellt. Als erstes erzeugt man die individuelle Präsentation, dann fügt man die Folien in der Reihenfolge ein, in der sie erscheinen sollen, und schließlich wird die individuelle Präsentation gespeichert. Sie wird genau so dargestellt wie die normalen Präsentationen, nämlich über ein Presentation-Objekt. Das Attribut CustomShow zeigt jedoch auf die individuelle Präsentation.

Listing 539. *Erzeugung einer individuellen Präsentation.*

```
Sub CustomPresentation()
  Dim oPres   'Präsentationen, sowohl individuell wie normal
  Dim oPages  'Folien

  oPres  = ThisComponent.getCustomPresentations().createInstance()
  If Not ThisComponent.getCustomPresentations().hasByName("custom") Then
    oPages = ThisComponent.getDrawPages()

    REM Gezeigt werden die Folien 0, 2, 1, 0
    oPres.insertByIndex(0, oPages.getByIndex(0))
    oPres.insertByIndex(1, oPages.getByIndex(2))
    oPres.insertByIndex(2, oPages.getByIndex(1))
    oPres.insertByIndex(3, oPages.getByIndex(0))
    ThisComponent.getCustomPresentations().insertByName("custom", oPres)
```

```
    End If

    REM Nun starten wir die individuelle Präsentation.
    oPres = ThisComponent.getPresentation()
    oPres.CustomShow = "custom"
    oPres.Start()
End Sub
```

16.4.1. Präsentationsfolien

Folien in einem Präsentationsdokument unterscheiden sich ein wenig von denen in Zeichnungsdokumenten. Zusätzlich zu den Eigenschaften aus der Tabelle 237 steuern die Eigenschaften in der Tabelle 262, wie und wann Folien während einer Präsentation wechseln.

Tabelle 262. *Eigenschaften im Service com.sun.star.presentation.DrawPage.*

Eigenschaft	Beschreibung
Change	Auslöser für den Folienwechsel. Long Integer: 0 = Mausklick für die nächste Animation oder für den Folienwechsel. 1 = Automatischer Folienwechsel. 2 = Objektanimationen laufen automatisch, Folienwechsel aber durch Mausklick.
Duration	Anzeigedauer jeder einzelnen Folie in Sekunden, wenn die Eigenschaft Change auf 1 steht. Long Integer.
HighResDuration	Anzeigedauer jeder einzelnen Folie in Sekunden und Bruchteilen von Sekunden, wenn die Eigenschaft Change auf 1 steht. Typ Double.
Effect	Animation für den Folienübergang: Enumeration com.sun.star.presentation.FadeEffect (s. Tabelle 263).
Layout	Indexwert des Layouts der Präsentation, falls nicht null.
Speed	Geschwindigkeit der Übergangsänderung als Enumeration com.sun.star.presentation.AnimationSpeed: SLOW (langsam), MEDIUM (moderat), FAST (schnell).
IsHeaderVisible	True = ein Kopfzeilenobjekt der Masterfolie ist auf der Seite sichtbar.
HeaderText	Der Text des Kopfzeilenobjekts.
IsFooterVisible	True = ein Fußzeilenobjekt der Masterfolie ist auf der Seite sichtbar.
FooterText	Der Text des Fußzeilenobjekts.
IsPageNumberVisible	True = die Seitenzählung ist sichtbar.
IsDateTimeVisible	True = ein Datum/Uhrzeitobjekt der Masterfolie ist auf der Seite sichtbar.
IsDateTimeFixed	True = fixes Datum im Datum/Uhrzeitobjekt, False = aktuelles Datum im Datum/Uhrzeitobjekt.
DateTimeText	Der Textstring in einem durch IsDateTimeFixed = True als fix definierten Datum/Uhrzeitobjekt.
DateTimeFormat	Das Format zur Anzeige des Strings in einem durch IsDateTimeFixed = True als fix definierten Datum/Uhrzeitobjekt.

Die Seitenübergänge werden durch die Eigenschaft Effect der Präsentationsfolie gesteuert (s. Tabelle 263).

Tabelle 263. *Die Enumeration com.sun.star.presentation.FadeEffect.*

Wert	Beschreibung
NONE	Ohne Übergang.

Wert	Beschreibung
FADE_FROM_LEFT	Von links rollen.
FADE_FROM_TOP	Von oben rollen.
FADE_FROM_RIGHT	Von rechts rollen.
FADE_FROM_BOTTOM	Von unten rollen.
FADE_TO_CENTER	Von außen einblenden.
FADE_FROM_CENTER	Von innen einblenden.
MOVE_FROM_LEFT	Von links überdecken.
MOVE_FROM_TOP	Von oben überdecken.
MOVE_FROM_RIGHT	Von rechts überdecken.
MOVE_FROM_BOTTOM	Von unten überdecken.
ROLL_FROM_LEFT	Nach rechts schieben.
ROLL_FROM_TOP	Nach unten schieben.
ROLL_FROM_RIGHT	Nach links schieben.
ROLL_FROM_BOTTOM	Nach oben schieben.
VERTICAL_STRIPES	Vertikal blenden.
HORIZONTAL_STRIPES	Horizontal blenden.
CLOCKWISE	Im Uhrzeigersinn rollen.
COUNTERCLOCKWISE	Gegen den Uhrzeigersinn rollen.
FADE_FROM_UPPERLEFT	Diagonale Kästen nach rechts unten.
FADE_FROM_UPPERRIGHT	Diagonale Kästen nach links unten.
FADE_FROM_LOWERLEFT	Diagonale Kästen nach rechts oben.
FADE_FROM_LOWERRIGHT	Diagonale Kästen nach links oben.
CLOSE_VERTICAL	Horizontal(!) schließen (laut AOO 3.4.1 und LO 3.6!).
CLOSE_HORIZONTAL	Vertikal(!) schließen (laut AOO 3.4.1 und LO 3.6!).
OPEN_VERTICAL	Horizontal(!) öffnen (laut AOO 3.4.1 und LO 3.6!).
OPEN_HORIZONTAL	Vertikal(!) öffnen (laut AOO 3.4.1 und LO 3.6!).
SPIRALIN_LEFT	In einer Linksspirale von außen einblenden.
SPIRALIN_RIGHT	In einer Rechtsspirale von außen einblenden.
SPIRALOUT_LEFT	In einer Linksspirale von innen einblenden.
SPIRALOUT_RIGHT	In einer Rechtsspirale von innen einblenden.
DISSOLVE	Auflösen.
WAVYLINE_FROM_LEFT	Mit Wellenlinie nach rechts aufdecken.
WAVYLINE_FROM_TOP	Mit Wellenlinie nach unten aufdecken.
WAVYLINE_FROM_RIGHT	Mit Wellenlinie nach links aufdecken.
WAVYLINE_FROM_BOTTOM	Mit Wellenlinie nach oben aufdecken.
RANDOM	Automatisch (Zufall).
STRETCH_FROM_LEFT	In AOO 3.4.1 und LO 3.6 = FADE_FROM_LEFT.
STRETCH_FROM_TOP	In AOO 3.4.1 und LO 3.6 = FADE_FROM_TOP.
STRETCH_FROM_RIGHT	In AOO 3.4.1 und LO 3.6 = FADE_FROM_RIGHT.
STRETCH_FROM_BOTTOM	In AOO 3.4.1 und LO 3.6 = FADE_FROM_BOTTOM.
VERTICAL_LINES	Vertikale Linien.
HORIZONTAL_LINES	Horizontale Linien.

Wert	Beschreibung
MOVE_FROM_UPPERLEFT	Von links oben überdecken.
MOVE_FROM_UPPERRIGHT	Von rechts oben überdecken.
MOVE_FROM_LOWERRIGHT	Von rechts unten überdecken.
MOVE_FROM_LOWERLEFT	Von links unten überdecken.
UNCOVER_TO_LEFT	Nach links aufdecken.
UNCOVER_TO_UPPERLEFT	Nach links oben aufdecken.
UNCOVER_TO_TOP	Nach oben aufdecken.
UNCOVER_TO_UPPERRIGHT	Nach rechts oben aufdecken.
UNCOVER_TO_RIGHT	Nach rechts aufdecken.
UNCOVER_TO_LOWERRIGHT	Nach rechts unten aufdecken.
UNCOVER_TO_BOTTOM	Nach unten aufdecken.
UNCOVER_TO_LOWERLEFT	Nach links unten aufdecken.
VERTICAL_CHECKERBOARD	Horizontal(!) versetzt einblenden (laut AOO 3.4.1 und LO 3.6!).
HORIZONTAL_CHECKERBOARD	Vertikal(!) versetzt einblenden (laut AOO 3.4.1 und LO 3.6!).

Die Beschreibungen in der Tabelle 263 sind der Liste entnommen, die AOO 3.4.1 zur Auswahl im GUI bereitstellt. Ein paar wenige Effekte fehlen dort (zum Beispiel die WAVYLINE...-Effekte), allerdings finden sich auch einige zusätzliche nette Animationen.

Das folgende Makro setzt die Übergänge aller Folien auf RANDOM = zufällig.

Listing 540. *Setzt die Animationen der Folienübergänge auf RANDOM.*

```
Sub SetTransitionEffects()
  Dim oPages  'Folien
  Dim i%

  oPages = ThisComponent.getDrawPages()
  For i = 0 To oPages.getCount() - 1
    With oPages.getByIndex(i)
      .Effect = com.sun.star.presentation.FadeEffect.RANDOM
      .Change = 1
      .Duration = 2
      .Speed = com.sun.star.presentation.AnimationSpeed.FAST
    End With
  Next
End Sub
```

16.4.2. Formen für Präsentationen

Formen, die sich in Impress-Dokumenten befinden, unterscheiden sich von denen in Draw-Dokumenten dadurch, dass sie den Service com.sun.star.presentation.Shape unterstützen. Dieser Service stellt Eigenschaften zur Verfügung, die eigens zur Bereicherung der Darstellung von Präsentationen definiert sind (s. Tabelle 264).

Tabelle 264. *Eigenschaften im Service com.sun.star.presentation.Shape.*

Eigenschaft	Beschreibung
Bookmark	URL-String, für den Fall, dass die Eigenschaft OnClick einen URL benötigt.
DimColor	Farbe zum Abdunkeln der Form, wenn DimPrevious = True und DimHide = False ist.

Eigenschaft	Beschreibung
DimHide	True = Die Form wird verborgen, wenn zugleich DimPrevious = True ist.
DimPrevious	True = Die Form wird nach der Ausführung ihrer Animation abgedunkelt.
Effect	Animationseffekt für die Form (s. Tabelle 265)
IsEmptyPresentationObject	True = Dies ist das Standard-Präsentationsobjekt, und es ist leer.
IsPresentationObject	True = Dies ist ein Präsentationsobjekt. Präsentationsobjekte sind Objekte wie TitleTextShape und OutlinerShape.
OnClick	Eine spezifische Aktion, wenn der Benutzer auf diese Form klickt (s. Tabelle 266).
PlayFull	True = Der Sound dieser Form wird in voller Länge gespielt. Als Standard (False) wird der Sound gestoppt, wenn der Animationseffekt beendet ist.
PresentationOrder	Die Position dieser Form in der Reihenfolge der Formen, die auf der Folie animiert werden können, beginnend mit 1. Long Integer.
Sound	URL-String der Sounddatei, die während der Animation der Form abgespielt wird.
SoundOn	True = Während der Animation wird ein Sound abgespielt.
Speed	Effektive Dauer der Überblendung: Enumeration com.sun.star.presentation.AnimationSpeed: SLOW (langsam), MEDIUM (moderat), FAST (schnell).
TextEffect	Animationseffekt für den Text innerhalb dieser Form (s. Tabelle 265).
Verb	„ole2“-Verb, wenn die ClickAction-Eigenschaft von OnClick den Wert VERB hat (s. Tabelle 266).

Die von Formen unterstützten Animationseffekte (s. Tabelle 265) ähneln denen, die von Folien unterstützt werden (s. Tabelle 263), sind aber viel zahlreicher. Ich habe in der Tabelle 265 darauf verzichtet, die im GUI der aktuellen deutschsprachigen Versionen von AOO und LO aufgelisteten Bezeichnungen den Enumerationswerten gegenüberzustellen. Probieren Sie es selbst einmal aus.

Tabelle 265. *Die Enumeration com.sun.star.presentation.AnimationEffect.*

Wert	Wert	Wert
NONE	DISSOLVE	CLOCKWISE
RANDOM	APPEAR	COUNTERCLOCKWISE
PATH	HIDE	
MOVE_FROM_LEFT	MOVE_TO_LEFT	MOVE_SHORT_TO_LEFT
MOVE_FROM_TOP	MOVE_TO_TOP	MOVE_SHORT_TO_TOP
MOVE_FROM_RIGHT	MOVE_TO_RIGHT	MOVE_SHORT_TO_RIGHT
MOVE_FROM_BOTTOM	MOVE_TO_BOTTOM	MOVE_SHORT_TO_BOTTOM
MOVE_FROM_UPPERLEFT	MOVE_TO_UPPERLEFT	MOVE_SHORT_TO_UPPERLEFT
MOVE_FROM_UPPERRIGHT	MOVE_TO_UPPERRIGHT	MOVE_SHORT_TO_UPPERRIGHT
MOVE_FROM_LOWERRIGHT	MOVE_TO_LOWERRIGHT	MOVE_SHORT_TO_LOWERRIGHT
MOVE_FROM_LOWERLEFT	MOVE_TO_LOWERLEFT	MOVE_SHORT_TO_LOWERLEFT
MOVE_SHORT_FROM_LEFT	LASER_FROM_LEFT	STRETCH_FROM_LEFT
MOVE_SHORT_FROM_TOP	LASER_FROM_TOP	STRETCH_FROM_UPPERLEFT
MOVE_SHORT_FROM_RIGHT	LASER_FROM_RIGHT	STRETCH_FROM_TOP
MOVE_SHORT_FROM_BOTTOM	LASER_FROM_BOTTOM	STRETCH_FROM_UPPERRIGHT
MOVE_SHORT_FROM_UPPERLEFT	LASER_FROM_UPPERLEFT	STRETCH_FROM_RIGHT
MOVE_SHORT_FROM_UPPERRIGHT	LASER_FROM_UPPERRIGHT	STRETCH_FROM_LOWERRIGHT
MOVE_SHORT_FROM_LOWER-RIGHT	LASER_FROM_LOWERLEFT	STRETCH_FROM_BOTTOM
MOVE_SHORT_FROM_LOWERLEFT	LASER_FROM_LOWERRIGHT	STRETCH_FROM_LOWERLEFT

Wert	Wert	Wert
ZOOM_IN_FROM_LEFT ZOOM_IN_FROM_TOP ZOOM_IN_FROM_RIGHT ZOOM_IN_FROM_BOTTOM ZOOM_IN_FROM_CENTER ZOOM_IN_FROM_UPPERLEFT ZOOM_IN_FROM_UPPERRIGHT ZOOM_IN_FROM_LOWERRIGHT ZOOM_IN_FROM_LOWERLEFT	ZOOM_OUT_FROM_LEFT ZOOM_OUT_FROM_TOP ZOOM_OUT_FROM_RIGHT ZOOM_OUT_FROM_BOTTOM ZOOM_OUT_FROM_CENTER ZOOM_OUT_FROM_UPPERLEFT ZOOM_OUT_FROM_UPPERRIGHT ZOOM_OUT_FROM_LOWERRIGHT ZOOM_OUT_FROM_LOWERLEFT	FADE_FROM_LEFT FADE_FROM_TOP FADE_FROM_RIGHT FADE_FROM_BOTTOM FADE_FROM_CENTER FADE_FROM_UPPERLEFT FADE_FROM_UPPERRIGHT FADE_FROM_LOWERLEFT FADE_FROM_LOWERRIGHT FADE_TO_CENTER
ZOOM_IN ZOOM_IN_SMALL ZOOM_IN_SPIRAL ZOOM_OUT ZOOM_OUT_SMALL ZOOM_OUT_SPIRAL	VERTICAL_CHECKERBOARD HORIZONTAL_CHECKERBOARD HORIZONTAL_ROTATE VERTICAL_ROTATE HORIZONTAL_STRETCH VERTICAL_STRETCH	VERTICAL_STRIPES HORIZONTAL_STRIPES VERTICAL_LINES HORIZONTAL_LINES
WAVYLINE_FROM_LEFT WAVYLINE_FROM_TOP WAVYLINE_FROM_RIGHT WAVYLINE_FROM_BOTTOM	SPIRALIN_LEFT SPIRALIN_RIGHT SPIRALOUT_LEFT SPIRALOUT_RIGHT	CLOSE_VERTICAL CLOSE_HORIZONTAL OPEN_VERTICAL OPEN_HORIZONTAL

Formen in Präsentationsdokumenten erlauben besondere Aktionen (s. Tabelle 266) je nach dem Wert ihrer Eigenschaft OnClick (s. Tabelle 264).

Tabelle 266. *Die Enumeration com.sun.star.presentation.ClickAction.*

Wert	Beschreibung
NONE	Keine Aktion.
PREVPAGE	Sprung zur vorhergehenden Folie.
NEXTPAGE	Sprung zur folgenden Folie.
FIRSTPAGE	Sprung zur ersten Folie.
LASTPAGE	Sprung zur letzten Folie.
BOOKMARK	Sprung zu einer in der Eigenschaft Bookmark (s. Tabelle 264) festgelegten Sprungmarke.
DOCUMENT	Sprung zu einem in der Eigenschaft Bookmark (s. Tabelle 264) festgelegten anderen Dokument.
INVISIBLE	Das Objekt wird unsichtbar.
SOUND	Spielt einen in der Eigenschaft Bookmark (s. Tabelle 264) festgelegten Sound.
VERB	Ein OLE-Verb wird ausgeführt nach Maßgabe der Eigenschaft Verb (s. Tabelle 264). Ein OLE-Objekt unterstützt Aktionen, die „Verben“ genannt werden. Beispielsweise könnte ein OLE-Objekt, das einen Videoclip abspielt, das Verb „play“ unterstützen.
VANISH	Das Objekt verschwindet.
PROGRAM	Führt ein anderes in der Eigenschaft Bookmark (s. Tabelle 264) festgelegtes Programm aus.
MACRO	Führt ein anderes in der Eigenschaft Bookmark (s. Tabelle 264) festgelegtes Basic-Makro aus.
STOPPRESENTATION	Hält die Präsentation an.

16.5. Fazit

Impress- und Draw-Dokumente verfügen über zahlreiche Merkmale zur Unterstützung von Zeichnungen und Grafiken in Dokumenten. Beide Dokumenttypen haben viele dieser Merkmale gemeinsam. Impress-Dokumente erleichtern den Aufbau grafischer Präsentationen sowohl für den manuellen als auch den automatischen Ablauf. Zwar erlauben diese Dokumente die Darstellung von Bitmap-Bildern, doch ihre Stärke liegt eher in Vektorgrafiken als in Fotografien. Die Bandbreite der mit Impress- und Draw-Dokumenten möglichen Resultate reicht von einfach bis ziemlich komplex. Betrachten Sie dieses Kapitel nur als Ausgangspunkt für Ihre Erkundungen der Leistungsfähigkeit dieser beiden Dokumenttypen.

17. Verwaltung der Bibliotheken

In diesem Kapitel geht es nicht nur darum, wie und wo Makrobibliotheken gespeichert werden und mit welchen Methoden man sie bearbeiten kann, sondern auch um die Feinheiten der Makroverwaltung und um die Möglichkeit, über die UNO API Bibliotheken und Module zu bearbeiten.

Dieses Kapitel setzt voraus, dass Sie ein Grundwissen über Bibliothekscontainer, Bibliotheken und Module besitzen. Wenn Ihnen die folgende Übersicht fremd vorkommt, sehen Sie sich noch einmal das Material im Kapitel 2. Die Grundlagen an.

- Ein Bibliothekscontainer enthält keine, eine oder mehrere Bibliotheken.
- Eine Bibliothek enthält keine, eine oder mehrere Module und Dialoge.
- Ein Modul enthält keine, eine oder mehrere Subroutinen oder Funktionen.
- Die Anwendung ist ein Bibliothekscontainer. In der Anwendung gespeicherte Bibliotheken sind global für alle Makros erreichbar.
- Jedes Dokument ist ein Bibliothekscontainer.
- Die Bibliothek mit dem Namen Standard hat eine Sonderrolle: Sie ist immer vorhanden und kann nicht überschrieben werden. Mein Vorschlag ist, sie nicht zu nutzen.
- Geben Sie Ihren Bibliotheken, Modulen und Makros anschauliche Namen. „Bibliothek1“ oder „Modul4“ sagen so gar nichts aus, wohingegen „AXONRechnungsFormular1“ weitaus vielsagender und hilfreicher sein kann.

17.1. Zugriff auf Bibliotheken mit Basic

Einige meiner Makros hatte ich in einer Bibliothek auf der Anwendungsebene mit dem Namen „Pitonyak“ gebündelt. Als ich am nächsten Tag OOo startete, stellte ich zu meiner Überraschung fest, dass ich keins der Makros in meiner neuen Bibliothek nutzen konnte. Es stellte sich heraus, dass man Bibliotheken vor der Benutzung laden muss.

Tipp	Eine Bibliothek muss geladen werden, danach erst ist sie erreichbar.

Manuell laden Sie eine Bibliothek, indem Sie den Dialog **Extras | Makros | Makros verwalten | Basic** öffnen und auf die gewünschte Bibliothek doppelklicken. Sie können die Aktivierung an der Änderung des zugehörigen Symbols erkennen. Der Zugang zu Bibliotheken ist auch in Makros möglich. Basic stellt die Variable GlobalScope zur Kontrolle von Bibliotheken auf Anwendungsebene zur Verfügung (s. Listing 541).

Listing 541. *Mit GlobalScope werden Bibliotheken auf Anwendungsebene geladen.*

```
GlobalScope.BasicLibraries.loadLibrary("Pitonyak")
```

Tipp	Die Variable GlobalScope unterstützt nicht die „dbg“-Eigenschaften. Das heißt, Sie können die Variable GlobalScope nicht inspizieren.

Bibliotheken enthalten zweierlei, Dialoge und Makros. Die Variable GlobalScope hat zwei Eigenschaften – BasicLibraries und DialogLibraries – für den Zugang zu den Basic- und Dialog-Bibliothekscontainern im Bibliothekscontainer der Anwendung. Die Eigenschaften BasicLibraries und DialogLibraries unterstützen beide denselben Satz an Interfaces zum Zugriff auf die enthaltenen Bibliothekscontainer (s. Tabelle 267).

Tabelle 267. *Von Bibliothekscontainer-Objekten unterstützte Methoden.*

Methode	Beschreibung
createLibrary(Name)	Erzeugt eine neue Bibliothek mit diesem Namen.
createLibraryLink(Name, Url, ReadOnly)	Erzeugt einen Link zu einer „externen“ Bibliothek. Name und URL sind Strings. ReadOnly = True: die Bibliothek kann nicht bearbeitet werden.
removeLibrary(Name)	Entfernt die Bibliothek dieses Namens. Wenn die Bibliothek ein Link ist, wird nur der Link entfernt, nicht die Bibliothek selbst.
isLibraryLoaded(Name)	True = die Bibliothek ist geladen, False = sie ist nicht geladen.
loadLibrary(Name)	Lädt eine Bibliothek, falls sie nicht schon geladen ist.
isLibraryLink(Name)	True = die Bibliothek ist ein Link zu einer anderen Bibliothek.
getLibraryLinkURL(Name)	Gibt den URL einer verlinkten Bibliothek zurück. Falls die Bibliothek kein Link ist, wird ein Fehler erzeugt.
isLibraryReadOnly(Name)	True = die Bibliothek ist schreibgeschützt.
setLibraryReadOnly(Name, ReadOnly)	Setzt den Schreibschutz der genannten Bibliothek (ReadOnly = True) oder hebt ihn auf (ReadOnly = False).
renameLibrary(Name, NewName)	Ändert den Namen einer Bibliothek. Wenn die Bibliothek verlinkt ist, wird nur der Link umbenannt.
changeLibraryPassword(Name, Pass, NewPass)	Ändert das Passwort einer Bibliothek.
getByName(Name)	Gibt die genannte Bibliothek zurück.
getElementNames()	Gibt ein Array von Bibliotheksnamen zurück.
hasByName(Name)	True = die genannte Bibliothek existiert.
hasElements()	True = mindestens eine Bibliothek existiert.
isLibraryPasswordProtected(Name)	True = die Bibliothek ist passwortgeschützt.
isLibraryPasswordVerified(Name)	True = das Passwort ist schon zum Öffnen der Bibliothek verwendet worden.
verifyLibraryPassword(Name, Pass)	Öffnet eine passwortgeschützte Bibliothek.

Tipp Die Variable GlobalScope ist außerhalb von Basic nicht verfügbar. Man kann jedoch den nicht dokumentierten UNO-Service com.sun.star.script.ApplicationScriptLibraryContainer erzeugen und nutzen, um Zugriff auf die globalen Basic-Bibliotheken zu erhalten.

Innerhalb benannter Container sind die enthaltenen Bibliotheken als XML-Strings gespeichert. Tabelle 268 zeigt die von Bibliotheksobjekten unterstützten Methoden.

Tabelle 268. *Von Bibliotheksobjekten unterstützte Methoden.*

Methode	Beschreibung
getByName(Name)	Gibt das genannte Modul als String zurück.
getElementNames()	Gibt ein Array der Modulnamen zurück.
hasByName(Name)	True = die Bibliothek enthält das genannte Modul.
hasElements()	True = die Bibliothek enthält mindestens ein Modul.
insertByName(Name, Modul)	Fügt das genannte Modul in die Bibliothek ein.
removeByName(Name)	Entfernt das genannte Modul.
replaceByName(Name, Modul)	Ersetzt das genannte Modul.

Wenn Sie eine Bibliothek über die Makroverwaltung erstellen, wird automatisch sowohl eine Dialogbibliothek als auch eine Basic-Bibliothek erzeugt. Diese neue Basic-Bibliothek enthält ein Modul mit

dem Namen „Module1“, das seinerseits eine leere Subroutine mit dem Namen „Main“ enthält. Die folgenden Schritte verdeutlichen den Prozess:

1. Öffnen des Makrodialogs über **Extras | Makros | Makros verwalten | Basic**.
2. Klick auf die Schaltfläche **Verwalten...**, um den Dialog *Basic Makros verwalten* zu öffnen. Diesen Dialog kann man auch über **Extras | Makros | Dialoge verwalten...** öffnen.
3. Klick auf den Reiter *Bibliotheken*.
4. Klick auf die Schaltfläche **Neu...**. Geben Sie den Bibliotheksnamen „TestLib“ ein (s. Bild 138).
5. Klick auf **OK**, um den Dialog *Neue Bibliothek* zu schließen.
6. Klick auf die Schaltfläche **Schließen** im Dialog *Basic Makros verwalten*.
7. Klick auf die Schaltfläche **Schließen** im Dialog *Basic Makros*.

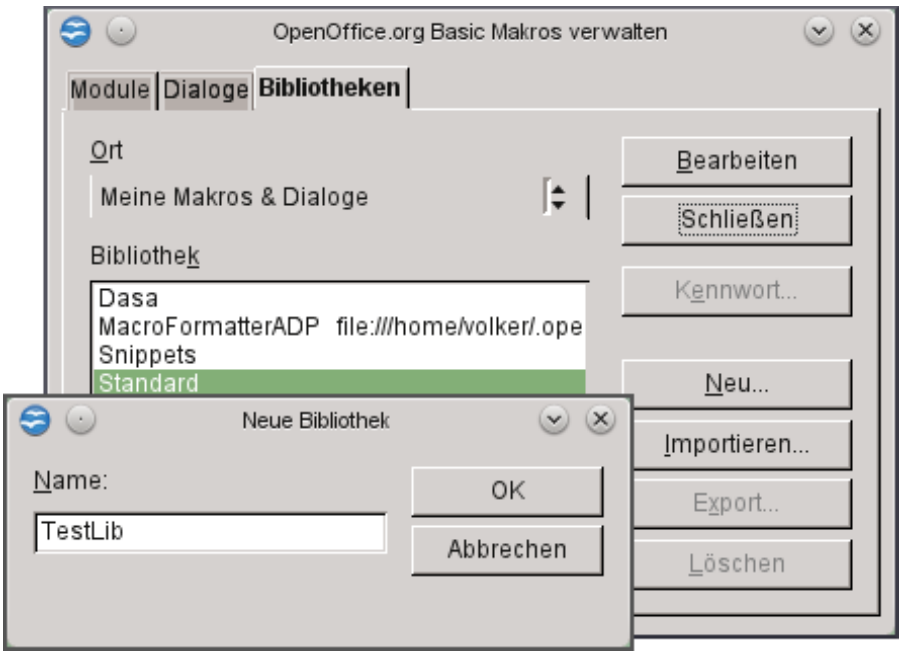


Bild 138. *Eine neue Bibliothek im Bibliothekscontainer der Anwendung.*

Den Inhalt der Bibliothek TestLib können Sie mit dem Makro im Listing 542 anzeigen. Das Bild 139 zeigt Ihnen, dass TestLib das Module1 mit einer einzigen Subroutine enthält.

Listing 542. *Ausgabe von Module1 in der neu geschaffenen Bibliothek TestLib.*

```
Sub PrintTestLib
  If GlobalScope.BasicLibraries.hasByName("TestLib") Then
    Dim oLib
    oLib = GlobalScope.BasicLibraries.getByName("TestLib")
    MsgBox oLib.getByName("Module1"), 0, "Module1 in TestLib"
  Else
    MsgBox "Bibliothek 'TestLib' existiert nicht"
  End If
End Sub
```

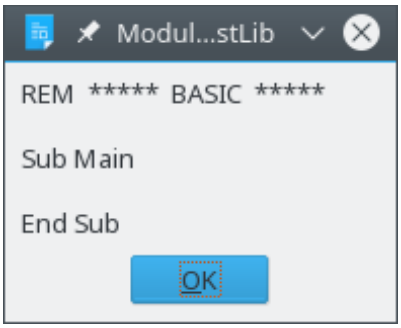


Bild 139. *Module1 wird vom Verwalten-Dialog automatisch angelegt.*

Sie können mit dem Code im Listing 543 überprüfen, dass sowohl eine Basic- als auch eine Dialogbibliothek existieren. Doch obwohl es die Dialogbibliothek gibt, enthält sie doch keine Dialoge. Wenn die Bibliothek TestLib nicht existieren sollte, hat das zurückgegebene Objekt den Wert Null.

Listing 543. *Zugriff sowohl auf die Basic- als auch auf die Dialogbibliothek.*

```
oLib = GlobalScope.BasicLibraries.getByName("TestLib")
oLib = GlobalScope.DialogLibraries.getByName("TestLib")
```

Tipp In Bibliotheken, die mit den Methoden in der Tabelle 268 erzeugt wurden, werden weder eine Basic- noch eine Dialogbibliothek automatisch eingefügt.

Über die UNO API (Tabelle 268) kann nur jeweils eine Bibliothek erzeugt werden. Das Makro im Listing 544 erstellt eine Dialogbibliothek sowie eine zugehörige Basic-Bibliothek, fügt der Makrobibliothek das Module1 und schließlich diesem Modul die Subroutine Main hinzu (s. Bild 140).

Listing 544. *Erzeugung einer globalen Bibliothek.*

```
Sub CreateAGlobalLib
  Dim oLib
  Dim s$
  If GlobalScope.BasicLibraries.hasByName("TestLib") Then
    GlobalScope.BasicLibraries.removeLibrary("TestLib")
    GlobalScope.DialogLibraries.removeLibrary("TestLib")
    MsgBox "TestLib gelöscht"
  Else
    GlobalScope.BasicLibraries.createLibrary("TestLib")
    GlobalScope.DialogLibraries.createLibrary("TestLib")
    oLib = GlobalScope.BasicLibraries.getByName("TestLib")
    s = "Sub Main" & Chr$(10) &_
      "  x = x + 1" & Chr$(10) &_
      "End Sub"
    oLib.insertByName("Module1", s)
    s = "=== Basic-Bibliotheken ===" & Chr$(10)
    s = s & Join(oLib.getElementNames(), Chr$(10))
    oLib = GlobalScope.DialogLibraries.getByName("TestLib")
    s = s & Chr$(10) & Chr$(10) & "=== Dialogbibliotheken ===" & Chr$(10)
    s = s & Join(oLib.getElementNames(), Chr$(10))
    MsgBox s, 0, "Module in der Bibliothek TestLib"
    oLib = GlobalScope.BasicLibraries.getByName("TestLib")
    MsgBox oLib.getByName("Module1"), 0, "Module1 in TestLib"
  End If
End Sub
```

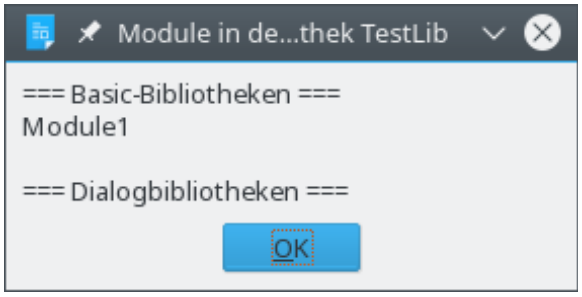


Bild 140. *Eine existierende Dialogbibliothek, die keine Dialoge enthält.*

17.2. Bibliotheken in einem Dokument

Basic bietet die Variablen BasicLibraries und DialogLibraries zum Zugriff auf Bibliotheken, die sich in einem Dokument befinden. Sie verhalten sich genauso wie ihre GlobalScope-Kollegen, außer dass sie auf den Bibliothekscontainer für das aktuelle Dokument zugreifen. Wenn Sie Listing 544 so verändern, dass Sie alle Vorkommen von GlobalScope entfernen, wird sich das Makro auf das aktuelle Dokument beziehen.

Die Variablen BasicLibraries und DialogLibraries stellen eine ausgezeichnete Methode dar, auf Dialoge und Makrobibliotheken im Dokument zuzugreifen. Dennoch findet man immer noch Code, der die veraltete Methode getLibraryContainer() nutzt. Jedes Dokument unterstützt diese Methode. Das zurückgegebene Objekt unterstützt die Methode getModuleContainer(), die die enthaltenen Basic-Module zurückgibt, und die Methode getDialogContainer(), die nach meinen Tests immer ein Null-Objekt zurückgibt. Die Tabelle 269 zeigt, wie man ein Modul auf die neue und die alte Art erhält.

Tabelle 269. *Zugriff auf das Modul einMod aus der Bibliothek TestLib eines Dokuments.*

Verwendet BasicLibraries	Verwendet oDoc.getLibraryContainer()

`oLibs = BasicLibraries`
`If oLibs.hasByName("TestLib") Then`
`  oLib = oLibs.getByName("TestLib")`
`  If oLib.hasByName("einMod") Then`
`  oMod = oLib.getByName("einMod")`
`  End If`
`End If`
	`oLibs = oDoc.getLibraryContainer()`
	`If oLibs.hasByName("TestLib") Then`
	`  oLib = oLibs.getByName("TestLib")`
	`  oMods = oLib.getModuleContainer()`
	`  If Not IsNull(oMods) Then`
	`  If oMods.hasByName("einMod") Then`
	`  oMod = oMods.getByName("einMod")`
	`  End If`
	`  End If`
	`End If`

Tipp Die Variablen BasicLibraries und DialogLibraries stehen nur dann zur Verfügung, wenn sich die gerade aktive Subroutine im Dokument befindet. Leider gibt es keinen anderen Weg, einen Dialog aus einem Dokument aufzurufen. Es muss aus dem Code heraus erfolgen, der im Dokument liegt. Schreiben Sie also zur Erzeugung eines Dialogs eine Funktion, die sich in demselben Dokument befindet wie der besagte Dialog.

17.3. Eine Installationsroutine

Ich will nicht so weit gehen, eine vollwertige Installationsroutine zu schreiben, aber ein paar Hinweise sollen es doch sein. Das Makro im Listing 545 kopiert eine einzelne Bibliothek, deren Name beim Kopieren geändert werden kann. Der Vorgang ist einfach und geradeheraus. Die Module der Quellbibliothek werden nacheinander in die Zielbibliothek kopiert. Das letzte Argument bestimmt, ob die Zielbibliothek gelöscht wird, bevor der Kopierprozess beginnt. Denn wenn die Zielbibliothek nicht erst gelöscht wird, dann werden die Module, die sich nicht in der Quellbibliothek befinden, nach dem Kopieren weiterhin in der Zielbibliothek enthalten sein.

Listing 545. *Kopie einer Bibliothek.*

```
REM sSrcLib ist der Name der Quellbibliothek im Container oSrcLibs.
REM sDestLib ist der Name der Zielbibliothek im Container oDestLibs.
REM oSrcLibs ist der Quellbibliothekscontainer.
REM oDestLibs ist der Zielbibliothekscontainer.
REM bClearDest = True: die Zielbibliothek wird gelöscht.
Sub AddOneLib(sSrcLib$, sDestLib$, oSrcLibs, oDestLibs, bClearDest As Boolean)
  Dim oSrcLib   'Die zu kopierende Quellbibliothek
  Dim oDestLib  'Die Zielbibliothek, die die Module aus oSrcLib erhalten soll
  Dim sNames
  Dim i%

  REM Wenn es keine Zielbibliothek gibt, wird die Routine einfach beendet.
  If IsNull(oDestLibs) Or IsEmpty(oDestLibs) Then
    Exit Sub
  End If

  REM Falls gewünscht, wird die Zielbibliothek gelöscht.
  If bClearDest And oDestLibs.hasByName(sDestLib) Then
    oDestLibs.removeLibrary(sDestLib)
  End If

  REM Wenn es keine Quellbibliothek gibt, ist nichts weiter zu tun.
  If IsNull(oSrcLibs) Or IsEmpty(oSrcLibs) Then
    Exit Sub
  End If

  REM Wenn die Quellbibliothek nicht existiert, gibt es auch nichts zu tun.
  If Not oSrcLibs.hasByName(sSrcLib) Then
    Exit Sub
  End If

  REM Wenn die Zielbibliothek nicht existiert, dann wird sie erzeugt.
  If Not oDestLibs.hasByName(sDestLib) Then
    oDestLibs.createLibrary(sDestLib)
  End If

  REM Los geht's mit dem Vergnügen.
  REM Auch wenn es offensichtlich erscheint, aber die Bibliotheken
  REM müssen erst geladen werden.
  REM Es wird häufig der Fehler begangen, die Bibliotheken nicht erst zu laden.
  oSrcLibs.loadLibrary(sSrcLib)
  oDestLibs.loadLibrary(sDestLib)

  REM Quell- und Zielbibliothek werden geholt,
  REM sowie alle enthaltenen und zu kopierenden Module.
  oSrcLib = oSrcLibs.getByName(sSrcLib)
  oDestLib = oDestLibs.getByName(sDestLib)
  sNames = oSrcLib.getElementNames()

  REM Die Module werden nacheinander entweder hinzugefügt oder ersetzt.
  For i = LBound(sNames) To UBound(sNames)
    If oDestLib.hasByName(sNames(i)) Then
      oDestLib.replaceByName(sNames(i), oSrcLib.getByName(sNames(i)))
    Else
      oDestLib.insertByName(sNames(i), oSrcLib.getByName(sNames(i)))
    End If
```

```
    Next
End Sub
```

Angenommen, Sie wollen eine bestimmte Bibliothek aus der Anwendung in ein Dokument kopieren, so dass Sie sie einem Freund zusenden können. Am schnellsten geht es, wenn Sie die Bibliothek aus der Anwendung über die Makroverwaltung dem Dokument hinzufügen. Natürlich können Sie ein Makro schreiben, aber manches geht manuell doch schneller. Wenn Sie jedoch häufig Bibliotheken kopieren, ist es sinnvoll, die Arbeit von einem Makro machen zu lassen. Das Makro im Listing 546 erwartet den Namen einer Bibliothek, die sich in der Anwendung befindet, und kopiert sie in das aktuelle Dokument. Es werden sowohl die Basic- als auch die Dialogbibliothek kopiert. Das Makro für den umgekehrten Vorgang, nämlich eine Bibliothek aus dem Dokument heraus in die Anwendung zu übertragen, ist trivial und nahezu identisch mit Listing 546.

Listing 546. *Übertragung einer globalen Bibliothek in das aktuelle Dokument.*

```
Sub AppLibToDocLib(sLibName$)
  Dim oGlobalLib
  oGlobalLib = GlobalScope.BasicLibraries
  AddOneLib(sLibName, sLibName, oGlobalLib, BasicLibraries, True)
  oGlobalLib = GlobalScope.DialogLibraries
  AddOneLib(sLibName, sLibName, oGlobalLib, DialogLibraries, True)
End Sub
```

17.4. Fazit

Obwohl es manchmal schneller geht, Bibliotheken manuell zu bearbeiten, ist es dennoch einfach, sie mit Makros zu bearbeiten und zu kopieren. Mit den in diesem Kapitel vorgestellten Methoden werden Sie die meisten Aufgaben im Zusammenhang mit Bibliotheken erledigen können.

18. Dialoge und Steuerelemente

In diesem Kapitel geht es darum, wie Dialoge und die darin enthaltenen Steuerelemente erstellt und genutzt werden. Als Hauptmethode zum Erstellen von Dialogen wird die Basic-Entwicklungsumgebung (IDE) beschrieben. Sie werden jedes der verschiedenen Steuerelemente kennenlernen und Beispiele für die meisten davon. Ich werde Ihnen auch eine Methode vorstellen, wie man zur Laufzeit – also ohne die Basic-IDE – Dialoge und Steuerelemente aufbauen kann.

In der Computerterminologie ist ein Fenster eine Fläche auf dem Bildschirm, in dem gewisse Informationen angezeigt werden. In den meisten Anwendungen ist es nicht immer leicht, den Unterschied zwischen einem Dialog und einem Fenster auszumachen. Obwohl das Wort „Dialog“ unterschiedlich definiert sein kann, so bezeichnet es doch gewöhnlich einen Informationsaustausch zwischen zwei oder mehreren Funktionseinheiten – normalerweise Menschen. In OpenOffice ist ein Dialog ein Fenster, das zur Interaktion mit einem Benutzer dient.

Ein „modales“ Fenster blockiert weitere Aktivitäten der Hauptanwendung (oder des Ausgangsfensters) so lange, bis es seinen Programmteil vollständig ausgeführt hat und erfolgreich geschlossen wurde. Wenn Sie also ein „modales“ Fenster öffnen, erhält es den Fokus, und Sie können es nicht einfach zur Seite schieben, um das Fenster darunter zu benutzen. Wenn ein Fenster modal ist, wird es im allgemeinen als Dialog betrachtet.

Tipp Während der Anzeige eines modalen Dialogs können Sie zwar auf den Dialog, nicht aber auf das Dokumentfenster zugreifen. Dialoge sind fast immer modal, wohingegen allgemeine Anwendungsfenster, also solche, die Dokumente zur Bearbeitung durch den Anwender anzeigen, fast niemals modal sind.

Der Dialog „Suchen & Ersetzen“ in einem Textdokument ist eines der wenigen Beispiele eines nicht modalen Dialogs. Nachdem Sie diesen Dialog geöffnet haben, können Sie immer noch schreibend auf das Dokument zugreifen. Ein anderes Kriterium für einen Dialog basiert auf seiner Funktion. Das Hauptanzeigefenster einer Anwendung ist normalerweise kein Dialog. Die Information, die in einem Dialog ausgegeben oder angefragt wird, ist gemeinhin der Programmfunktion untergeordnet. Zum Beispiel werden häufig Informationen über Einstellungen und Fehlermeldungen in Dialogen ausgegeben, aber nicht der Haupttext eines Writer-Dokuments. Obwohl der Unterschied zwischen Fenster und Dialog letztlich willkürlich ist, so können Sie mit Basic zwar Ihre eigenen Dialoge, aber keine allgemeinen Anwendungsfenster bauen und ausführen. Das hat seinen Grund, denn der übliche Zweck eines Dialogs besteht darin, eine Verständigung über einzelne Inhaltsteile zu ermöglichen, die wichtig zur Funktion einer bereits aktiven Anwendung, eines Objekts oder eines Dokuments sind.

18.1. Mein erster Dialog

Bevor Sie in der IDE einen Dialog aufbauen können, müssen Sie erst einen leeren Dialog erzeugen. Es gibt zwei Methoden, einer Bibliothek einen leeren Dialog hinzuzufügen. Eine davon geht über den Dialog der Makroverwaltung, mit dem Vorteil, dem neuen Dialog sofort einen sprechenden Namen geben zu können.

Die andere Methode zur Erzeugung von Dialogen und Modulen geht über einen Rechtsklick auf einen Modulenreiter direkt in der Basic-IDE. Die Reiter am unteren Rand der IDE zeigen die geöffneten Module und Dialoge. Nach einem Rechtsklick auf irgendeinen dieser Reiter wählen Sie aus dem Kontextmenü den Dialog **Einfügen | Basic-Dialog** (s. Bild 141).

Es wird ein neuer Dialog mit dem Namen Dialog1 erzeugt. Rechtsklicken Sie auf diesen Reiter und benennen Sie ihn über **Umbenennen** in HelloDlg um. Mitten in der IDE-Seite erscheint ein noch leerer Dialog. Unter dem Namen HelloDlg werden Sie ihn anzeigen können. Um ihn zur Bearbeitung zu markieren, klicken Sie auf eine Stelle der Umrandung. Danach können Sie seine Größe und Lage mit der Maus ändern. Hätten Sie nicht auf den Rand, sondern woanders hin geklickt, wäre es schwieriger geworden, die nötigen Steuerelemente für diesen Dialog auszuwählen.

Tipp Wenn etwas markiert ist, erscheinen grüne Kästchen rund um das ausgewählte Element. Diese grünen Kästchen heißen Griffe.

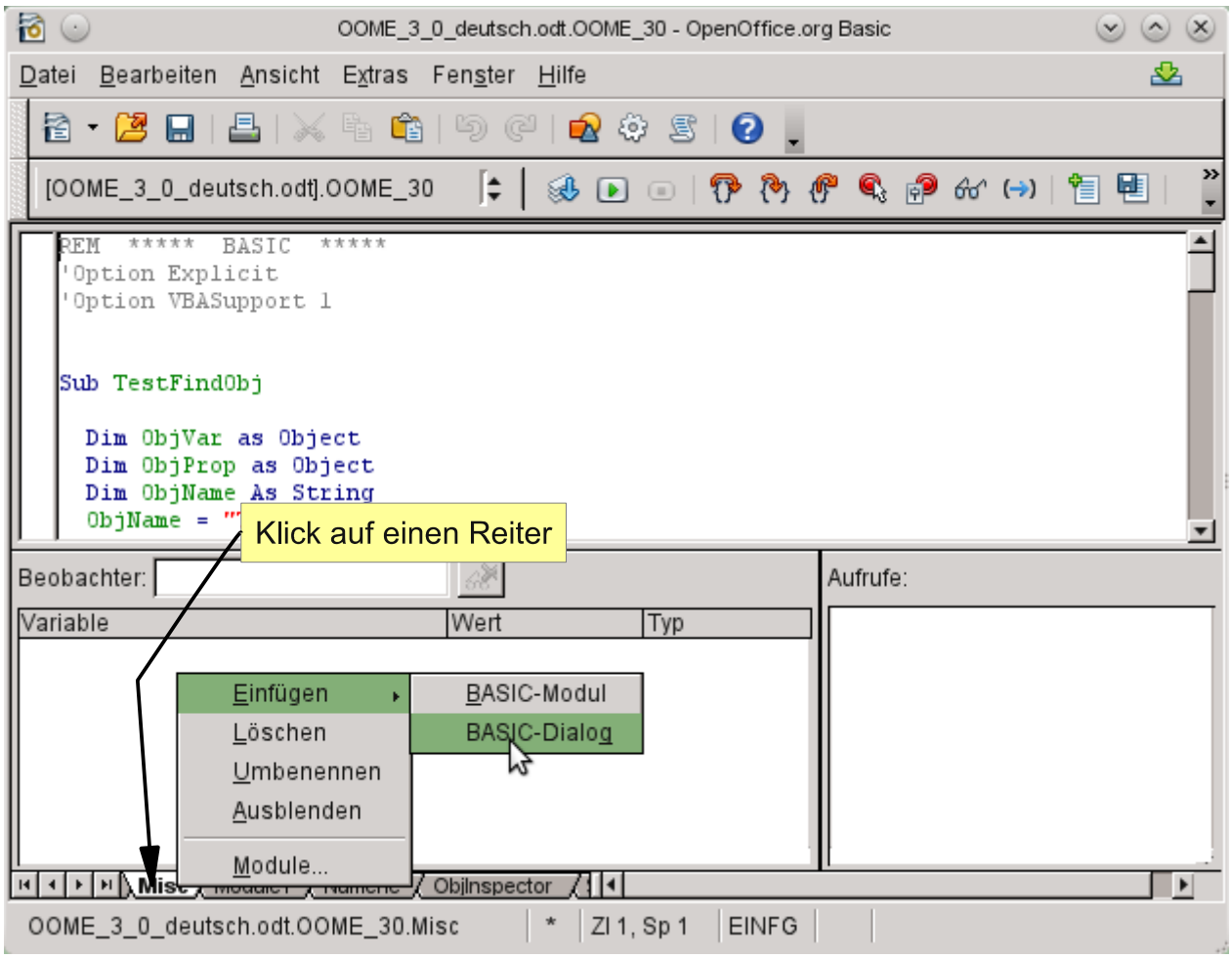


Bild 141. *Einfügen eines neuen Dialogs über die IDE: Rechtsklick auf einen Modulreiter und Einfügen | Basic-Dialog auswählen.*

Ziehen Sie die Werkzeugleiste in die Arbeitsfläche. Falls sie noch nicht angezeigt wird, schalten Sie die Anzeige über **Ansicht | Symbolleisten | Werkzeugleiste** ein. Sie können auch ein Drop-Down-Menü mit Klick auf das entsprechende Symbol der Dialogleiste aufklappen. Die Werkzeugleiste ist nicht modal, also können Sie sie geöffnet lassen und dennoch den Dialog bearbeiten. Wenn Sie mit der Maus über die einzelnen Symbole fahren, wird Ihnen der Name des Werkzeugs angezeigt. Klicken Sie nun auf das Symbol *Testmodus ein/aus*, um eine Kopie des Dialogs im Testmodus anzuzeigen. So sehen Sie, wie der Dialog in Wirklichkeit aussieht. Um den Dialog im Testmodus wieder zu schließen, klicken Sie auf das Schließen-Kreuzchen in der oberen rechten Ecke des Dialogs.

Tipp Die Werkzeugleiste wird in verschiedenen Versionen von AOO und LO unterschiedlich benannt und unterscheidet sich auch im Aussehen und im Inhalt.

Die meisten der Symbole in der Werkzeugleiste dienen dazu, Ihrem Dialog Steuerelemente hinzuzufügen. Wenn Sie auf eines dieser Symbole klicken, ändert sich die Form des Mauscursors vom Pfeil in ein Kreuz. Sie kehren zum pfeilförmigen Auswahlcursor zurück, indem Sie statt eines neuen Steuerelements das Auswahlsymbol anklicken (s. Tabelle 270).

Tabelle 270. *Steuerelemente eines Dialogs.*

AOO	LO	Beschreibung
		Auswahl.
		Sprache verwalten.
		Testmodus ein/aus.
		Eigenschaften.
		Schaltfläche.
		Grafisches Kontrollfeld.
		Markierfeld.
		Optionsfeld.
		Beschriftungsfeld.
		Textfeld.
		Listenfeld.
		Kombinationsfeld.
		Vertikale Bildlaufleiste.
		Horizontale Bildlaufleiste.
		Gruppierungsrahmen.
		Fortschrittsbalken.
		Horizontale Linie.
		Vertikale Linie.
		Datumsfeld.
		Zeitfeld.
		Numerisches Feld.
		Währungsfeld.
		Formatiertes Feld.
		Maskiertes Feld.
		Dateiauswahl.

AOO	LO	Beschreibung
		Baumansicht-Steuerelement.
		Drehfeld. Dieses Symbol fehlt standardmäßig in der AOO-Werkzeugleiste und ist auch wohl nicht in einem AOO-Dialog nutzbar (glaube ich).

Tipp Es gibt noch andere Steuerelemente, sie werden aber standardmäßig nicht in der Werkzeugleiste angezeigt, beispielsweise die Datenbankfelder. Mit Recht, denn man kann sie in einem Dialog nicht nutzen.

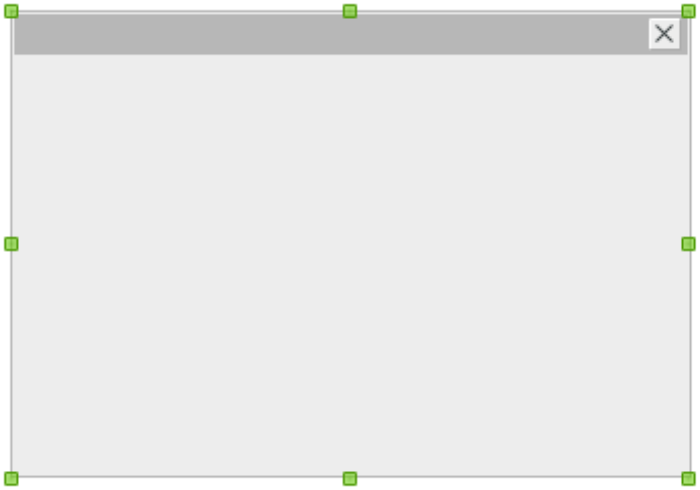


Bild 142. *Klick auf den Rahmen des Dialogs, um die grünen Griffe einzuschalten.*

18.1.1. Der Eigenschaften-Dialog

Vieles von dem, was ein Dialog oder ein Steuerelement tut, wird von Eigenschaften gesteuert, deren Wert man sowohl bei der Designgestaltung als auch zur Laufzeit setzen kann. Zur Bearbeitung während des Entwurfs markieren Sie als erstes das Objekt. Danach rechtsklicken Sie auf das Objekt und wählen **Eigenschaften** aus dem Kontextmenü. Alternativ können Sie auch auf das Eigenschaften-Symbol in der Werkzeugleiste klicken. Bild 143 zeigt den Dialog.

Die Eigenschaft Name bestimmt den Namen, mit dem das Steuerelement vom Dialog aufgerufen wird. Der Name des Dialogs selbst muss aber nicht unbedingt geändert werden, denn der Dialog wird nicht auf dieselbe Weise aufgerufen wie ein Steuerelement. Für Dialoge (und zwar nur für Dialoge) gibt es die Eigenschaft Titel. Sie enthält die Titelzeile des Dialogs – für mein Beispiel habe ich „Hallo Welt“ gewählt. Obwohl der Eigenschaften-Dialog die Möglichkeit zur Festlegung von Position und Größe bietet, macht man das meistens mit der Maus.

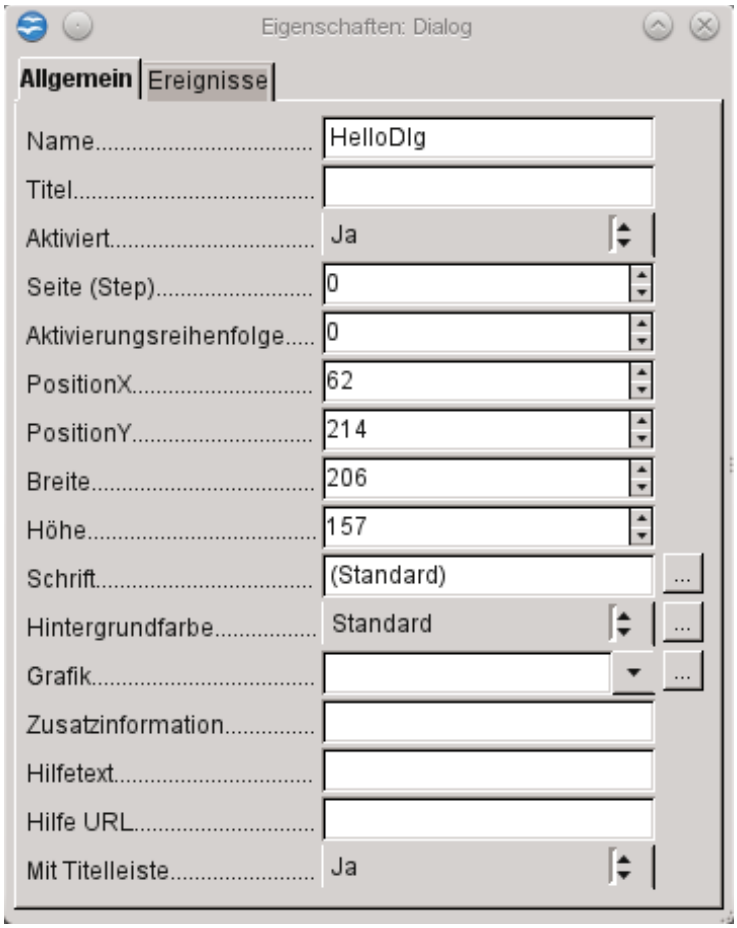


Bild 143. *Die Eigenschaften eines Dialogs.*

Tipp Die im Eigenschaften-Dialog gezeigten Eigenschaften sind abhängig vom markierten Objekt.

Gehen Sie Schritt für Schritt vor, um den Beispieldialog zu gestalten:

1. Öffnen Sie die Werkzeugleiste über **Ansicht | Symbolleisten | Werkzeugleiste**.
2. Markieren Sie den Dialog mit Klick auf dessen äußeren Rahmen (s. Bild 142).
3. Öffnen Sie den Eigenschaften-Dialog mit Klick auf das entsprechende Symbol in der Dialogleiste (s. Bild 143).
4. Geben Sie dem Dialog den Namen HelloDlg – ich jedenfalls gebe Dingen gerne einen Namen.
5. Geben Sie dem Dialog den Titel „Hallo Welt“. Wenn der Cursor das Titelfeld verlässt, wird der Titel in der Titelzeile des Dialogs angezeigt.
6. Verschieben Sie den Eigenschaften-Dialog so, dass er nicht den Hallo-Welt-Dialog überdeckt.
7. Fügen Sie dem Dialog eine Schaltfläche hinzu. Dazu klicken Sie in der Werkzeugleiste auf das Symbol für Schaltfläche. Wenn sich der Mauscursor über dem Dialog befindet, ändert sich sein Aussehen in ein Kreuz. Positionieren Sie den Cursor dorthin, wo Sie die Schaltfläche haben wollen, klicken Sie dann und ziehen die Fläche zur passenden Größe auf.
8. Wenn die Schaltfläche erzeugt ist, erhält sie standardmäßig den Namen CommandButton1. Im Eigenschaften-Dialog ändern Sie dann den Namen der Schaltfläche in „OKButton“ und geben Sie ihr den Titel „OK“.

9. Im Eigenschaften-Dialog ändern Sie die Art der Schaltfläche zu OK. Somit wird ein Klick auf die Schaltfläche den Dialog automatisch schließen. Sollten Sie das nicht tun, wird die Schaltfläche keinerlei Wirkung zeigen, es sei denn, Sie registrieren einen Handler dafür – Ereigniszuordnungen werden über den Reiter „Ereignisse“ des Eigenschaften-Dialogs eingestellt.
10. Als Hilfetext geben Sie „Hier klicken“ ein, so dass dieser Text als Hilfe angezeigt wird, wenn der Cursor über der Schaltfläche schwebt.

Tipp OOo stellt eine ganze Anzahl ausgezeichneter Beispiele für die Behandlung von Dialogen und Steuerelementen im Modul ModuleControls in der Bibliothek Tools zur Verfügung.

18.1.2. Aufruf eines Dialogs aus einem Makro heraus

Dialoge können im Dokument oder in einer globalen Bibliothek gespeichert werden (s. Kapitel 17. Verwaltung der Bibliotheken). Egal wo der Dialog gespeichert ist, er muss vor der ersten Nutzung geladen werden. Sie wissen schon, dass Sie auf die Dialogbibliotheken des aktuellen Dokuments über DialogLibraries und global auf die der Anwendung über GlobalScope.DialogLibraries zugreifen können. Die folgenden Schritte sind nötig, um einen Dialog zu laden, zu erzeugen und auszuführen:

1. Laden Sie die den Dialog enthaltene Bibliothek mit loadLibrary(). Falls die Bibliothek schon geladen ist, können Sie auf diesen Schritt verzichten. Es ist aber sicher klug, sich nicht darauf zu verlassen.
2. Holen Sie die Bibliothek von DialogLibraries mit der Methode getByName().
3. Holen Sie den Dialog aus der Bibliothek mit der Methode getByName().
4. Erzeugen Sie den Dialog mit CreateUnoDialog().
5. Führen Sie den Dialog aus.

Listing 547 lädt einen Dialog, der im aktuellen Dokument gespeichert ist, und führt ihn aus.

Listing 547. *Lädt den Testdialog und führt ihn aus.*

```
Dim oHelloDlg     'Der zur Laufzeit erzeugte Dialog
Sub RunHelloDlg
  Dim oLib        'Bibliothek, die den Dialog enthält
  Dim oLibDlg     'Dialog, wie er in der Bibliothek gespeichert ist

  REM Als erstes wird die Bibliothek geladen.
  REM Wenn der Dialog in einer Bibliothek auf Anwendungsebene gespeichert ist
  REM statt im Dokument, muss GlobalScope.DialogLibraries verwendet werden.
  DialogLibraries.loadLibrary("OOME_40")

  REM Holt die gesamte Bibliothek, nachdem sie geladen ist.
  oLib = DialogLibraries.getByName("OOME_40")

  REM Holt den Dialog, wie er in der Bibliothek gespeichert ist.
  REM Ich betrachte das Objekt als Dialogdefinition.
  oLibDlg = oLib.getByName("HelloDlg")

  REM Erzeugt einen ausführbaren Dialog aus der Dialogdefinition.
  oHelloDlg = CreateUnoDialog(oLibDlg)

  REM Die letzen drei Schritte können Sie auch in einer Zeile ausdrücken:
  'oHelloDlg = CreateUnoDialog(DialogLibraries.OOME_40.HelloDlg)

  REM Nun wird der Dialog ausgeführt.
```

```
  REM Die Methode execute() ist eine Funktion, die den Wert 1 zurückgibt,
  REM wenn der Dialog mit OK geschlossen wurde, und die 0 zurückgibt,
  REM wenn der Dialog mit Abbrechen geschlossen wurde. Ein Klick auf das Kreuzchen
  REM in der rechten oberen Ecke des Dialogs ist gleichbedeutend mit Abbrechen.
  oHelloDlg.execute()
End Sub
```

Tipp Die Variable, die den Dialog referenziert, wird außerhalb der Subroutine deklariert, die den Dialog erzeugt. Somit können Ereignisprozeduren, die als Subroutinen angelegt sind, auf den Dialog zugreifen.

18.1.3. Eine Ereignisprozedur zuweisen

Kontrollelemente und Dialoge können externe Ereignisprozeduren aufrufen. Über den Reiter Ereignisse im Eigenschaften-Dialog können Sie selbst gefertigte Subroutinen als externe Ereignungsbehandlungen zuweisen. Ein kurzer Blick auf den Titel im Bild 144 zeigt, dass es sich um die Ereignisse einer Schaltfläche handelt. Je nach ausgewähltem Objekt können sich die Ereignisse unterscheiden.

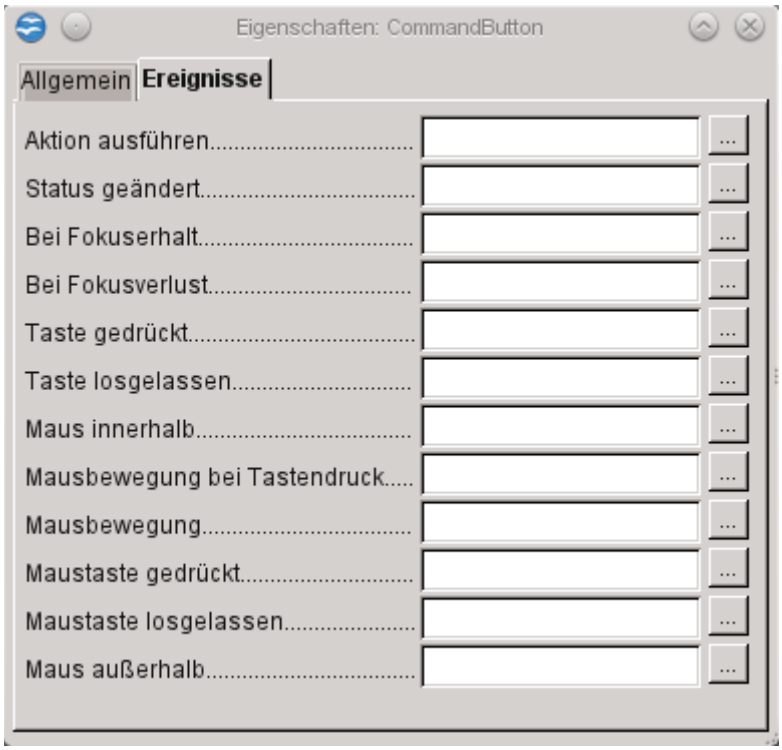


Bild 144. *Ereignisse für eine Schaltfläche.*

Zum Schließen des „Hallo Welt“-Dialogs können Sie eine individuelle Ereignisbehandlung bereitstellen. Stellen Sie im Eigenschaften-Dialog die Art der Schaltfläche von OK auf Standard um, und der Dialog wird sich nicht mehr von selber schließen. Das mag von Ihnen so gewollt sein, wenn Sie zum Beispiel vor dem Schließen des Dialogs die Eingaben überprüfen wollen. Nun schreiben wir die Ereignisprozedur, die den Dialog schließt (s. Listing 548).

Listing 548. *Schließen des Hello-Dialogs.*

```
Sub ExitHelloDlg
  oHelloDlg.endExecute()
End Sub
```

Damit das Ereignis „Aktion ausführen“ das Makro ExitHelloDlg aufruft, öffnen Sie den Eigenschaften-Dialog der Schaltfläche und klicken auf auf den Reiter Ereignisse. Klicken Sie dann auf die Flä-

che mit den drei Pünktchen rechts neben dem gewünschten Ereignis (s. Bild 144) und finden sich im Dialog „Aktion zuweisen“ wieder (s. Bild 145).

Achtung Ich habe je nach OOo-Variante und -Version unterschiedliches Verhalten erlebt. Wenn ich in LO 4.0.2.2 auf die Pünktchen neben „Aktion ausführen“ klicke (s. Bild 144) und dann ein Makro dem Ereignis „Status geändert“ zuweise, wird nichts zugewiesen. In anderen Versionen (nicht LO) konnte ich mehrere und verschiedene Ereignisprozeduren zuweisen, und alles funktionierte.

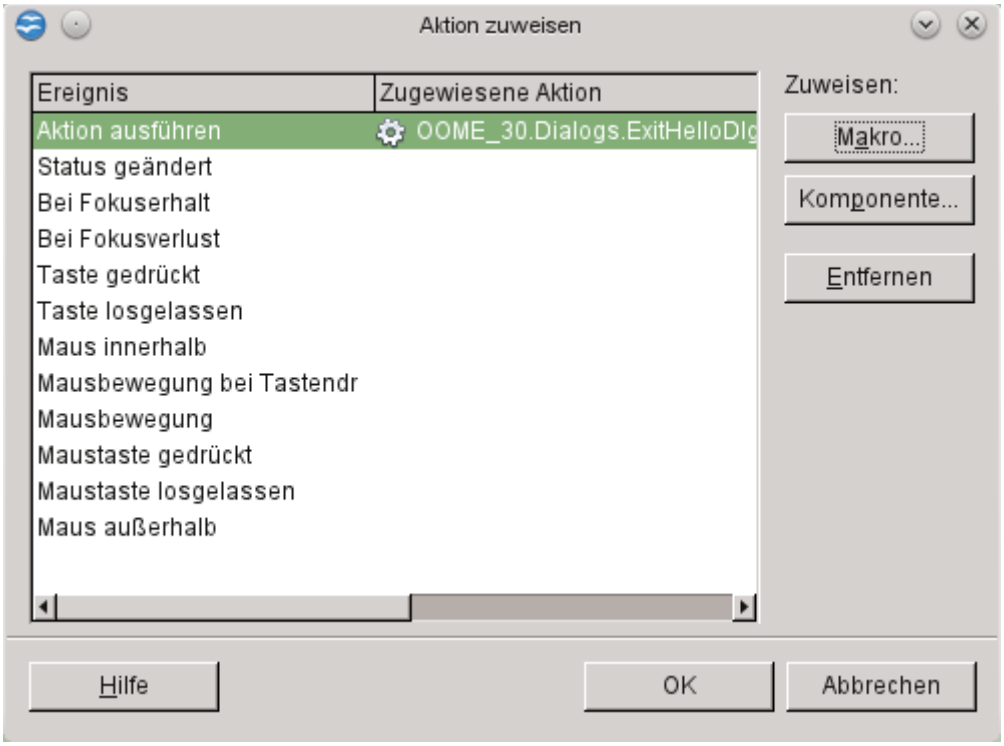


Bild 145. *Makros einem Ereignis zuweisen.*

Selektieren Sie das Ereignis und klicken Sie auf die Fläche „Makro...“. Damit öffnen Sie den Dialog „Makro-Selektor“ (s. Bild 146).

Tipp Wenn der Dialog mit einem XDialogEventHandler verwendet wird, klicken Sie im Dialog „Aktion zuweisen“ (s. Bild 145) auf die Fläche „Komponente...“ und geben Sie den Namen der aufzurufenden Methode ein.

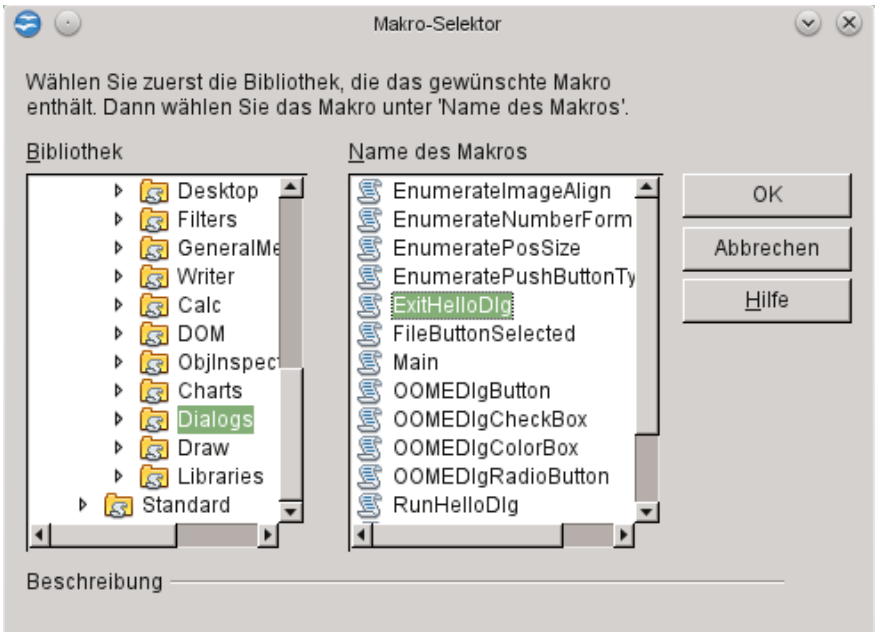


Bild 146. *Auswahl eines Makros für eine Ereignisbehandlung.*

Selektieren Sie in der Spalte „Bibliothek“ den Bibliothekscontainer (Meine Makros, Projektmakros oder ein Dokument), die Bibliothek und das Modul. Schließlich markieren Sie in der Spalte „Name des Makros“ das Makro Ihrer Wahl.

18.2. Dialoge und Steuerungsmuster

Zum Verständnis der Funktionsweise der Steuerelemente ist es wichtig, den Unterschied zwischen Modell (Model), Präsentation (View) und Steuerung (Controller) zu erkennen. Dieses Model-View-Control-Muster, kurz MVC genannt, ist im Bereich der Informationswissenschaft sehr weit verbreitet und akzeptiert.

Das *Modell* beschreibt die Form, die Daten und die Funktionalität des Steuerelements. Das Modell tritt nicht in direkten Kontakt mit dem Benutzer und kann sich auch nicht selbst darstellen. Wenn ein Objekt das MVC-Muster unterstützt, sollten Änderungen immer am Objektmodell vorgenommen werden. Diese Änderungen werden automatisch über die Steuerung in die Präsentationen übertragen. Für ein Objekt kann es nur ein Modell geben.

Die *Präsentation* ist das, was der Benutzer sieht. Obwohl nur ein Modell für ein Objekt existiert, kann es mehrere Präsentationen zu seiner Darstellung geben.

Die *Steuerung* sorgt für die Interaktion mit dem Benutzer. Wenn er der Steuerung einen Handlungsbefehl erteilt, erfolgt eine Änderung am Modell, danach wird die Präsentation aktualisiert.

Tipp	Ein Dokument unterscheidet sich dadurch von einer Desktop-Komponente, die kein Dokument ist (die Basic-IDE oder das Hilfefenster), dass es ein Datenmodell besitzt. OOo nutzt also für Dokumente das MVC-Muster.

Die in Dialogen verwendeten Steuerelemente verfügen im allgemeinen über Methoden im Präsentationsteil, die Änderungen auf das Modell übertragen. Meine Empfehlung lautet aber, direkt auf das Modell zuzugreifen und nicht auf die Präsentationsmethoden, außer Sie haben zwingende Gründe, es anders zu machen. Ich habe gehört, der Weg über die Präsentation kann zu Ungereimtheiten führen. Bei Steuerelementen in Formularen soll die Gefahr ausgeprägter sein als in Dialogen. Ich habe allerdings solche Ungereimtheiten noch nie direkt bemerkt.

18.3. Gemeinsamkeiten von Dialogen und Steuerelementen

In mancher Hinsicht ist ein Dialog auch ein Steuerelement. Dialoge verwenden das MVC-Muster und teilen manche Gemeinsamkeiten mit Steuerelementen. Zum Beispiel unterstützen sie den Service UnoControl, der für alle Steuerelemente gilt.

18.3.1. Interfaces

UNO-Steuerelemente unterstützen, wie auch eine Vielzahl anderer Objekte in OOo, das Interface XComponent (s. Tabelle 271).

Tabelle 271. *Methoden im Interface com.sun.star.lang.XComponent.*

Methode	Beschreibung
dispose()	Der Besitzer des Objekts ruft diese Methode auf, um alle Ressourcen des Objekts zu lösen. Diese Methode wollen Sie normalerweise nicht aufrufen.
addEventListener(XEventListener)	Fügt dem Objekt einen Ereignisbeobachter hinzu.
removeEventListener(XEventListener)	Entfernt einen Ereignisbeobachter aus der Beobachterliste des Objekts.

Das Interface XControl charakterisiert ein Objekt als Steuerelement. Alle Objekte, die den Service UnoControl unterstützen, bieten auch das Interface XControl an. Für Basic-Programmierer ist womöglich die Methode getModel() am interessantesten, denn sie gibt das Datenmodell des Steuerelements zurück (s. Tabelle 272).

Tabelle 272. *Methoden im Interface com.sun.star.awt.XControl.*

Methode	Beschreibung
setContext(XInterface)	Legt den Kontext des Steuerelements fest.
getContext()	Gibt den Kontext des Steuerelements zurück.
createPeer(XToolkit, Parent)	Erzeugt die Fensterfunktionalität XWindowPeer als Kind des als Parent angegebenen XWindowPeer. Wenn Parent Null ist, ist der Desktop die Überordnung. Diese Methode ist dazu da, ein Fenster für einen Dialog zu erzeugen, der in einem Makro anstatt in der IDE erstellt wurde (s. Listing 580).
getPeer()	Gibt den zuvor erzeugten oder bestimmten XWindowPeer zurück.
setModel(XControlModel)	Legt das Modell des Steuerelements fest. Gibt True zurück, wenn erfolgreich.
getModel()	Gibt das XModel des Steuerelements zurück.
getView()	Gibt das Präsentationsobjekt (XView) des Steuerelements zurück.
setDesignMode(Boolean)	Schaltet den Entwurfsmodus an oder ab.
isDesignMode()	True = Steuerelement befindet sich im Entwurfsmodus.
isTransparent()	True = Steuerelement ist transparent.

Ein Fenster ist eine rechteckige Fläche auf einem Ausgabegerät, im Wesentlichen durch Position und Größe definiert. Das Interface XWindow definiert die Grundfunktionen für eine Fensterkomponente.

Tabelle 273. *Methoden im Interface com.sun.star.awt.XWindow.*

Methode	Beschreibung
setPosSize (x, y, breite, höhe, PosSize)	Legt die äußeren Begrenzungen des Fensters fest (s. Tabelle 275).
getPosSize()	Gibt die äußeren Begrenzungen des Fensters als Rechteck zurück.
setVisible(Boolean)	Zeigt oder verbirgt das Fenster.
setEnable(Boolean)	Aktiviert oder deaktiviert das Fenster.
setFocus()	Setzt den Fokus auf das Fenster.

Methode	Beschreibung
addWindowListener(listener)	Fügt einen Fensterbeobachter (XWindowListener) hinzu.
removeWindowListener(listener)	Löscht den XWindowListener aus der Beobachterliste.
addFocusListener(listener)	Fügt einen Fokusbeobachter (XFocusListener) hinzu.
removeFocusListener(listener)	Löscht den XFocusListener aus der Beobachterliste.
addKeyListener(listener)	Fügt einen Tastaturbeobachter (XKeyListener) hinzu.
removeKeyListener(listener)	Löscht den XKeyListener aus der Beobachterliste.
addMouseListener(listener)	Fügt einen Mausbeobachter (XMouseListener) hinzu.
removeMouseListener(listener)	Löscht den XMouseListener aus der Beobachterliste.
addMouseMotionListener(listener)	Fügt einen Mausbewegungsbeobachter (XMouseMotionListener) hinzu.
removeMouseMotionListener(listener)	Löscht den XMouseMotionListener aus der Beobachterliste.
addPaintListener(listener)	Fügt einen Erneuerungsbeobachter (XPaintListener) hinzu.
removePaintListener(listener)	Löscht den XPaintListener aus der Beobachterliste.

Das Interface com.sun.star.awt.XWindow2 bietet weitere Methoden, s. Tabelle 274.

Tabelle 274. *Methoden im Interface com.sun.star.awt.XWindow2*

Methode	Beschreibung
setOutputSize	Setzt die inneren Abmessungen (com.sun.star.awt.Size) des Fensters.
getOutputSize()	Rückgabe: die inneren Abmessungen (com.sun.star.awt.Size) des Fensters .
hasFocus()	Rückgabe: True = das Fenster hat den Fokus.
isActive()	Rückgabe: True = das Fenster ist aktiv.
isEnabled()	Rückgabe: True = das Fenster ist aktiviert.
isVisible()	Rückgabe: True = das Fenster ist sichtbar.

Wenn Sie die Position und die Größe mit einer einzigen Methode einstellen, nämlich mit setPosSize() (s. Tabelle 273), wird über die Werte der Konstantengruppe PosSize gesteuert, welche Teile überhaupt neu gezeichnet werden (s. Tabelle 275). Beispiele für die Methoden zum Auslesen und Festlegen der Position finden Sie in Listing 567 und Listing 568.

Tabelle 275. *Die Konstantengruppe com.sun.star.awt.PosSize.*

Konstante	Name	Beschreibung
1	X	Kennzeichnet die X-Koordinate.
2	Y	Kennzeichnet die Y-Koordinate.
4	WIDTH	Kennzeichnet die Breite.
8	HEIGHT	Kennzeichnet die Höhe.
3	POS	Kennzeichnet die X- und Y-Koordinaten.
12	SIZE	Kennzeichnet die Breite und die Höhe.
15	POSSIZE	Kennzeichnet alles.

Das Interface XView definiert die Methoden zur Darstellung eines Objekts (s. Tabelle 276). Diese Methoden sind vor allem für fortgeschrittene Benutzer von Interesse und sind hier nur der Vollständigkeit halber aufgeführt.

Tabelle 276. *Methoden im Interface com.sun.star.awt.XView.*

Methode	Beschreibung
setGraphics(XGraphics)	Legt das Ausgabegerät fest.

Methode	Beschreibung
getGraphics()	Gibt das Ausgabegerät zurück.
getSize()	Gibt die Größe des Objekts in der Geräteeinheit zurück.
draw(x, y)	Zeichnet das Objekt an der angegebenen Position.
setZoom(x, y)	Legt den Vergrößerungsfaktor für die Inhalte des Steuerelements fest.

18.3.2. Modelle

Alle Steuerelemente, also auch Dialoge, verfügen über ein Modell, das die Form, die Daten und die Funktionalität über einen Satz von Eigenschaften beschreibt. Alle Modelle erben die Eigenschaften des Service com.sun.star.awt.UnoControlDialogElement, s. Tabelle 277.

Tabelle 277. *Eigenschaften im Service com.sun.star.awt.UnoControlDialogElement.*

Eigenschaft	Beschreibung
Height	Die Höhe des Steuerelements.
Name	Der Name des Steuerelements.
PositionX	Die horizontale Position des Steuerelements.
PositionY	Die vertikale Position des Steuerelements.
Step	Die Dialogseite, auf der das Steuerelement erscheint.
TabIndex	Die Position in der Aktivierungsreihenfolge der Steuerelemente.
Tag	Der Titel des Steuerelements.
Width	Die Breite des Steuerelements.

In der ganzen Vielfalt von Elementen und ihrer unterschiedlichen Aufgaben gilt einen kleiner Satz von Eigenschaften für alle oder zumindest die meisten Modelle von Steuerelementen, s. Tabelle 278.

Tabelle 278. *Von vielen Steuerelementmodellen genutzte Eigenschaften.*

Eigenschaft	Beschreibung
HelpText	Der Hilfetext als String.
HelpURL	Der URL-String zum Hilfetext.
Enabled	True = das Steuerelement ist aktiviert.
Printable	True = das Steuerelement wird in der Druckausgabe erscheinen.
BackgroundColor	Die Hintergrundfarbe des Steuerelements. Long Integer.
BorderColor	Rahmenfarbe (RGB). Long Integer.
TextColor	Die Textfarbe des Steuerelements. Long Integer.
TextLineColor	Die Farbe der Textzeile des Steuerelements. Long Integer.
FontDescriptor	Die Schriftartattribute für die Beschriftung des Steuerelements. Struct com.sun.star.awt.FontDescriptor.
FontEmphasisMark	Legt den Typ und die Position des Hervorhebungskennzeichens für ostasiatischen Text in dem Steuerelement fest. Konstantengruppe com.sun.star.text.FontEmphasisMark.
FontRelief	Bestimmt, ob die Beschriftung des Steuerelements erhaben oder vertieft ist. Konstantengruppe com.sun.star.text.FontRelief.
TabStop	True = das Steuerelement kann mit der Tabulatortaste angesteuert werden.

18.4. Dialoge

Die in Tabelle 271 bis Tabelle 276 vorgestellten Methoden gelten für alle Steuerelemente. Es sollte aber nicht überraschen, dass Dialoge ganz eigene Methoden und Eigenschaften besitzen (s. Tabelle 279).

Tabelle 279. *Methoden im Interface com.sun.star.awt.XDialog.*

Methode	Beschreibung
setTitle()	Legt den Dialogtitel fest.
getTitle()	Holt den Dialogtitel.
execute()	Stellt den Dialog dar.
endExecute()	Verbirgt den Dialog und beendet die Methode execute().

Alle Steuerelemente verhalten sich wie Fenster. Sie sind rechtwinklig, können den Fokus erhalten, können aktiviert und deaktiviert werden und besitzen Beobachter (s Tabelle 273). Fenster der obersten Ebene benötigen zusätzliche Funktionalitäten, weil sie nicht zu einem anderen Fenster gehören (s. Tabelle 280).

Tabelle 280. *Methoden im Interface com.sun.star.awt.XTopWindow.*

Methode	Beschreibung
addTopWindowListener(XTopWindowListener)	Fügt diesem Fenster einen Beobachter hinzu.
removeTopWindowListener(XTopWindowListener)	Entfernt einen Beobachter, so dass er von diesem Fenster keine Ereignisse mehr erfährt.
toFront()	Legt dieses Fenster über alle anderen Fenster.
toBack()	Legt dieses Fenster unter alle anderen Fenster.
setMenuBar(XMenuBar)	Gibt diesem Fenster eine Menüleiste.

Die Hauptaufgabe eines Dialogs besteht darin, Steuerelemente aufzunehmen. Die Methoden in der Tabelle 281 ermöglichen das Hinzufügen, Entfernen und Referenzieren von Steuerelementen in einem Dialog. Sehr häufig muss man auf ein Steuerelement eines Dialogs zugreifen, um seinen Wert zu lesen oder zu ändern.

Tabelle 281. *Methoden im Interface com.sun.star.awt.XControlContainer.*

Methode	Beschreibung
setStatusText(String)	Legt den Text in der Statusleiste des Containers fest.
getControls()	Gibt alle Steuerelemente als Array des Typs XControl zurück.
getControl(name)	Gibt das Steuerelement (XControl) dieses Namens zurück.
addControl(name, XControl)	Fügt dem Container ein Steuerelement (XControl) unter diesem Namen hinzu.
removeControl(XControl)	Löscht das Steuerelement (XControl) aus dem Container.

Normalerweise aber werden dem Dialog keine Steuerelemente hinzugefügt oder aus ihm gelöscht. Wenn Sie einen Dialog über ein Makro erstellen, anstatt ihn in der IDE zu entwerfen, fügen Sie dem Modell des Dialogs die Modelle der Steuerelemente hinzu und nicht die Steuerelemente direkt dem Dialog. Es läuft eben so ab, dass die Arbeit im Modell getan wird und die Änderungen dann in der Präsentation nachvollzogen werden. Die am meisten verwendete Methode in der Tabelle 281 ist getControl(name).

Das Modell des Dialogs erhält man mit der Methode getModel() (s. Tabelle 272). Dialoge unterstützen den Service UnoControlDialogModel (s. Tabelle 282), der viele der Eigenschaften definiert, die man in der Basic-IDE einstellen kann (s. Bild 143). Einige dieser Eigenschaften kann man vom Dialogobjekt aus einstellen – zum Beispiel mit der Methode setTitle() in der Tabelle 279 –, besser ist es aber, direkt das Modell zu modifizieren.

Tabelle 282. *Eigenschaften im Service com.sun.star.awt.UnoControlDialogModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer: BorderColor, Printable, TabStop. Weitere Eigenschaften:	
Closeable	True = der Dialog ist schließbar.
Moveable	True = der Dialog ist verschiebbar.
Sizeable	True = die Größe des Dialogs ist veränderbar.
Title	String, der als Text in der Titelleiste des Dialogs steht.
DesktopAsParent	True = der Dialog ist ein Desktop-Dialog.
ImageURL	URL, der eine Hintergrundgrafik referenziert. String.
Graphic	Grafikobjekt. Interaktion zwischen Graphic und ImageURL: • Wenn ImageURL gesetzt ist, wird Graphic zu dem Objekt, das von dem URL geladen wird, oder NULL, wenn der URL auf keine gültige Grafik zeigt. • Wenn Graphic gesetzt ist, wird aus ImageURL ein leerer String.

Man kann Dialoge und Steuerelemente in einem Makro erzeugen, anstatt sie in der Basic-IDE zu entwickeln. Das Modell des Dialogs bietet die vom Interface com.sun.star.lang.XMultiServiceFactory definierte Methode createInstance(). Wenn ein Makro ein in einen Dialog einzufügendes Steuerelement erzeugt, sollte dieses vom Modell des Dialogs erzeugt werden. Ein Steuerelement, das vom globalen Servicemanager erzeugt wurde, besitzt nicht die in der Tabelle 277 gezeigten Eigenschaften. Später in diesem Kapitel werde ich darauf eingehen, wie man Steuerelemente in Makros erzeugt. Das Makro im Listing 549 listet die Objekte auf, die das Modell eines Dialogs erzeugen kann (s. Bild 147). LO (5.3. und 5.4) gibt für die vier Services com.sun.star.awt.UnoControlProgressBarModel, UnoControlScrollBarModel, UnoControlFixedLineModel und UnoControlRoadmapModel jeweils nur einen leeren String aus. Die Services können aber erzeugt werden.

Listing 549. *Die Services, die das Modell eines Dialogs erzeugen kann.*

```
Sub ShowDialogServices
  Dim oLib        'Bibliothek, die den Dialog enthält
  Dim oLibDlg     'Dialog, wie er in der Bibliothek gespeichert ist

  DialogLibraries.loadLibrary("OOME_40")
  oLib = DialogLibraries.getByName("OOME_40")
  oLibDlg = oLib.getByName("HelloDlg")
  oHelloDlg = CreateUnoDialog(oLibDlg)
  MsgBox Join(oHelloDlg.getModel().getAvailableServiceNames(), Chr$(10))
End Sub
```

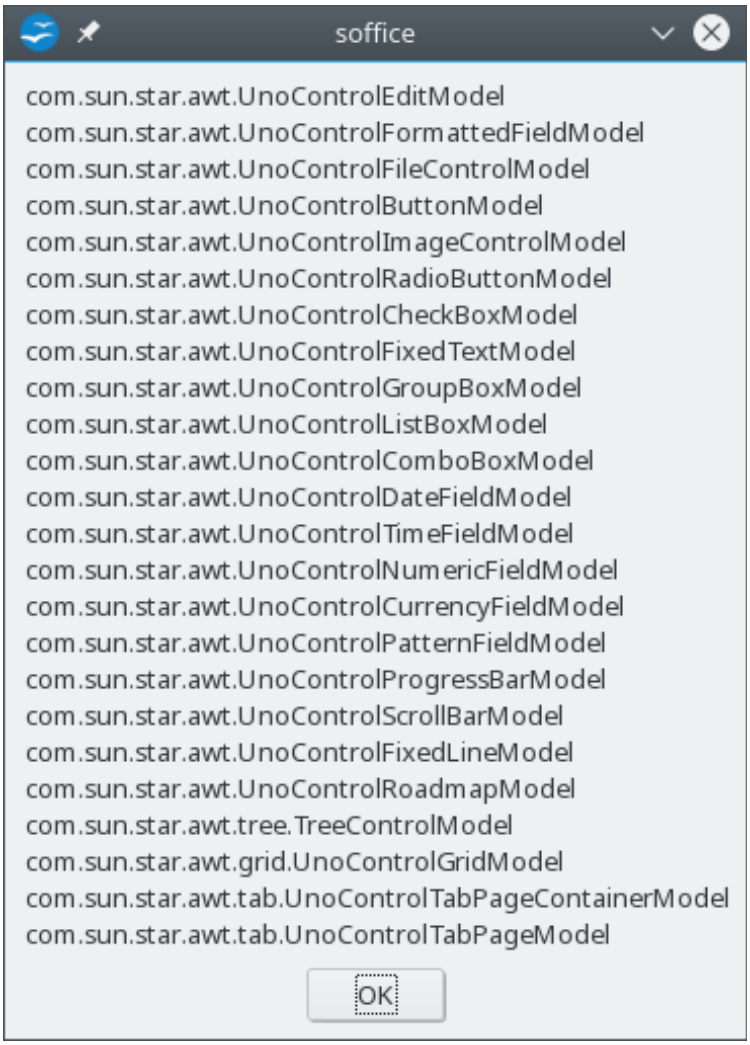


Bild 147. *Services, die das Modell eines Dialogs in AOO 4.1.4 erzeugen kann.*

18.5. Steuerelemente

Man kann ein Steuerelement mit der Methode getControl() direkt aus einem Dialog adressieren. Mit der Methode getModel() des Steuerelements kann man dann auf sein Modell zugreifen. Man kann auf das Modell aber auch mit getByName() vom Modell des Objekts her zugreifen, in dem es enthalten ist, also vom Modell des Dialogs. Der Code im Listing 550 demonstriert die verfügbaren Optionen.

Tipp	Steuerelemente können in Dialoge und Formulare eingefügt werden, aber nicht alle Steuerelemente funktionieren in Dialogen. Manche Steuerelemente benötigen eine Datenquelle, doch Dialoge können keine Datenquelle enthalten. In diesem Abschnitt geht es nur um Steuerelemente, die in Dialogen verwendbar sind.

Listing 550. *Zugriff auf die Schaltfläche oder auf das Modell der Schaltfläche.*

```
oHelloDlg.getControl("OKButton")                'Die Schaltfläche (UnoControlButton)
oHelloDlg.getControl("OKButton").getModel()     'Das Modell der Schaltfläche
                                                '(UnoControlButtonModel)
oHelloDlg.getModel().getByName("OKButton")      'Das Modell der Schaltfläche
                                                '(UnoControlButtonModel)
```

Die in einem Dialog nutzbaren Steuerelemente sind im Modul com.sun.star.awt definiert (s. Tabelle 283 und Bild 147). Die Servicenamen der Steuerelemente sind in den Servicenamen ihrer Modelle enthalten.

Tabelle 283. *Im Modul com.sun.star.awt definierte Steuerelemente und ihre Modelle.*

Steuerelement	Modell	Beschreibung
TreeControl	TreeControlModel	Baumansicht-Steuerelement.
UnoControlButton	UnoControlButtonModel	Schaltfläche.
UnoControlCheckBox	UnoControlCheckBoxModel	Markierfeld.
UnoControlComboBox	UnoControlComboBoxModel	Kombinationsfeld.
UnoControlCurrencyField	UnoControlCurrencyFieldModel	Währungsfeld.
UnoControlDateField	UnoControlDateFieldModel	Datumsfeld.
UnoControlDialog	UnoControlDialogModel	Dialog.
UnoControlEdit	UnoControlEditModel	Textfeld.
UnoControlFileControl	UnoControlFileControlModel	Dateiauswahl.
UnoControlFixedLine	UnoControlFixedLineModel	Linie.
UnoControlFixedText	UnoControlFixedTextModel	Beschriftungsfeld.
UnoControlFormattedField	UnoControlFormattedFieldModel	Formatiertes Feld.
UnoControlGroupBox	UnoControlGroupBoxModel	Gruppierungsrahmen.
UnoControlImageControl	UnoControlImageControlModel	Grafisches Feld.
UnoControlListBox	UnoControlListBoxModel	Listenfeld.
UnoControlNumericField	UnoControlNumericFieldModel	Numerisches Feld.
UnoControlPatternField	UnoControlPatternFieldModel	Maskiertes Feld.
UnoControlProgressBar	UnoControlProgressBarModel	Fortschrittsbalken.
UnoControlRadioButton	UnoControlRadioButtonModel	Optionsfeld.
UnoControlRoadmap	UnoControlRoadmapModel	Roadmapfeld.
UnoControlScrollBar	UnoControlScrollBarModel	Bildlaufleiste.
UnoControlTimeField	UnoControlTimeFieldModel	Zeitfeld.

Zur Demonstration der Funktionsweise verschiedener Steuerelemente in einem Dialog habe ich einen Beispieldialog zusammengestellt. Dieser Dialog ist unter dem Namen OOMESample in diesem Dokument enthalten. Die für die Steuerung nötigen Makros finden Sie im Modul Dialogs.

Ich möchte Sie dazu einladen, als Übung diesen Dialog in einem neuen Textdokument Stück für Stück aufzubauen, das heißt, dass Sie mit fast jedem Steuerelement, das Sie auf den folgenden Seiten kennenlernen, dem Dialog einen weiteren Baustein hinzufügen.

Erstellen Sie also in einem neuen Textdokument eine Bibliothek „OOME_40“ und darin einen Dialog mit dem Namen OOMESample. Im Laufe dieses Kapitels werde ich Ihnen in „Übung“ genannten grünen Boxen sagen, welche Steuerelemente Sie dem Dialog hinzufügen.

Bild 148 zeigt Ihnen den vorgefertigten Dialog, an dem Sie sich orientieren können. Machen Sie den Dialog so groß, dass die Steuerelemente Platz finden. Sollten Sie bemerken, dass die Fläche zu groß oder zu klein ist, können Sie sie ja über die grünen Griffe anpassen.

Bild 148. *Der Dialog OOMESample, der im Laufe dieses Kapitels aufgebaut wird.*

Übung

Erstellen Sie in einem neuen Textdokument eine Bibliothek „OOME_40“ und darin einen Dialog mit dem Namen OOMESample.

Bild 148 zeigt Ihnen den Dialog, wie er am Ende aussehen könnte. Machen Sie den Dialog so groß, dass die Steuerelemente Platz finden. Sollten Sie bemerken, dass die Fläche zu groß oder zu klein ist, können Sie sie ja über die grünen Griffe anpassen.

18.5.1. Schaltfläche

Eine Schaltfläche (Control button) ist eine Art Knopf, auf den man drücken muss, damit etwas passiert. Der Typ der Schaltfläche wird von der Eigenschaft PushButtonType im Modell der Schaltfläche gesteuert. Sie legt fest, ob eine eigens geschriebene Ereignisbehandlung erwartet wird oder ob der Dialog ohne weiteres geschlossen oder abgebrochen wird. Für diese vordefinierten Aktionen sind Standard-Ereignisbehandlungen automatisch verfügbar (s. Tabelle 284).

Tabelle 284. *Die Enumeration com.sun.star.awt.PushButtonType.*

Wert	Beschreibung
STANDARD	Der Standardwert. Sie müssen Ihre eigene Ereignisbehandlung schreiben.
OK	Schließt den Dialog ordnungsgemäß und kehrt mit dem Wert 1 aus der Funktion execute() zurück.
CANCEL	Schließt den Dialog durch Abbruch und kehrt mit dem Wert 0 aus der Funktion execute() zurück.
HELP	Hilfefläche. Die Hilfe steckt im angegebenen URL.

Die Fläche einer Schaltfläche kann entweder eine Beschriftung zeigen – Eigenschaft Label – oder eine Grafik – Eigenschaft ImageURL. Wird keine Beschriftung, sondern eine Grafik dargestellt, wird das Bild abhängig von einem Wert der Konstantengruppe ImageAlign ausgerichtet (s. Tabelle 285).

Tabelle 285. *Die Konstantengruppe com.sun.star.awt.ImageAlign.*

Konstante	Name	Beschreibung
0	LEFT	Am linken Rand ausgerichtet.
1	TOP	Am oberen Rand ausgerichtet.
2	RIGHT	Am rechten Rand ausgerichtet.
3	BOTTOM	Am unteren Rand ausgerichtet.

Tipp Eine Schaltfläche kann zwar ein Bild darstellen, sie beschneidet es aber auf die passende Größe. Wenn Sie das Bild skalieren wollen, wählen Sie statt der Schaltfläche ein grafisches Feld (ImageControl).

Mittlerweile ist mit ImagePosition eine Eigenschaft eingeführt, die erheblich mehr Möglichkeiten bietet als ImageAlign, s. Tabelle 286.

Tabelle 286. *Die Konstantengruppe com.sun.star.awt.ImagePosition.*

Konstante	Name	Beschreibung
0	LeftTop	Am linken und oberen Rand des Textes positioniert.
1	LeftCenter	In der Mitte des linken Randes des Textes positioniert.
2	LeftBottom	Am linken und unteren Rand des Textes positioniert.
3	RightTop	Am rechten und oberen Rand des Textes positioniert.
4	RightCenter	In der Mitte des rechten Randes des Textes positioniert.
5	RightBottom	Am rechten und unteren Rand des Textes positioniert.
6	AboveLeft	Am linken Rand und oberhalb des Textes positioniert.
7	AboveCenter	In der Mitte und oberhalb des Textes positioniert.
8	AboveRight	Am rechten Rand und oberhalb des Textes positioniert.
9	BelowLeft	Am linken Rand und unterhalb des Textes positioniert.
10	BelowCenter	In der Mitte und unterhalb des Textes positioniert.
11	BelowRight	Am rechten Rand und unterhalb des Textes positioniert.
12	Centered	Am Zentrum des Textes positioniert.

Das Modell der Schaltfläche hat viele Eigenschaften mit dem Modell des Dialogs gemein, bietet aber auch einige Eigenschaften, die nur für Schaltflächen gelten (s. Tabelle 287).

Tabelle 287. *Eigenschaften im Service com.sun.star.awt.UnoControlButtonModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer: BorderColor. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in der Schaltfläche (Optional): 0: links, 1: zentriert, 2: rechts.
DefaultButton	True = dies ist die Standardschaltfläche.
FocusOnClick	Regelt, ob bei einem Klick auf die Schaltfläche der Fokus wechselt. True = die Schaltfläche erhält automatisch den Fokus (Standard) False = der Fokus bleibt auf dem Steuerelement, wo er vorher war.

Eigenschaft	Beschreibung
ImageAlign	Die Ausrichtung eines Bildes in der Schaltfläche (s. Tabelle 285).
ImagePosition	Die Positionierung des Bildes relativ zum Text als Wert der Konstantengruppe com.sun.star.awt.ImagePosition (s. Tabelle 286). Falls diese Eigenschaft gesetzt ist, verdrängt sie die Eigenschaft ImageAlign.
ImageURL	Der URL des Bildes, das in der Schaltfläche angezeigt wird. Ist das Bild zu groß, wird es abgeschnitten.
Graphic	Grafikobjekt. Interaktion zwischen Graphic und ImageURL: • Wenn ImageURL gesetzt ist, wird Graphic zu dem Objekt, das von dem URL geladen wird, oder NULL, wenn der URL auf keine gültige Grafik zeigt. • Wenn Graphic gesetzt ist, wird aus ImageURL ein leerer String.
Label	String der Beschriftung der Schaltfläche.
MultiLine	True = die Beschriftung kann mehr als eine Zeile umfassen. Optional.
PushButtonType	Die Standard-Ereignisbehandlung der Schaltfläche (s. Tabelle 284).
Repeat	Klickwiederholung beim Halten der Maustaste nach dem Klicken. False = keine Wiederholung (Standard) True = rhythmische Wiederholung
RepeatDelay	Der zeitliche Abstand zwischen Klickwiederholungen, in Millisekunden.
State	1 = die Schaltfläche ist aktiviert, 0 = die Schaltfläche ist deaktiviert.
Toggle	Legt fest, ob der Status der Schaltfläche in einer einzigen Aktion umgeschaltet wird. Optional. True = ein Klick auf die Schaltfläche oder das Drücken der Leertaste, wenn der Fokus auf der Schaltfläche liegt, schaltet den Status zwischen „gedrückt“ und „nicht gedrückt“ um. Standard = False, so dass sich die Fläche wie eine normale Schaltfläche verhält.
VerticalAlign	Die vertikale Textausrichtung in der Schaltfläche. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.

Übung

Fügen Sie dem Dialog OOMESample eine OK-Schaltfläche hinzu. Geben Sie ihr die „Titel“-Eigenschaft „Schließen“, und als „Art der Schaltfläche“ wählen Sie „OK“ aus.

Kopieren Sie den Code im Listing 551 in das Modul in der Makrobibliothek OOME_40.

Fügen Sie dem Dialog noch eine zweite Schaltfläche hinzu mit dem Namen „ClickButton“ und dem Titel „Hier klicken“ und verknüpfen Sie deren Ereignis „Aktion ausführen“ mit der Subroutine OOMEDlgButton.

Fügen Sie dem Dialog OOMESample zwei Schaltflächen hinzu. Richten Sie eine davon so ein, dass ein Klick darauf den Dialog schließt. Dazu geben Sie der Schaltfläche den Titel „Schließen“ und wählen Sie „OK“ als Art der Schaltfläche. Die Schließen-Schaltfläche wird den Dialog automatisch beenden, wenn sie ausgelöst wird. Nennen Sie die zweite Schaltfläche ClickButton und geben Sie ihr den Titel „Hier klicken“. Das Ereignis „Aktion ausführen“ der Schaltfläche ClickButton wird mit der Subroutine OOMEDlgButton (s. Listing 551) verknüpft. Wenn man nun auf die Schaltfläche klickt, ändert sich ihre Beschriftung. Sie zeigt nun die Anzahl der bisherigen Klicks.

Listing 551. *Startet den Beispieldialog OOMEDlg.*

```
Option Explicit

REM Globale Variablen
Dim oOOMEDlg     'Der zur Laufzeit erzeugte Dialog
Dim nClickCount% 'Wie oft die Schaltfläche geklickt wurde

Sub RunOOMEDlg
  REM Deklaration der Hauptvariablen
  Dim oLib        'Bibliothek, die den Dialog enthält
```

```
  Dim oLibDlg     'Dialog, wie er in der Bibliothek gespeichert ist

  REM Laden der Bibliothek und des Dialogs.
  DialogLibraries.loadLibrary("OOME_40")
  oLib = DialogLibraries.getByName("OOME_40")
  oLibDlg = oLib.getByName("OOMESample")
  oOOMEDlg = CreateUnoDialog(oLibDlg)

  REM Start der Konfiguration.
  nClickCount = 0
  oOOMEDlg.getModel.Title = "OOME-Beispiel-Dialog"

  REM Aufruf des Dialogs.
  oOOMEDlg.execute()
End Sub

Sub OOMEDlgButton
  Dim oButtonModel
  nClickCount = nClickCount + 1
  oButtonModel = oOOMEDlg.getModel().getByName("ClickButton")
  oButtonModel.Label = nClickCount & " mal geklickt"
End Sub
```

18.5.2. Markierfeld

Normalerweise benötigt man ein Markierfeld (Check box) zur Anzeige zweier Zustände, ja oder nein. In OOo ist es hingegen auch für drei Zustände nutzbar. Standardmäßig sind nur zwei Zustände ermöglicht, 0 (nicht markiert) und 1 (markiert). Die Eigenschaft State des Modells enthält den jeweiligen Zustand. Wenn die Eigenschaft TriState aber True ist, gibt es auch einen dritten Status: 2 (unbestimmt). In der Darstellung des Markierfelds ist dann der Haken gesetzt, das Feld aber ist ausgegraut (AOO 3.4.1 zeigt einen waagerechten Strich). Die Tabelle 288 führt die Eigenschaften des Modells des Markierfelds auf.

Tabelle 288. *Eigenschaften im Service com.sun.star.awt.UnoControlCheckBoxModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer: BorderColor. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in dem Markierfeld (Optional): 0: links, 1: zentriert, 2: rechts.
ImagePosition	Die Positionierung des Bildes relativ zum Text als Wert der Konstantengruppe com.sun.star.awt.ImagePosition (s. Tabelle 286).
ImageURL	Der URL des Bildes, das in dem Markierfeld angezeigt wird. Ist das Bild zu groß, wird es abgeschnitten.
Graphic	Grafikobjekt. Interaktion zwischen Graphic und ImageURL: • Wenn ImageURL gesetzt ist, wird Graphic zu dem Objekt, das von dem URL geladen wird, oder NULL, wenn der URL auf keine gültige Grafik zeigt. • Wenn Graphic gesetzt ist, wird aus ImageURL ein leerer String.
Label	String der Beschriftung des Markierfelds.
MultiLine	True = die Beschriftung kann mehr als eine Zeile umfassen. Optional.
State	0 = das Markierfeld ist nicht markiert, 1 = das Markierfeld ist markiert, 2 = der Status des Markierfelds ist unbestimmt.
TriState	True = das Markierfeld unterstützt den Status 2 „unbestimmt“.

Eigenschaft	Beschreibung
VerticalAlign	Die vertikale Textausrichtung in dem Markierfeld. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.
VisualEffect	Visueller Effekt des Markierfelds. Konstantengruppe com.sun.star.awt.VisualEffect: NONE = 0 (kein Effekt), LOOK3D = 1 (dreidimensional), FLAT = 2 (zweidimensional).
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.

Übung

Fügen Sie dem Dialog OOMESample ein Markierfeld hinzu und geben Sie ihm den Namen „CheckBox“.

Kopieren Sie die Zeilen aus dem Listing 552 hinter den bereits vorhandenen Code zur Konfiguration innerhalb der Subroutine RunOOMEDlg.

Kopieren Sie die Subroutine OOMEDlgCheckBox im Listing 553 in das Makromodul und verknüpfen Sie damit das Ereignis „Aktion ausführen“.

Fügen Sie dem Beispieldialog ein Markierfeld hinzu und ändern Sie seine Eigenschaft „Name“ auf „CheckBox“. In der Subroutine RunOOMEDlg fügen Sie folgende Zeilen an den Konfigurationsblock an. Damit wird das Modell des Markierfelds für drei Zustände sensibilisiert. Diese Eigenschaft kann auch direkt in der Basic-IDE eingestellt werden, als Eigenschaft „Dreifacher Status“.

Listing 552. *Das Markierfeld erhält drei Statusmöglichkeiten.*

```
REM Einstellung für das Markierfeld.
oOOMEDlg.getModel().getByName("CheckBox").TriState = True
```

Fügen Sie dem Modul die Subroutine OOMEDlgCheckBox aus dem Listing 553 hinzu und verknüpfen Sie diese mit dem Ereignis „Aktion ausführen“ des Markierfelds. Durch Aktivierung des Feldes wird der Status umgeschaltet und die Beschriftung zur Anzeige des aktuellen Status geändert.

Listing 553. *Änderung der Beschriftung des Markierfelds zur Anzeige des aktuellen Status.*

```
Sub OOMEDlgCheckBox
  Dim oModel
  oModel = oOOMEDlg.getModel().getByName("CheckBox")
  oModel.Label = "Status " & oModel.State
End Sub
```

18.5.3. Optionsfeld

Optionsfelder (Radio button oder Option button) braucht man zur Auswahl eines Postens aus einer kurzen Liste – für eine lange Liste nimmt man eher ein Kombinationsfeld oder ein Listenfeld. Optionsfelder werden immer als Gruppe verwendet, in der jeweils nur ein Optionsfeld aktiv sein kann. Wenn ein Optionsfeld angewählt wird, kennzeichnet OOo alle anderen Optionsfelder derselben Gruppe automatisch als nicht markiert.

Die Eigenschaft TabIndex (s. Tabelle 277) legt die Reihenfolge fest, in der die Kontrollelemente den Fokus erhalten, wenn man wiederholt die Tabulatortaste drückt. Der TabIndex wird normalerweise beim Design des Dialogs in der Basic-IDE festgelegt, in der dortigen Eigenschaft „Aktivierungsreihenfolge“.

In OOo bewirkt ein Gruppierungsrahmen (UnoControlGroupBox) nichts anderes, als einen Rahmen um eine Gruppe von Steuerelementen zu ziehen. Es ist nur ein optischer Hinweis, dass zum Beispiel Optionsfelder als Gruppe verstanden werden, aber eine echte Gruppierung ist es nicht. Optionsfelder bilden eine echte Gruppe, wenn und nur wenn sie in direkter Aktivierungsreihenfolge stehen. Im Gegensatz dazu werden Optionsfelder in Visual Basic durch den Gruppierungsrahmen gruppiert. Erzeugen Sie also alle zu einer Gruppe gehörenden Optionsfelder direkt hintereinander. So erhalten sie automatisch aufeinander folgende Aktivierungszählungen und bilden dadurch eine Gruppe. Wenn Sie

nun anschließend den Gruppierungsrahmen um die Optionsfelder erzeugen, wird dieser Rahmen die Aktivierungsreihenfolge unterbrechen, und Sie können nun problemlos eine neue Gruppe von Optionsfeldern beginnen (s. Tabelle 289).

Tabelle 289. *Eigenschaften im Service com.sun.star.awt.UnoControlRadioButtonModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer: BorderColor. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in dem Optionsfeld (Optional): 0: links, 1: zentriert, 2: rechts.
ImagePosition	Die Positionierung des Bildes relativ zum Text als Wert der Konstantengruppe com.sun.star.awt.ImagePosition (s. Tabelle 286).
ImageURL	Der URL des Bildes, das in dem Optionsfeld angezeigt wird. Ist das Bild zu groß, wird es abgeschnitten.
Graphic	Grafikobjekt. Interaktion zwischen Graphic und ImageURL: • Wenn ImageURL gesetzt ist, wird Graphic zu dem Objekt, das von dem URL geladen wird, oder NULL, wenn der URL auf keine gültige Grafik zeigt. • Wenn Graphic gesetzt ist, wird aus ImageURL ein leerer String.
Label	String der Beschriftung des Optionsfelds.
MultiLine	True = die Beschriftung kann mehr als eine Zeile umfassen. Optional.
State	0 = das Optionsfeld ist nicht ausgewählt, 1 = das Optionsfeld ist ausgewählt.
VerticalAlign	Die vertikale Textausrichtung in dem Optionsfeld. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.
VisualEffect	Visueller Effekt des Optionsfelds. Konstantengruppe com.sun.star.awt.VisualEffect: NONE = 0 (kein Effekt), LOOK3D = 1 (dreidimensional), FLAT = 2 (zweidimensional).
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.

Übung

Fügen Sie dem Dialog OOMESample drei Optionsfelder hinzu mit den Namen „OBAus“, „OBEin“ und „OBUnbestimmt“ und geben Sie ihnen auch genau diese Titel.

Kopieren Sie die Subroutine OOMEDlgRadioButton im Listing 554 in das Makromodul und verknüpfen Sie damit bei jedem Optionsfeld das Ereignis „Aktion ausführen“.

Ändern Sie den Code der Subroutine OOMEDlgCheckBox in den des Listing 555.

Fügen Sie dem Beispieldialog drei Optionsfelder hinzu und geben Sie ihnen die Namen „OBAus“, „OBEin“ und „OBUnbestimmt“. Diese drei Optionsfelder werden den Status des Markierfelds beeinflussen. Geben Sie ihnen also ihre Namen auch als Titel (s. Bild 148). Das Optionsfeld OBEin wird also das Markierfeld anhaken, OBAus wird den Haken entfernen, und OBUnbestimmt wird es in den „Unbestimmt“-Status versetzen. Fügen Sie dem Modul die Subroutine OOMEDlgRadioButton aus dem Listing 554 hinzu und verknüpfen Sie diese mit dem Ereignis „Aktion ausführen“ eines jeden der drei Optionsfelder. Das Makro untersucht der Reihe nach jedes dieser Optionsfelder. Abhängig davon, welches ausgewählt wurde, werden der Status des Markierfelds und seine Beschriftung geändert.

Listing 554. *Aktionen der Optionsfelder.*

```
REM Wird ein Optionsfeld ausgewählt, wird der Status des Markierfelds
REM abhängig vom ausgewählten Optionsfeld geändert. Gleichzeitig zeigt die
REM Beschriftung des Markierfelds das Optionsfeld an, das den Status geändert hat.
```

```
Sub OOMEDlgRadioButton
  Dim oModel
  Dim i As Integer
  Dim sBNames (0 To 2) As String
  REM Die Reihenfolge der Namen entspricht der Reihenfolge
  REM der Statusänderungen des Markierfelds.
  sBNames(0) = "OBAus" : sBNames(1) = "OBEin" : sBNames(2) = "OBUnbestimmt"
  For i = 0 To 2
    oModel = oOOMEDlg.getModel().getByName(sbNames(i))
    If oModel.State = 1 Then
      oOOMEDlg.getModel().getByName("CheckBox").State = i
      oOOMEDlg.getModel().getByName("CheckBox").Label = "Gesetzt von " & sBNames(i)
      REM oOOMEDlg.getModel().getByName("OBFrame").Label = sBNames(i)
      Exit For
    End If
  Next
End Sub
```

Der Code für das Markierfeld wird wie folgt geändert:

Listing 555. *Änderung des Markierfelds.*

```
Sub OOMEDlgCheckBox
  Dim oModel
  Dim sBNames (0 To 2) As String
  sBNames(0) = "OBAus" : sBNames(1) = "OBEin" : sBNames(2) = "OBUnbestimmt"

  oModel = oOOMEDlg.getModel().getByName("CheckBox")
  oModel.Label = "Status " & oModel.State

  If oModel.State = 0 Then
    oOOMEDlg.getModel().getByName("OBAus").State = 1
  ElseIf oModel.State = 1 Then
    oOOMEDlg.getModel().getByName("OBEin").State = 1
  ElseIf oModel.State = 2 Then
    oOOMEDlg.getModel().getByName("OBUnbestimmt").State = 1
  End If
  oOOMEDlg.getModel().getByName("OBFrame").Label = sBNames(oModel.State)
End Sub
```

18.5.4. Gruppierungsrahmen

Der Gruppierungsrahmen (Group box) bietet durch den Rahmen, der um eine Gruppe von Steuerelementen gezogen wird, einen visuellen Hinweis darauf, dass die Steuerelemente irgendwie miteinander zu tun haben. Das Modell des Gruppierungsrahmens bietet folgende Eigenschaften, s. Tabelle 290.

Tabelle 290. *Eigenschaften im Service com.sun.star.awt.UnoControlGroupBoxModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer: BackgroundColor, BorderColor, Enabled. Weitere Eigenschaften:	
Label	String der Beschriftung des Gruppierungsrahmens.
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.

Übung Ziehen Sie einen Gruppierungsrahmen mit dem Namen OBFrame und dem Titel „Optionen“ um die drei Optionsfelder im Dialog OOMESample.

Entfernen Sie in der Subroutine OOMEDlgRadioButton die Kommentarkennung „REM “ am Anfang der Zeile vor der Zeile „Exit For“.

Ziehen Sie einen Gruppierungsrahmen um die drei Optionsfelder und nennen Sie ihn RBFrame. Das Makro im Listing 554 enthält eine Codezeile, die die Beschriftung des Rahmens in den Namen des aktuell ausgewählten Optionsfeldes ändert. Entfernen Sie die Kommentarkennung „REM “ am Anfang der Zeile, damit die Beschriftung aktualisiert werden kann.

18.5.5. Horizontale oder vertikale Linie

Die horizontale und die vertikale Linie (Fixed line control) sind wie der Gruppierungsrahmen für den Benutzer nur als visuelle Abgrenzung vorgesehen. Ihr einziger Sinn besteht in der optischen Trennung bestimmter Teile des Dialogs. In der Basic-IDE gibt es zwei getrennte Elemente für die horizontale und die vertikale Ausrichtung (s. Tabelle 314). Das Modell einer Linie hat die Eigenschaft Orientation, deren Wert 0 die horizontale und 1 die vertikale Ausrichtung bestimmt.

18.5.6. Kombinationsfeld

Ein Kombinationsfeld (Combo box) ist ein Textfeld mit beigefügtem Listenfeld, auch als Aufklappliste bekannt. Mit dem Kombinationsfeld wird ein einzelner Wert zurückgegeben, mit der Möglichkeit, aus der Liste vordefinierter Werte auszuwählen. Der ausgewählte oder eingegebene Text ist über die Eigenschaft Text des Modells des Kombinationsfelds verfügbar (s. Tabelle 291).

Tabelle 291. *Eigenschaften im Service com.sun.star.awt.UnoControlComboBoxModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in demKombinationsfeld (Optional): 0: links, 1: zentriert, 2: rechts.
Autocomplete	True = die automatische Wortergänzung ist aktiviert.
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
Dropdown	True = das Kombinationsfeld verfügt über einen Aufklappbutton.
HideInactiveSelection	True = die Listenauswahl ist verborgen, wenn das Kombinationsfeld nicht den Fokus hat.
LineCount	Maximale Anzahl der in der Aufklappliste angezeigten Zeilen.
MaxTextLen	Maximale Anzahl an Zeichen. 0 = die Textlänge ist nicht begrenzt.
ReadOnly	True = der Text kann nicht vom Benutzer geändert werden.
StringItemList	Die Einträge der Liste als Stringarray.
Text	Text, der im Eingabefeld des Kombinationsfelds angezeigt wird.
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.
MouseWheelBehavior	Scrollen mit dem Mausrad durch die Listeneinträge. Konstantangruppe com.sun.star.awt.MouseWheelBehavior: SCROLL_DISABLED = 0 (kein Scrollen) SCROLL_FOCUS_ONLY = 1 (Scrollen nur, wenn das Feld den Fokus hat) SCROLL_ALWAYS = 2 (Scrollen immer möglich).

Eigenschaft	Beschreibung
TypedItemList	Liste von Objekten. Diese Liste korrespondiert mit der StringItemList und muss dieselbe Länge haben. Wenn über die ComboBox ein neuer Wert eingegeben wurde, wird diese Liste ungültig. Seit LibreOffice 5.4

Obwohl man gut beraten ist, die Kommunikation mit einem Steuerelement über sein Modell zu pflegen, so bietet das Kombinationsfeld selbst ein paar nützliche Methoden für Einstellungen, die nicht direkt über das Modell verfügbar sind (s. Tabelle 292). Unter anderem handelt es sich um Methoden zur Verwaltung der Listeneinträge. Sie brauchen somit keine eigenen Funktionen zum Hinzufügen oder Entfernen von Einträgen in der StringItemList (s. Tabelle 291) zu schreiben.

Tabelle 292. *Weitere Methoden im Interface com.sun.star.awt.UnoControlComboBox.*

Methode	Beschreibung
addItem(String, position)	Fügt einen Eintrag an der angegebenen Position ein.
addItems(StringArray, position)	Fügt mehrere Einträge an der angegebenen Position ein.
removeItems(position, count)	Löscht eine Anzahl von Einträgen an der angegebenen Position.

Übung

Fügen Sie dem Dialog OOMESample ein Kombinationsfeld mit dem Namen „ColorBox“ hinzu.

Tragen Sie in der Eigenschaft „Listen-Einträge“ die drei Werte „Rot“, „Grün“ und „Blau“ ein.

Kopieren Sie die Subroutine OOMEDlgColorBox im Listing 556 in das Makromodul und verknüpfen Sie damit das Ereignis „Text modifiziert“.

Fügen Sie dem Dialog ein Kombinationsfeld mit dem Namen „ColorBox“ hinzu. Tragen Sie in der Eigenschaft „Listen-Einträge“ die drei Werte „Rot“, „Grün“ und „Blau“ ein. In früheren OOo-Versionen mussten Sie vor der Eingabe eines weiteren Werts die Tastenkombination Strg-Enter drücken. In AOO 3.4.1 und LO ist es die Kombination Shift-Enter. Sie schreiben also das Wort Rot und drücken dann Shift-Enter, dann schreiben Sie Grün und wieder Shift-Enter, dann Blau und – Enter. Das könnte vielleicht intuitiver sein. Ich habe jedenfalls eine Zeitlang gebraucht, um das herauszufinden. Es ist auf jeden Fall einfach, die Eigenschaft StringItemList des Modells (s. Tabelle 291) mit einem Stringarray dieser Werte zu setzen.

Fügen Sie dem Modul die Subroutine OOMEDlgColorBox im Listing 556 hinzu und verknüpfen Sie sie mit dem Ereignis „Text modifiziert“ des Kombinationsfeldes. Dieses Ereignis ruft die Subroutine jedes Mal auf, wenn sich der Text im Textfeld ändert. Wenn Sie also dort etwas schreiben, wird die Subroutine bei jedem Tastendruck aufgerufen. Wenn Sie stattdessen das Ereignis „Status geändert“ nehmen, wird die Subroutine immer dann aufgerufen, wenn aus der Liste ein neuer Eintrag gewählt wurde.

Das Makro im Listing 556 ändert die Hintergrundfarbe der Aufklappliste gemäß der ausgewählten (oder eingegebenen) Farbe. Der Textvergleich unterscheidet zwischen Groß- und Kleinschreibung. Wenn Sie es rot haben wollen, geben Sie also „Rot“ ein.

Listing 556. *Aktionen des Kombinationsfeldes ColorBox.*

```
Sub OOMEDlgColorBox
  Dim oModel
  Dim s As String
  oModel = oOOMEDlg.getModel().getByName("ColorBox")
  s = oModel.Text
  If s = "Rot" Then
    REM Umstellung auf Rot.
    oModel.BackGroundColor = RGB(255, 0, 0)
  ElseIf s = "Grün" Then
    REM Umstellung auf Grün.
```

```
    oModel.BackGroundColor = RGB(0, 255, 0)
  ElseIf s = "Blau" Then
    REM Umstellung auf Blau.
    oModel.BackGroundColor = RGB(0, 0, 255)
  Else
    REM Umstellung zurück auf Weiß.
    oModel.BackGroundColor = RGB(255, 255, 255)
  End If
End Sub
```

18.5.7. Texteingabefelder

Der Service UnoControlEdit dient als Basisservice für alle anderen spezielleren Eingabefelder. Ein Eingabefeld enthält regulären Text. Man kann die Textlänge begrenzen und sogar Bildlaufleisten anbringen. Das Modell eines Eingabefelds unterstützt die in Tabelle 293 aufgeführten Eigenschaften.

Tabelle 293. *Eigenschaften im Service com.sun.star.awt.UnoControlEditModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in dem Eingabefeld: 0: links, 1: zentriert, 2: rechts.
AutoHScroll	True = Bei Bedarf wird automatisch eine horizontale Bildlaufleiste eingefügt.
AutoVScroll	True = Bei Bedarf wird automatisch eine vertikale Bildlaufleiste eingefügt.
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
EchoChar	ASCII-Wert (Integer) des Zeichens, das in einem Eingabefeld, das ein Passwort erwartet, für jedes Zeichen des eingegebenen Texts angezeigt wird.
HardLineBreaks	True = manuelle Zeilenumbrüche werden von der Methode getText() mit zurückgegeben.
HideInactiveSelection	True = die Auswahl ist verborgen, wenn das Eingabefeld nicht den Fokus hat.
HScroll	True = der Text in dem Eingabefeld kann horizontal gescrollt werden.
LineEndFormat	Zeilenendekennung in mehrzeiligen Texten. Manuelle Zeilenumbrüche bei der Texteingabe werden gemäß der Einstellung kodiert, so dass die Text-Eigenschaft nur die eingestellten Zeilenumbrüche enthält. Diese Einstellung ist irrelevant bei neuem Text, der über die API eingegeben wird. Alle Zeilenendekennungen sollten unabhängig von dieser Einstellung korrekt erkannt und dargestellt werden. Konstantengruppe com.sun.star.awt.LineEndFormat: CARRIAGE_RETURN = 0 (Mac: Chr(13)) LINE_FEED = 1 (Linux: Chr(10)) CARRIAGE_RETURN_LINE_FEED = 2 (Windows: Chr(13) & Chr(10))
MaxTextLen	Maximale Anzahl an Zeichen. 0 = die Textlänge ist nicht begrenzt.
MultiLine	True = der Text kann über mehr als eine Zeile gehen.
PaintTransparent	True = der Hintergrund ist transparent.
ReadOnly	True = der Text kann nicht vom Benutzer geändert werden.
Text	Text, der im Textfeld des Eingabefelds angezeigt wird.
VScroll	True = der Text in dem Eingabefeld kann vertikal gescrollt werden.
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.
VerticalAlign	Die vertikale Textausrichtung in dem Eingabefeld. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.

Obwohl ich für die meisten Sachen empfehle, das Modell zu verwenden, so sind doch manche Funktionen nur über das Steuerelement selbst verfügbar. Beispielsweise bietet das Eingabefeld die Möglichkeit, einen Teil oder den gesamten Text zu selektieren und dann zu ermitteln, welcher Teil des Textes gerade selektiert ist. In einem normalen Textdokument erledigt der Controller diese Aufgabe, aber in Dialogen wird die Funktionalität des Controllers mit dem Steuerelement selbst kombiniert. Um einen selektierten Bereich festzulegen oder zu ermitteln, benötigt man das Struct Selection (s. Tabelle 294).

Tabelle 294. *Eigenschaften des Structs com.sun.star.awt.Selection.*

Eigenschaft	Beschreibung
Min	Untere Grenze des Bereichs. Long Integer.
Max	Obere Grenze des Bereichs. Long Integer.

Das Eingabefeld ermittelt oder bestimmt die aktuelle Textselektion mit Hilfe des Structs Selection. Es kann auch den selektierten Text direkt zurückgeben. Der Benutzer erhält dadurch die Möglichkeit, nur einen Teil des Eingabetextes zu markieren und eben diesen Teil zu ermitteln. In der Tabelle 295 finden Sie die von Texteingabefeldern unterstützten Standardmethoden.

Tabelle 295. *Methoden im Interface com.sun.star.awt.XTextComponent.*

Methode	Beschreibung
addTextListener(XTextListener)	Fügt einen Textbeobachter hinzu (für das Ereignis „Text modifiziert“).
getMaxTextLen()	Gibt die Eigenschaft MaxTextLen des Modells zurück (s. Tabelle 293).
getSelectedText()	Gibt den aktuell selektierten Textstring zurück.
getSelection()	Gibt ein Selection-Objekt zurück, das den selektierten Textbereich beschreibt (s. Tabelle 294).
getText()	Gibt die Eigenschaft Text des Modells zurück (s. Tabelle 293).
insertText(Selection, string)	Fügt den String an der angegebenen Selection-Stelle ein.
isEditable()	Gibt die Eigenschaft ReadOnly des Modells zurück (s. Tabelle 293).
removeTextListener(XTextListener)	Entfernt einen Textbeobachter.
setEditable(boolean)	Legt die Eigenschaft ReadOnly des Modells fest (s. Tabelle 293).
setMaxTextLen(len)	Legt die Eigenschaft MaxTextLen des Modells fest (s. Tabelle 293).
setSelection(Selection)	Legt den selektierten Textbereich fest (s. Tabelle 294).
setText(string)	Legt die Eigenschaft Text des Modells fest (s. Tabelle 293).

Währungsfeld

Wie sein Name schon sagt, dient das Währungsfeld (Currency control) der Eingabe von Währungswerten. Es ist durch den Service UnoControlCurrencyField definiert, der auch den Standard-Textfeld-Service einbindet. In der Tabelle 296 finden Sie einige speziell zum Modell des Währungsfelds gehörende Eigenschaften, die alle auch über den Eigenschaften-Dialog der IDE gesetzt werden können. Achten Sie besonders auf die Standardeinstellungen und nehmen Sie zur Kenntnis, dass in diesem Dialog Ja und Nein für die Werte True und False stehen. Neben den speziellen Eigenschaften der Tabelle 296 werden auch noch folgende Eigenschaften unterstützt: BackgroundColor, Border, Enabled, FontDescriptor, FontEmphasisMark, FontRelief, HelpText, HelpURL, Printable, ReadOnly, Tabstop, TextColor und TextLineColor.

Tabelle 296. *Eigenschaften im Service com.sun.star.awt.UnoControlCurrencyFieldModel.*

Eigenschaft	Deutsche Bezeichnung (IDE)	Standard
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier.		
Weitere Eigenschaften:		

Eigenschaft	Deutsche Bezeichnung (IDE)	Standard
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.	
CurrencySymbol	Währungssymbol. String.	€
DecimalAccuracy	Nachkommastellen. Integer.	2
HideInactiveSelection	True = die Auswahl ist verborgen, wenn das Währungsfeld nicht den Fokus hat.	
PrependCurrencySymbol	Symbol voranstellen. True = das Symbol wird vorangestellt, False = das Symbol wird nachgestellt.	False
ReadOnly	True = das Währungsfeld kann nicht geändert, sondern nur gelesen werden.	False
Repeat	True = die Maus wiederholt die Aktion, solange sie gedrückt ist.	True
RepeatDelay	Pause zwischen den Wiederholungen der Mausaktion in Millisekunden. Long Integer.	
ShowThousandsSeparator	Tausender-Trennzeichen. True = mit Tausender-Trennzeichen; False = ohne Tausender-Trennzeichen.	False
Spin	Drehfeld. True = mit zusätzlichem Drehfeld.	False
StrictFormat	Formatüberprüfung. True = das Format wird während der Eingabe überprüft.	False
Value	Der Zahlenwert. Fließkommazahl Double.	0
ValueMax	Maximaler Wert. Fließkommazahl Double.	1000000,00
ValueMin	Minimaler Wert. Fließkommazahl Double.	-1000000,00
ValueStep	Intervall. Der Betrag, um den das Drehfeld den vorhandenen Wert ändert. Double.	1.0
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.	
MouseWheelBehavior	Scrollen mit dem Mausrad über die numerischen Werte. Konstantangruppe com.sun.star.awt.MouseWheelBehavior: SCROLL_DISABLED = 0 (kein Scrollen) SCROLL_FOCUS_ONLY = 1 (Scrollen nur, wenn das Feld den Fokus hat) SCROLL_ALWAYS = 2 (Scrollen immer möglich).	
VerticalAlign	Die vertikale Textausrichtung. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.	

Tipp Die Standardwerte der Tabelle 296 gelten für Deutschland. Es werden die Einstellungen des aktuellen Gebietsschemas verwendet.

Das vom Währungsfeld eingebundene Interface XCurrencyField definiert get- und set-Methoden zum Zugriff auf die in der Tabelle 296 gezeigten Eigenschaften. Das Währungsfeld unterstützt auch das Interface XSpinField, das Methoden zur Interaktion mit dem Drehfeld bereithält (s. Tabelle 297).

Tabelle 297. *Methoden im Interface com.sun.star.awt.XSpinField.*

Methode	Beschreibung
addSpinListener(XSpinListener)	Fügt dem Steuerelement einen Drehfeld-Beobachter hinzu.
down()	Vermindert den aktuellen Wert um den Wert von ValueStep (s. Tabelle 296).

Methode	Beschreibung
enableRepeat(boolean)	Schaltet den Wiederholungsmodus aus oder ein. True = die Aktion wird automatisch dauernd wiederholt, solange die Maus gedrückt ist.
first()	Setzt den Wert auf den Wert von ValueMin (s. Tabelle 296).
last()	Setzt den Wert auf den Wert von ValueMax (s. Tabelle 296).
removeSpinListener(XSpinListener)	Entfernt einen Drehfeld-Beobachter.
up()	Erhöht den aktuellen Wert um den Wert von ValueStep (s. Tabelle 296).

Tipp Alle Texteingabefelder, die Drehfelder ermöglichen, unterstützen die Methoden der Tabelle 297.

Numerisches Feld

Das numerische Feld (Numeric control) ist für die Eingabe von Zahlen da. Es ist nahezu identisch mit dem Währungsfeld mit der Ausnahme, dass es die beiden Eigenschaften CurrencySymbol und PrependCurrencySymbol nicht gibt – sogar die Standardwerte sind dieselben. Wenn Sie wissen, wie ein Währungsfeld funktioniert, können Sie auch mit einem numerischen Feld umgehen.

Datumsfeld

Das Datumsfeld (Date control) ist ein Texteingabefeld zur Eingabe eines Datums. Die möglichen Eingabeformate sind vordefiniert und über eine numerische Konstante verfügbar (s. Tabelle 298). Die in der Spalte Beschreibung angegebenen Formate zeigen zwar Schrägstriche wie in TT/MM/JJ, tatsächlich aber ist das Trennzeichen abhängig vom Gebietsschema. Ein Datum wird also in der Form angezeigt und bearbeitet, wie es dort üblich ist, wo Sie leben. Die letzten beiden Formate sind allerdings nicht vom Gebietsschema abhängig, denn es sind die von der ISO-Norm 8601 beschriebenen Formate.

Das Datumsfeld ähnelt dem Währungsfeld darin, dass es einen Minimalwert, einen Maximalwert und ein Drehfeld unterstützt. Das Drehfeld zählt den Teil des Datums, in dem sich die Schreibmarke befindet, hoch oder runter. Wenn der Cursor zum Beispiel im Monatsfeld ist, wirkt das Drehfeld auf den Monat und nicht auf das Jahr oder den Tag. Die Eigenschaften des Modells des Datumsfeld finden Sie in der Tabelle 299.

Tipp Die Eigenschaft DropDown bietet einen Aufklappkalender, der Datumseinträge erheblich erleichtert..

Tabelle 298. *Vom Datumsfeld unterstützte Datumseingabeformate.*

Wert	Beschreibung (deutsch)
0	Standard (kurz)
1	Standard (kurz JJ)
2	Standard (kurz JJJJ)
3	Standard (lang)
4	TT/MM/JJ
5	MM/TT/JJ
6	JJ/MM/TT
7	TT/MM/JJJJ
8	MM/TT/JJJJ
9	JJJJ/MM/TT
10	JJ-MM-TT (ISO 8601 kurz)
11	JJJJ-MM-TT (ISO 8601 lang)

Tabelle 299. *Eigenschaften im Service com.sun.star.awt.UnoControlDateFieldModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier. Weitere Eigenschaften:	
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
Date	Das Datum: – LO alt und AOO: als Zahl Long Integer – LO neu: als Struct com.sun.star.util.Date , s. Tabelle 300.
DateFormat	Das Datumsformat. Integer (s. Tabelle 298).
DateMax	Das höchste erlaubte Datum. – LO alt und AOO: als Zahl Long Integer – LO neu: als Struct com.sun.star.util.Date , s. Tabelle 300.
DateMin	Das niedrigste erlaubte Datum. – LO alt und AOO: als Zahl Long Integer – LO neu: als Struct com.sun.star.util.Date , s. Tabelle 300.
DateShowCentury	True = das Jahrhundert wird angezeigt.
DropDown	True = ein Aufklappkalender wird eingefügt.
HideInactiveSelection	True = die Auswahl ist verborgen, wenn das Datumsfeld nicht den Fokus hat.
ReadOnly	True = das Datumsfeld kann nicht geändert, sondern nur gelesen werden.
Repeat	True = die Maus wiederholt die Aktion, solange sie gedrückt ist.
RepeatDelay	Pause zwischen den Wiederholungen der Mausaktion in Millisekunden. Long Integer.
Spin	True = ein Drehfeld wird eingefügt.
StrictFormat	True = das Format wird während der Eingabe überprüft.
Text	Text, der im Textfeld des Datumsfelds angezeigt wird.
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.
MouseWheelBehavior	Scrollen mit dem Mausrad über die numerischen Werte. Konstantangruppe com.sun.star.awt.MouseWheelBehavior: SCROLL_DISABLED = 0 (kein Scrollen) SCROLL_FOCUS_ONLY = 1 (Scrollen nur, wenn das Feld den Fokus hat) SCROLL_ALWAYS = 2 (Scrollen immer möglich).
VerticalAlign	Die vertikale Textausrichtung. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.

Übung Fügen Sie dem Dialog OOMESample ein Datumsfeld mit dem Namen MyDateField hinzu.

Kopieren Sie die Zeilen aus dem Listing 559 hinter den bereits vorhandenen Code zur Konfiguration innerhalb der Subroutine RunOOMEDlg.

Der von Basic verwendete Datentyp Date ist nicht kompatibel mit der Eigenschaft Date in einem Datumsfeld. Welches Format das Datumsfeld benötigt, hängt von der benutzten OpenOffice-Anwendung ab.

In AOO und in den LO-Versionen vor 4.1 verwendet das Datumsfeld einen Long-Integer-Wert der Form JJJJMMTT. Die Ziffernfolge entspricht also dem ISO-Format. Ein Beispiel: der 3. Februar 2004 wird als die Zahl 20040203 (in Worten: zwanzig Millionen 40 Tausend zweihundertdrei) repräsentiert. Die Basic-Funktion CDateToIso() konvertiert eine Variable vom Typ Date in das benötigte Format. Fügen Sie ein Datumsfeld hinzu und nennen es MyDateField. Die Anweisung im Listing 557 setzt den Initialwert des Datumsfelds auf das aktuelle Datum.

Listing 557. *Setzt das Datumsfeld auf das aktuelle Datum (LO alt und AOO).*

```
REM Setzt den Initialwert auf Heute.
oOOMEDlg.getModel().getByName("MyDateField").Date = CDateToIso(Date())
```

In LO hat sich mittlerweile das Format des Datumsfelds mit der Version 4.1 geändert. Statt des Long Integers wird das Struct com.sun.star.util.Date verwendet (s. Tabelle 300).

Tabelle 300. *Eigenschaften des Structs com.sun.star.util.Date.*

Eigenschaft	Beschreibung
Day	Tag des Monats. Integer (1-31 oder 0 für ein leeres Datum).
Month	Monat des Jahres. Integer (1-12 oder 0 für ein leeres Datum).
Year	Jahr. Integer.

In LO neu müssen Sie zuerst ein com.sun.star.util.Date-Objekt definieren, mit dem Sie dann das Datumsfeld initialisieren. Die Anweisung im Listing 558 setzt den Initialwert des Datumsfelds auf das aktuelle Datum.

Listing 558. *Setzt das Datumsfeld auf das aktuelle Datum (LO neu).*

```
Dim x As Date
Dim x2 As New com.sun.star.util.Date
....
x = Date() 'Das Struct soll das aktuelle Datum enthalten:
x2.Year  = Year(x)      ' Jahr
x2.Month = Month(x)     ' Monat
x2.Day   = Day(x)       ' Tag
oOOMEDlg.getModel().getByName("MyDateField").Date = x2
```

Wenn Sie also ein Datumsfeld in einem Makro benötigen, das sowohl in AOO als auch in allen LO-Versionen laufen soll, müssen Sie den Typ des Date-Objekts kennen. Der UNO-Service CoreReflection bietet das Instrumentarium für eine solche Abfrage (siehe 10.5.1 Der Service CoreReflection (V. Lenhardt) und Listing 222).

Der kürzeste Weg jedoch ist die Steuerung über eine Fehlerbehandlung (s. Listing 559). Falls Sie nicht mehr wissen, wie das geht, informieren Sie sich im Abschnitt 3.10. Fehlerbehandlung mit On Error.

Listing 559. *Setzt das Datumsfeld auf das aktuelle Datum (alle AOO- und LO-Versionen).*

```
REM Initialisierung des Datumsfelds.
REM Der Initialwert soll das aktuelle Datum sein.
Dim tToday As Date
tToday = Date()

On Error Goto LO
REM Versuch, das Datumsfeld als Long Integer zu initialisieren.
REM Dies ist das korrekte Format für AOO sowie für LO vor Version 4.1.
oOOMEDlg.getModel().getByName("MyDateField").Date = CDateToIso(tToday)
On Error Goto 0  'Die Fehlerbehandlung wird aufgehoben.
Goto GoOn

LO:
REM Bei Error wird das Datumsfeld als Struct initialisiert.
REM Dies ist das korrekte Format für neuere Versionen von LO.
On Error Goto 0  'Die Fehlerbehandlung wird aufgehoben.
Dim oToday As New com.sun.star.util.Date
oToday.Year  = Year(tToday)
oToday.Month = Month(tToday)
oToday.Day   = Day(tToday)
```

```
    oOOMEDlg.getModel().getByName("MyDateField").Date = oToday

    GoOn:
...
```

Das in Listing 559 gezeigte Verfahren ist immer dann zu empfehlen, wenn Sie einem Datumsfeld einen Wert zuweisen, denn häufig können Sie nicht sicher sein, ob das Feld momentan einen Wert enthält oder nicht. Wenn es allerdings schon einen Wert enthält, ist eine elegantere Lösung zur Hand: die Basic-Funktion IsUnoStruct(). Die können Sie auch immer verwenden, wenn Sie den Wert eines Datumsfelds auslesen, s. Listing 560.

Listing 560. *Lesen eines Datumsfelds (alle AOO- und LO-Versionen).*

```
Function GetDateFromCtrl(oCtrl, tDate As Date) As Boolean
  REM Auslesen eines Datumsfelds in die Date-Variable tDate.
  REM Gibt False zurück, wenn das Datumsfeld leer ist.
  REM Zur Erinnerung: Ein Date-Wert 0 ist der 30. Dezember 1899!

  Dim CtrlDate  'Typ Variant

  GetDateFromCtrl = False
  If Not oCtrl.isEmpty() Then
    CtrlDate = oCtrl.getModel().Date
    If IsUnoStruct(CtrlDate) Then
      'LO neu: Der Wert ist ein Struct
      tDate = DateSerial(CtrlDate.Year, _
                         CtrlDate.Month, _
                         CtrlDate.Day)
    Else
      'LO alt oder AOO: Der Wert ist vom Typ Long
      tDate = DateSerial(CtrlDate / 10000, _
                        (CtrlDate Mod 10000) / 100, _
                        (CtrlDate Mod 10000) Mod 100)
    End If
    GetDateFromCtrl = True
  End If
End Function
```

Es ist nicht zu empfehlen, die Unterscheidung vom Namen der aktuellen Anwendung abhängen zu lassen, den Sie über die Funktion GetProductName() im Modul Misc der Anwendungsbibliothek Tools auslesen können:

```
GlobalScope.BasicLibraries.loadLibrary("Tools")
Print GetProductName()
```

Auch wenn Sie sich wahrscheinlich darauf verlassen können, dass die Formatänderung seit LO 4.1 besteht, so können Sie sich nicht sicher sein, dass es AOO auf Dauer beim Status Quo bewenden lässt. Sie müssten immer sorgsam darauf achten, ob und wann die AOO-API korrigiert wird, und dann Ihre Makros ändern. Das ist mehr als lästig.

Das Datumsfeld hält eine Reihe von Methoden bereit, zum Beispiel die Eigenschaften des Modells zu holen und zu setzen. Interessanter jedoch sind die Methoden, die das Eingabeverhalten betreffen. Wenn Sie in einem Datumsfeld die Taste Bild Auf drücken, springt die Anzeige auf das „letzte“ Datum. Standardmäßig ist es dasselbe Datum wie in der Eigenschaft DateMax. Doch das letzte Datum wird vom Datumsfeld direkt gesetzt und nicht vom Modell, und es darf von DateMax abweichen. Analog dazu bewirkt das Drücken der Taste Bild Ab, dass der Wert auf das „erste“ Datum springt. Das erste und das letzte Datum werden über die Methoden setFirst() und setLast() eingestellt (s. Ta-

belle 301). Mit der Methode isEmpty() ermitteln Sie, ob das Datumsfeld einen leeren Wert enthält, und mit der Methode setEmpty() setzen Sie den Wert des Datumsfelds auf leer.

Tabelle 301. *Methoden im Interface com.sun.star.awt.UnoControlDateField.*

Methode	Beschreibung
setDate(long) [LO alt und AOO] setDate(com.sun.star.util.Date) [LO neu]	Setzt die Eigenschaft Date des Modells.
getDate()	Holt die Eigenschaft Date des Modells.
setMin(long)	Setzt die Eigenschaft DateMin des Modells.
getMin(long)	Holt die Eigenschaft DateMin des Modells.
setMax(long)	Setzt die Eigenschaft DateMax des Modells.
getMax(long)	Holt die Eigenschaft DateMax des Modells.
setFirst(long)	Setzt das Datum für die Taste Bild Ab.
getFirst()	Holt das Datum für die Taste Bild Ab.
setLast(long)	Setzt das Datum für die Taste Bild Auf.
getLast()	Holt das Datum für die Taste Bild Auf.
setLongFormat(boolean)	Setzt oder beendet die Verwendung des Langformats für das Datum.
isLongFormat()	Gibt True zurück, wenn das Langformat für das Datum verwendet wird.
setEmpty()	Setzt den Datumswert auf leer.
isEmpty()	Gibt True zurück, wenn der aktuelle Wert leer ist.
setStrictFormat(boolean)	Setzt die Eigenschaft StrictFormat des Modells.
isStrictFormat()	Holt die Eigenschaft StrictFormat des Modells.

Uhrzeitfeld

Das Feld für die Uhrzeit (Time control) ist dem Datumsfeld sehr ähnlich. Uhrzeitwerte können einerseits die Zeit auf einer Uhr anzeigen, andererseits aber auch die Dauer einer Zeit. Tabelle 302 zeigt die unterstützten Formate.

Tabelle 302. *Vom Uhrzeitfeld unterstützte Eingabeformate.*

Wert	Beschreibung
0	Kurzformat für 24 Stunden.
1	Langformat für 24 Stunden.
2	Kurzformat für 12 Stunden.
3	Langformat für 12 Stunden.
4	Kurzformat für eine Zeitdauer.
5	Langformat für eine Zeitdauer.

Die Eigenschaften des Modells des Uhrzeitfelds sind in der Tabelle 303 zu sehen. Wenn Sie in der Tabelle 299 das Wort Date durch Time ersetzen und die Eigenschaft DropDown entfernen, erhalten Sie die Tabelle 303. Datums- und Uhrzeitfeld sind also weitgehend identisch.

Tabelle 303. *Eigenschaften im Service com.sun.star.awt.UnoControlTimeFieldModel.*

Eigenschaft	Beschreibung
Time	Die dargestellte Uhrzeit: – LO alt und AOO: als Zahl Long Integer – LO neu: als Struct com.sun.star.util.Time, s. Tabelle 304.
TimeFormat	Das Uhrzeitformat. Integer (s. Tabelle 302).

Eigenschaft	Beschreibung
TimeMax	Die höchste erlaubte Uhrzeit. – LO alt und AOO: als Zahl Long Integer – LO neu: als Struct com.sun.star.util.Time, s. Tabelle 304.
TimeMin	Die niedrigste erlaubte Uhrzeit. – LO alt und AOO: als Zahl Long Integer – LO neu: als Struct com.sun.star.util.Time, s. Tabelle 304.

Wie zu erwarten ist, gleichen die Methoden des Datumsfelds denen des Uhrzeitfelds (s. Tabelle 301). Mit zwei Unterschieden: statt getDate() und setDate() heißt es im Uhrzeitfeld getTime() und setTime(), und das Uhrzeitfeld bietet keine Methoden setLongFormat() und isLongFormat().

In älteren Versionen von LO und in AOO (auch in der aktuellen Version 4.1.1) verwendet das Uhrzeitfeld einen Long-Integer-Wert der Form HHMMSSNN. Die beiden Stellen ganz rechts repräsentieren die Sekundenbruchteile, die beiden nach links anschließenden enthalten die Sekunden, die nächsten beiden die Minuten und die beiden ganz links die Stunden. Führende Nullen werden natürlich nicht zurückgegeben, denn es handelt sich ja um den Typ Long Integer. Glücklicherweise kann man die Werte leicht extrahieren. Unter der Annahme, dass der Dialog ein Uhrzeitfeld mit dem Namen MyTimeField enthält, ermittelt der Code im Listing 561 den Wert der Uhrzeit und berechnet daraus die Stunden, Minuten, Sekunden und Sekundenbruchteile.

Listing 561. *Uhrzeitwerte aus einem Uhrzeitfeld (LO alt und AOO).*

```
n = oOOMEDlg.getModel().getByName("MyTimeField").Time
h = n / 1000000 Mod 100  REM Die Stunden.
m = n / 10000 Mod 100    REM Die Minuten.
s = n / 100 Mod 100      REM Die Sekunden.
ss = n Mod 100           REM Die Sekundenbruchteile.
```

In den neueren Versionen von LO wird das Struct com.sun.star.util.Time für das Uhrzeitfeld verwendet, s. Tabelle 304.

Tabelle 304. *Eigenschaften des Structs com.sun.star.util.Time.*

Eigenschaft	Beschreibung
	Sekundenbruchteile:
HundredthSeconds [LO alt und AOO]	100stel Sekunden. Integer (0 – 99)
NanoSeconds [LO neu]	Nanosekunden. Long Integer (0 – 999 999 999)
Seconds	Sekunden. Integer (0 – 59).
Minutes	Minuten. Integer (0 – 59).
Hours	Stunden. Integer (0 – 23).
IsUTC [nur LO neu]	True: Zeitzone ist UTC, False: unbekannte Zeitzone.

Das Auslesen eines Zeitwerts aus einem Uhrzeitfeld benötigt also in den neueren Versionen von LibreOffice eine Variable des Typs com.sun.star.util.Time, s. Listing 562.

Listing 562. *Uhrzeitwerte aus einem Uhrzeitfeld (LO neu).*

```
Dim oTime As New com.sun.star.util.Time
oTime = oOOMEDlg.getModel().getByName("MyTimeField").Time
h = oTime.Hours          REM Die Stunden.
m = oTime.Minutes        REM Die Minuten.
s = oTime.Seconds        REM Die Sekunden.
ss = oTime.NanoSeconds   REM Die Sekundenbruchteile.
```

Wenn Sie also ein Uhrzeitfeld in einem Makro benötigen, das sowohl mit AOO als auch mit allen LO-Versionen laufen soll, können Sie den Programmfluss ähnlich steuern, wie es oben für ein Datumsfeld beschrieben ist: entweder durch eine Fehlerbehandlung wie im Listing 559 oder mithilfe der Basic-Funktion IsUnoStruct() wie im Listing 560.

Formatiertes Feld

Das formatierte Feld (Formatted control) ist ein allgemeines Eingabefeld für Zahlenwerte. Zahlen können in einem benutzerdefinierten Format, als Prozent, Währung, Datum oder Uhrzeit, in wissenschaftlicher Form, als Bruch oder als Wahrheitswert eingegeben werden. Leider hat dieser Komfort seinen Preis. Das formatierte Feld funktioniert nur mit einem von OOo bereitgestellten Zahlenformat. Am einfachsten ist es, einem formatierten Feld ein numerisches Formatobjekt über den Eigenschaften-Dialog in der IDE zuzuweisen. Über ein Makro ist folgende Abfolge die Regel:

- Entscheiden Sie sich für ein Zahlenformat.
- Durchsuchen Sie das FormatsSupplier-Objekt (s. Tabelle 305) mit den Methoden aus der Tabelle 307. Verwenden Sie die Konstanten aus der Tabelle 306 bei der Suche nach enthaltenen passenden Formaten.
- Weisen Sie der Eigenschaft FormatKey (s. Tabelle 305) den im FormatsSupplier gefundenen Schlüsselwert für das gewünschte Format zu.

Tabelle 305. *Eigenschaften im Service com.sun.star.awt.UnoControlFormattedFieldModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in dem formatierten Feld: 0: links, 1: zentriert, 2: rechts.
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
EffectiveDefault	Der Standardwert des formatierten Felds als Zahl oder String, je nach Wert der Eigenschaft TreatAsNumber.
EffectiveMax	Der Maximalwert als Double, wenn TreatAsNumber den Wert True hat.
EffectiveMin	Der Minimalwert als Double, wenn TreatAsNumber den Wert True hat.
EffectiveValue	Der aktuelle Wert als Zahl oder String, je nach dem Wert von TreatAsNumber.
FormatKey	Der Schlüsselwert für das Format. Long Integer.
FormatsSupplier	XNumberFormatsSupplier, der die von diesem Feld unterstützten Formate verwaltet.
HideInactiveSelection	True = die Auswahl ist verborgen, wenn das formatierte Feld nicht den Fokus hat.
MaxTextLen	Maximale Anzahl an Zeichen. 0 = die Textlänge ist nicht begrenzt.
ReadOnly	True = der Text kann nicht vom Benutzer geändert werden.
Repeat	True = die Maus wiederholt die Aktion, solange sie gedrückt ist.
RepeatDelay	Pause zwischen den Wiederholungen der Mausaktion in Millisekunden. Long Integer.
Spin	True = ein Drehfeld wird eingefügt.
StrictFormat	True = das Format wird während der Eingabe überprüft.
Text	Text, der im Eingabefeld angezeigt wird.
TreatAsNumber	True = der Text wird als Zahl behandelt. Diese Eigenschaft bestimmt die Behandlung der „Effective...“-Eigenschaften.
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.

Eigenschaft	Beschreibung
MouseWheelBehavior	Scrollen mit dem Mausrad über die numerischen Werte. Konstantangruppe com.sun.star.awt.MouseWheelBehavior: SCROLL_DISABLED = 0 (kein Scrollen) SCROLL_FOCUS_ONLY = 1 (Scrollen nur, wenn das Feld den Fokus hat) SCROLL_ALWAYS = 2 (Scrollen immer möglich).
VerticalAlign	Die vertikale Textausrichtung. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.

Das Zahlenformat, mit dem das Eingabefeld kontrolliert wird, muss vom Formatverwalter erzeugt werden. Sie können nicht einfach ein vom Dokument erzeugtes oder irgendein anderes Zahlenformat nehmen. Wenn Sie den Formatverwalter nach enthaltenen Zahlenformaten durchsuchen, benötigen Sie gegebenenfalls den Typ des Formats als Wert der Konstantengruppe NumberFormat (s. Tabelle 306).

Tabelle 306. *Die Konstantengruppe com.sun.star.util.NumberFormat.*

Konstante	Wert	Beschreibung
0	ALL	Alle Zahlenformate.
1	DEFINED	Benutzerdefinierte Zahlenformate.
2	DATE	Datumsformate.
4	TIME	Uhrzeitformate.
8	CURRENCY	Währungsformate.
16	NUMBER	Dezimale Zahlenformate.
32	SCIENTIFIC	Wissenschaftliche Zahlenformate.
64	FRACTION	Bruchzahlenformate.
128	PERCENT	Prozentzahlenformate.
256	TEXT	Textformate.
6	DATETIME	Zahlenformate für Kombinationen von Datum und Uhrzeit.
1024	LOGICAL	Zahlenformate für Wahrheitswerte.
2048	UNDEFINED	Rückgabewert, wenn kein Zahlenformat existiert.

Die Eigenschaft FormatsSupplier (s. Tabelle 305) verfügt über die Methode getNumberFormats(), mit der der Formatverwalter (Interface XNumberFormats) aufgerufen wird. Der Verwalter kennzeichnet die enthaltenen Zahlenformate mit einem numerischen Schlüssel. Die Eigenschaft FormatKey (s. Tabelle 305) benötigt den konkreten Schlüsselwert, der vom Formatverwalter geliefert wird. Mit den Methoden aus der Tabelle 307 ermitteln oder erstellen Sie das Zahlenformat für das formatierte Feld.

Tabelle 307. *Methoden im Interface com.sun.star.util.XNumberFormats.*

Methode	Beschreibung
getByKey(key)	Gibt die Eigenschaften des mit dem Schlüssel (key) indexierten Zahlenformats zurück. Service com.sun.star.util.NumberFormatProperties • FormatString – das Format als String. • Locale – Gebietsschema. • Type – Typ (s. Tabelle 306). • Comment – String zur Information für den Benutzer.

Methode	Beschreibung
queryKeys(NumberFormat, Locale, boolean)	Gibt ein Array von Schlüsseln (Long Integer) für die Zahlenformate des angegebenen Typs NumberFormat (s. Tabelle 306) im angegebenen Gebietsschema Locale zurück. Ist der letzte Parameter True, wird ein Zahlenformat erzeugt, falls keines existierte.
queryKey(format, Locale, boolean)	Gibt den Schlüssel (Long Integer) für den angegebenen Formatstring im angegebenen Gebietsschema zurück, oder -1, falls der Formatstring nicht gefunden wird.
addNew(format, Locale)	Fügt ein benutzerdefiniertes Zahlenformat als String im angegebenen Gebietsschema hinzu und gibt den neuen Formatschlüssel zurück. Long Integer.
addNewConverted(format, Locale, NewLocale)	Fügt im Gebietsschema NewLocale ein Zahlenformat hinzu, dessen String (format) in einem anderen Gebietsschema (Locale) vorkommt.
removeByKey(key)	Löscht das Zahlenformat mit dem angegebenen Schlüssel (Long Integer).
generateFormat(key, Locale, bThousands, bRed, nDecimals, nLeading)	Gibt einen Formatstring zurück, der aus den angegebenen Werten erstellt wird. Es wird aber kein Zahlenformat erzeugt.

Übung

Fügen Sie dem Dialog OOMESample ein formatiertes Feld mit dem Namen MyFormattedField hinzu.

Kopieren Sie die Zeilen aus dem Listing 563 hinter den bereits vorhandenen Code zur Konfiguration innerhalb der Subroutine RunOOMEDlg.

Obwohl es nicht übermäßig schwer ist, zur Suche und Erzeugung eigener Zahlenformate die Methoden in der Tabelle 307 anzuwenden, ist es doch am einfachsten, die Zuordnung eines Zahlenformats zu einem formatierten Feld über den Eigenschaften-Dialog in der IDE vorzunehmen.

Fügen Sie dem Dialog ein formatiertes Feld hinzu und nennen Sie es MyFormattedField. Der Code im Listing 563 initialisiert das Feld so, dass das aktuelle Datum im gewünschten Format als Startwert angezeigt wird. Das Datum wird mit „TT. MMM JJJJ“ formatiert.

Listing 563. *Ein formatiertes Feld als Datum mit dem Format „TT. MMM JJJJ“.*

```
REM Gibt dem formatierten Feld ein bestimmtes Format.
Dim oSupplier     'Das Objekt FormatsSupplier
Dim oFormats      'Der Formatverwalter mit allen Formatobjekten
Dim nKey As Long  'Index des Zahlenformats im Formatverwalter
Dim oLocale As New com.sun.star.lang.Locale
Dim sFormat As String

sFormat = "TT. MMM JJJJ"
oSupplier = oOOMEDlg.getModel().getByName("MyFormattedField").FormatsSupplier
oFormats  = oSupplier.getNumberFormats()

REM Ermittelt den Schlüssel des Zahlenformats. -1, falls es nicht existiert.
nKey = oFormats.queryKey(sFormat, oLocale, True)

REM Wenn das Zahlenformat nicht existiert, wird es hinzugefügt.
If (nKey = -1) Then
  nKey = oFormats.addNew(sFormat, oLocale)
  REM Im unwahrscheinlichen Fall, dass das Hinzufügen scheitert, wird 0 genommen.
  If (nKey = -1) Then nKey = 0
End If

REM Das Datum kann als String oder als Date-Wert gesetzt werden.
REM ...EffectiveValue = "12. Apr 2004"
REM ...EffectiveValue = Date() scheitert, aber folgendes funktioniert:
oOOMEDlg.getModel().getByName("MyFormattedField").EffectiveValue = CStr(Date())
```

```
REM Jetzt wird der Schlüssel für das gewünschte Datumsformat gesetzt,
REM und der Wert soll als Zahl behandelt werden.
oOOMEDlg.getModel().getByName("MyFormattedField").FormatKey = nKey
oOOMEDlg.getModel().getByName("MyFormattedField").TreatAsNumber = True
```

Das Listing 563 enthält einige interessante Dinge. Zum einen wird die Eigenschaft EffectiveValue verwendet anstatt der auch möglichen Eigenschaft Text. Die Eigenschaft TreatAsNumber ist auf True gesetzt, weil das Drehfeld ein Startdatum nicht ordentlich hoch- oder runterzählt, wenn TreatAsNumber auf False steht. Auch wenn man es nicht sieht, das formatierte Feld verwendet im Gegensatz zum Datumsfeld denselben Zahlenwert für ein Datum wie der Basic-Date-Typ. Und schließlich zählt das Drehfeld in einem formatierten Feld immer um den Wert 1 hoch oder runter – im Gegensatz zum Datumsfeld, das jeden Teil des Datums unabhängig hoch- oder runterzählen kann.

Maskiertes Feld

Wenn Sie wollen, dass der Benutzer den Text des Eingabefelds nur mit Einschränkungen bearbeiten kann, verwenden Sie ein maskiertes Feld (Pattern control). Es besteht aus einer Bearbeitungsmaske und einer Zeichenmaske. Die Zeichenmaske ist der String, der beim ersten Erscheinen des Felds angezeigt wird. Der Benutzer kann nun den Text bearbeiten, aber nur in den durch die Bearbeitungsmaske freigegebenen Bereichen. Die Bearbeitungsmaske enthält bestimmte Zeichen zur Kennzeichnung der Werte, die an bestimmten Stellen eingegeben werden dürfen. Zum Beispiel bedeutet das Zeichen L (für Literal), dass der Benutzer an der betreffenden Stelle keine Eingabe tätigen darf, und das Zeichen N (für Numeral), dass der Benutzer an dieser Stelle nur eines der Zeichen 0 bis 9 verwenden darf.

In den USA hat jeder Bürger für die Besteuerung und Sozialversicherung eine Sozialversicherungsnummer. Sie besteht aus drei Ziffern, einem Bindestrich, zwei Ziffern, einem Bindestrich und drei Ziffern. Die Bearbeitungsmaske wird also als „NNNLNNLNNN“ angegeben. Der Benutzer kann somit an den entsprechenden Stellen Ziffern eintragen, kann aber an den anderen Stellen, den Bindestrichen, keine Änderungen vornehmen. Eine Zeichenmaske der Form „___-__-___“ gibt dem Benutzer als Start eine visuelle Form, zu sehen im Listing 564. Das Zeichen „_“ ist willkürlich gewählt. Sie können jedes andere Zeichen nehmen, auch Leerzeichen.

Listing 564. *Bearbeitungsmaske und Zeichenmaske für eine Sozialversicherungsnummer.*

```
EditMask    = "NNNLNNLNNN"
LiteralMask = "___-__-___"
```

Die Anzahl der einzugebenden Zeichen wird durch die Länge der Bearbeitungsmaske bestimmt. Eine unterschiedliche Länge der Bearbeitungsmaske und der Zeichenmaske wird als Fehler angesehen. Tabelle 308 zeigt die vom maskierten Feld unterstützten Zeichen für die Bearbeitungsmaske. Die Eigenschaften des Modells des maskierten Felds sehen Sie in der Tabelle 309.

Tabelle 308. *Die vom maskierten Feld unterstützten Zeichen für die Bearbeitungsmaske.*

Zeichen	Beschreibung
L	Textkonstante, die angezeigt wird, aber nicht bearbeitet werden kann.
a	Die Zeichen a-z oder A- Z sind erlaubt.
A	Die Zeichen a-z und A-Z sind erlaubt, aber a-z wird zu Großbuchstaben konvertiert.
c	Die Zeichen a-z, A-Z und 0-9 sind erlaubt.
C	Die Zeichen a-z, A-Z und 0-9 sind erlaubt, aber a-z wird zu Großbuchstaben konvertiert.
N	Die Zeichen 0-9 sind erlaubt.
x	Jedes druckbare Zeichen ist erlaubt.
X	Jedes druckbare Zeichen ist erlaubt, aber a-z wird zu Großbuchstaben konvertiert.

Tabelle 309. *Eigenschaften im Service com.sun.star.awt.UnoControlPatternFieldModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier. Weitere Eigenschaften:	
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
EditMask	Die Bearbeitungsmaske.
HideInactiveSelection	True = die Auswahl ist verborgen, wenn das maskierte Feld nicht den Fokus hat.
LiteralMask	Die Zeichenmaske.
MaxTextLen	Maximale Anzahl an Zeichen. 0 = die Textlänge ist nicht begrenzt.
ReadOnly	True = der Text kann nicht vom Benutzer geändert werden.
StrictFormat	True = das Format wird während der Eingabe überprüft.
Text	Text, der im Textfeld des maskierten Felds angezeigt wird.
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.
MouseWheelBehavior	Scrollen mit dem Mausrad über die numerischen Werte. Konstantangruppe com.sun.star.awt.MouseWheelBehavior: SCROLL_DISABLED = 0 (kein Scrollen) SCROLL_FOCUS_ONLY = 1 (Scrollen nur, wenn das Feld den Fokus hat) SCROLL_ALWAYS = 2 (Scrollen immer möglich).
VerticalAlign	Die vertikale Textausrichtung. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.

Tipp Wenn StrictFormat False ist, ignoriert das maskierte Feld sowohl die Bearbeitungsmaske als auch die Zeichenmaske.

Beschriftungsfeld

Das Beschriftungsfeld (Fixed text control) dient der Beschriftung anderer Felder und Bereiche des Dialogs. Es ist ein sehr einfaches Texteingabefeld. Die Eigenschaften seines Modells finden Sie in der Tabelle 310.

Tabelle 310. *Eigenschaften im Service com.sun.star.awt.UnoControlFixedTextModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer TabStop. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in dem Beschriftungsfeld: 0: links, 1: zentriert, 2: rechts.
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
Label	Text, der im Beschriftungsfeld angezeigt wird.
MultiLine	True = der Text kann über mehr als eine Zeile gehen.
VerticalAlign	Die vertikale Textausrichtung. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.
NoLabel	True = Keine automatische Kurzwahlbestimmung.

Übung Fügen Sie dem Dialog OOMESample drei Beschriftungsfelder hinzu, jeweils eines links vom Datumsfeld, vom formatierten Feld und vom Kombinationsfeld mit den entsprechenden Titeln „Datum:“, „Format:“ und „Farbe:“.

Dateiauswahl

Die Dateiauswahl (File control) ist ein Texteingabefeld mit einer zusätzlichen Schaltfläche, die den Dateiauswahldialog öffnet. Tabelle 311 enthält die vom Modell der Dateiauswahl definierten Eigenschaften. Das beim ersten Aufruf gewählte Verzeichnis wird der Eigenschaft Text entnommen.

Tabelle 311. *Eigenschaften im Service com.sun.star.awt.UnoControlFileControlModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier. Weitere Eigenschaften:	
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
HideInactiveSelection	True = die Auswahl ist verborgen, wenn die Dateiauswahl nicht den Fokus hat.
ReadOnly	True = der Text kann nicht vom Benutzer geändert werden.
Text	Text, der im Eingabefeld angezeigt wird.
VerticalAlign	Die vertikale Textausrichtung. Optional. Enumeration com.sun.star.style.VerticalAlignment: TOP, MIDDLE oder BOTTOM.

Übung Fügen Sie dem Dialog OOMESample eine Dateiauswahl und links davon ein Beschriftungsfeld mit dem Titel „Dateiauswahl:“ hinzu.

Die Dateiauswahl ist leicht zu handhaben, man kann aber leider keine Filter einstellen. Als Lösung für dieses Problem wird vielfach ein Textfeld in Verbindung mit einer Schaltfläche eingesetzt. Die Schaltfläche öffnet den Dateiauswahldialog FilePicker mit den gewünschten Filtern. Beim Schließen dieses Dialogs wird die gewählte Datei in das Texteingabefeld eingetragen.

Übung Fügen Sie dem Dialog OOMESample ein Textfeld mit dem Namen FileTextfield und daneben eine Schaltfläche mit dem Titel „Datei auswählen“ hinzu.

Kopieren Sie die Subroutine FileButtonSelected im Listing 565 in das Makromodul und verknüpfen Sie damit bei der Schaltfläche das Ereignis „Aktion ausführen“.

Erzeugen Sie ein Textfeld mit dem Namen FileTextField. Setzen Sie neben das Textfeld eine Schaltfläche mit dem Titel „Datei auswählen“ und verknüpfen ihr Ereignis „Aktion ausführen“ mit dem Makro im Listing 565. Dieses Makro öffnet den Dateiauswahldialog FilePicker mit der Verzeichnis- oder Dateiangabe, die als Text im FileTextField steht. Wenn das Textfeld leer ist, prüft das Makro die Konfigurationsinformationen und startet mit dem Arbeitsverzeichnis. Das wirklich Interessante an diesem Beispiel ist, dass mehrere Dateifilter definiert sind, was bei der oben beschriebenen Dateiauswahl nicht möglich ist.

Listing 565. *Auswahl einer Datei.*

```
Sub FileButtonSelected
  Dim oFileDlg           'Dateiauswahldialog
  Dim oSettings          'Objekt zur Ermittlung des Arbeitsverzeichnisses
  Dim sFile As String    'Dateipfad als URL-String
  Dim oFileAccess        'Objekt für den einfachen Dateizugriff
  Dim oFiles             'Der Dateiauswahldialog gibt ein Array der ausgewählten
                         'Dateinamen zurück.

  REM Startwert ist der Inhalt des Textfelds.
  sFile = oOOMEDlg.getModel().getByName("FileTextField").Text
  If sFile <> "" Then
    sFile = ConvertToURL(sFile)
  Else
```

```
    REM Das Textfeld ist leer, also wird das Arbeitsverzeichnis genommen.
    oSettings = CreateUnoService("com.sun.star.util.PathSettings")
    sFile = oSettings.Work
  End If

  REM Der Dateiauswahldialog.
  oFileDlg = CreateUnoService("com.sun.star.ui.dialogs.FilePicker")

  REM Definition der einzusetzenden Filter.
  oFileDlg.AppendFilter("Alle Dateien (*.*)", "*.*")
  oFileDlg.AppendFilter("JPG-Dateien (*.jpg)", "*.jpg")
  oFileDlg.AppendFilter("OOo Writer (*.odt)", "*.odt")
  oFileDlg.AppendFilter("OOo Calc (*.ods)", "*.ods")
  oFileDlg.AppendFilter("OOo Draw (*.odg)", "*.odg")
  oFileDlg.AppendFilter("OOo Impress (*.odp)", "*.odp")
  oFileDlg.SetCurrentFilter("Alle Dateien (*.*)")

  REM Untersuchung, ob der Pfad ein Verzeichnis oder eine Datei ist.
  oFileAccess = CreateUnoService("com.sun.star.ucb.SimpleFileAccess")
  If oFileAccess.exists(sFile) Then
    REM Das "Anzeigeverzeichnis" muss nicht zwingend ein Verzeichnis sein.
    REM Ich könnte etwas Kompliziertes machen wie
    REM If Not oFileAccess.isFolder(sFile) Then extrahiere das Verzeichnis ...,
    REM brauche ich aber nicht!
    oFileDlg.setDisplayDirectory(sFile)
  End If

  REM Ausführung des Dateiauswahldialogs.
  If oFileDlg.execute() Then
    oFiles = oFileDlg.getFiles()
    If UBound(oFiles) >= 0 Then
      sFile = ConvertFromURL(oFiles(0))
      oOOMEDlg.getModel().getByName("FileTextField").Text = sFile
    End If
  End If
End Sub
```

18.5.8. Grafisches Kontrollfeld

Das grafische Kontrollfeld (Image control) dient der Darstellung einer Grafik. Es kann ein Bild automatisch skalieren, so dass es in das Kontrollfeld passt. Tabelle 312 zeigt die Eigenschaften des Modells des grafischen Kontrollfelds. Die Adresse des zu ladenden Bilds steht in der Eigenschaft ImageURL, und die Eigenschaft ScaleImage gibt an, ob das Bild automatisch skaliert werden soll.

Tabelle 312. *Eigenschaften im Service com.sun.star.awt.UnoControlImageControlModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer TextColor, TextLineColor, FontDescriptor, FontEmphasisMark und FontRelief. Weitere Eigenschaften:	
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
ImageURL	Der URL des Bildes, das in dem Kontrollfeld angezeigt wird. Ist das Bild zu groß, wird es abgeschnitten.
Graphic	Grafikobjekt. Interaktion zwischen Graphic und ImageURL: • Wenn ImageURL gesetzt ist, wird Graphic zu dem Objekt, das von dem URL geladen wird, oder NULL, wenn der URL auf keine gültige Grafik zeigt. • Wenn Graphic gesetzt ist, wird aus ImageURL ein leerer String.

Eigenschaft	Beschreibung
ScaleImage	True = das Bild wird automatisch auf die Größe des Felds skaliert.
ScaleMode	Modus der Bildskalierung. ScaleMode verdrängt ScaleImage. Konstantengruppe com.sun.star.awt.ImageScaleMode: NONE = 0 (keine Skalierung) ISOTROPIC = 1 (Skalierung unter Beibehaltung des Seitenverhältnisses) ANISOTROPIC = 2 (Skalierung ohne Rücksicht auf das Seitenverhältnis).

18.5.9. Fortschrittsbalken

Ein Fortschrittsbalken (Progress bar) ist ein gerader Streifen, der beginnend am linken Ende allmählich nach rechts hin mit einer Farbe gefüllt wird. Beim Start ist der Fortschrittsbalken leer, und zum Ende der Arbeit ist er gefüllt. Sie kennen den Fortschrittsbalken am unteren Rand eines OpenOffice-Fensters, wenn Sie auf „Speichern“ klicken, um ein offenes Dokument zu sichern.

Über die Eigenschaften ProgressValueMin und ProgressValueMax teilen Sie dem Kontrollelement den zu beachtenden Wertebereich mit. Wenn der Wert der Eigenschaft ProgressValue gleich dem Minimumwert ist, ist der Balken leer – nichts davon ist gefüllt. Wenn der ProgressValue-Wert in der Mitte zwischen dem Minimal- und dem Maximalwert liegt, ist der Balken zur Hälfte gefüllt. Und wenn schließlich der ProgressValue-Wert gleich dem Maximalwert ist, ist der Balken komplett gefüllt. Der Fortschrittsbalken ist nicht auf die Positionen leer, halb und voll beschränkt. Das waren nur Beispiele. Tabelle 313 enthält die vom Modell des Fortschrittsbalkens bereitgestellten Eigenschaften.

Tabelle 313. *Eigenschaften im Service com.sun.star.awt.UnoControlProgressBarModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer TextColor, TextLineColor, FontDescriptor, FontEmphasisMark, FontRelief und TabStop. Weitere Eigenschaften:	
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
FillColor	Die Füllfarbe des Fortschrittsbalkens. Long Integer.
ProgressValue	Der aktuelle Fortschrittswert. Long Integer.
ProgressValueMax	Der maximale Fortschrittswert. Long Integer.
ProgressValueMin	Der minimale Fortschrittswert. Long Integer.

Tipp Man kann sowohl die Hintergrundfarbe als auch die Füllfarbe setzen.

18.5.10. Listenfeld

Ein Listenfeld (List box control) enthält eine Liste von Einträgen – jeweils einen pro Zeile –, aus der der Benutzer keinen, einen oder mehrere selektieren kann. Ein Listenfeld stellt also einen Mechanismus zur Mehrfachauswahl aus einer Liste dar. Das Listenfeld stellt automatisch, falls erforderlich, Bildlaufleisten zur Verfügung. Tabelle 314 enthält eine Liste der vom Modell des Listenfelds unterstützten Eigenschaften.

Tabelle 314. *Eigenschaften im Service com.sun.star.awt.UnoControlListBoxModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier. Weitere Eigenschaften:	
Align	Die horizontale Textausrichtung in dem Listenfeld (Optional): 0: links, 1: zentriert, 2: rechts.

Eigenschaft	Beschreibung
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
Dropdown	True = das Listenfeld verfügt über einen Aufklappbutton.
LineCount	Die maximale Anzahl der in der Aufklappliste angezeigten Zeilen.
MultiSelection	True = es können mehrere Einträge ausgewählt werden.
ReadOnly	True = das Listenfeld kann nicht vom Benutzer geändert werden.
SelectedItems	Array der Indexwerte der ausgewählten Einträge. Short Integer.
StringItemList	Die Einträge der Liste als Stringarray.
WritingMode	Schreibrichtung. Konstantengruppe com.sun.star.text.WritingMode2: Nur LR_TB = 0 (von links nach rechts) und RL_TB = 1 (von rechts nach links) verfügbar.
MouseWheelBehavior	Scrollen mit dem Mausrad durch die Listeneinträge. Konstantangruppe com.sun.star.awt.MouseWheelBehavior: SCROLL_DISABLED = 0 (kein Scrollen) SCROLL_FOCUS_ONLY = 1 (Scrollen nur, wenn das Feld den Fokus hat) SCROLL_ALWAYS = 2 (Scrollen immer möglich).
ItemSeparatorPos	Position einer horizontalen Trennlinie unter dem entsprechenden Listeneintrag.
TypedItemList	Liste von Objekten. Diese Liste korrespondiert mit der StringItemList und muss dieselbe Länge haben. Seit LibreOffice 5.4

Obwohl ich immer wieder darauf hinweise, die meisten Manipulationen über das Modell durchzuführen, so wird doch einiges über Methoden des Kontrollelements selbst gesteuert. Das Listenfeld verfügt über dieselben Spezialfunktionen wie das Kombinationsfeld zum Hinzufügen und Entfernen von Einträgen an beliebigen Positionen (s. Tabelle 292). Das Listenfeld unterstützt noch andere nützliche Methoden, die vor allem mit der Benutzerkommunikation zu tun haben (s. Tabelle 315).

Tabelle 315. *Methoden im Interface com.sun.star.awt.XListBox.*

Methode	Beschreibung
addItemListener(XItemListener)	Fügt einen Beobachter für Statusänderungen von Einträgen hinzu.
removeItemListener(XItemListener)	Entfernt einen Beobachter für Statusänderungen von Einträgen.
addActionListener(XActionListener)	Fügt einen Beobachter für Aktionen hinzu.
removeActionListener(XActionListener)	Entfernt einen Beobachter für Aktionen.
addItem(String, position)	Fügt einen Eintrag an der angegebenen Position ein.
addItems(StringArray, position)	Fügt mehrere Einträge an der angegebenen Position ein.
removeItems(position, count)	Löscht eine Anzahl von Einträgen an der angegebenen Position.
getItemCount()	Gibt die Anzahl der Einträge im Listenfeld zurück.
getItem(position)	Gibt den String des Eintrags an der angegebenen Position zurück.
getItems()	Gibt die Eigenschaft StringItemList zurück (s. Tabelle 314).
getSelectedItemPos()	Gibt die Position des aktuell ausgewählten Eintrags zurück. Dann nützlich, wenn nur ein Eintrag ausgewählt werden darf.
getSelectedItemsPos()	Gibt die Eigenschaft SelectedItems zurück (s. Tabelle 314).
getSelectedItem()	Gibt den String des aktuell ausgewählten Eintrags zurück. Dann nützlich, wenn nur ein Eintrag ausgewählt werden darf.
getSelectedItems()	Gibt ein Stringarray aller ausgewählten Einträge zurück.
selectItemPos(position, boolean)	Setzt oder entfernt die Auswahl des Eintrags an der angegebenen Position.
selectItemsPos(positions, boolean)	Setzt oder entfernt die Auswahl mehrerer Einträge an den im Array angegebenen Positionen.

Methode	Beschreibung
selectItem(item, boolean)	Setzt oder entfernt die Auswahl des Eintrags mit dem angegebenen String.
isMultipleMode()	Gibt die Eigenschaft MultiSelection zurück (s. Tabelle 314).
setMultipleMode(boolean)	Setzt die Eigenschaft MultiSelection (s. Tabelle 314).
getDropDownLineCount()	Gibt die Eigenschaft LineCount zurück (s. Tabelle 314).
setDropDownLineCount(integer)	Setzt die Eigenschaft LineCount (s. Tabelle 314).
makeVisible(position)	Scrollt die Einträge im Listenfeld, so dass der Eintrag an der angegebenen Position sichtbar ist.

18.5.11. Horizontale und vertikale Bildlaufleiste

Die Bildlaufleiste (Scroll bar) dient dem Scrollen eines Kontrollelements oder eines Dialogs, in dem nicht schon eine Bildlaufleiste enthalten ist oder unterstützt wird. Die ganze Arbeit der Steuerung eines anderen Objekts und der Synchronisierung dieses Objekts mit der Bildlaufleiste muss über Makros geleistet werden. Die Bildlaufleiste kann nämlich nicht automatisch zur Steuerung mit einem anderen Objekt verbunden werden. Diesen organisatorischen Zusammenhang kann nur ein Makro aufbauen. Schreiben Sie also ein Makro, das auf die Ereignisse der Bildlaufleiste anspricht und dann auf der Grundlage des aktuellen Status ein anderes Objekt beeinflusst. Wenn Sie andererseits wollen, dass die Bildlaufleiste die Änderungen an einem anderen Objekt nachvollzieht, müssen Sie Handlers für das andere Objekt schreiben, die auf Aktionen reagieren und entsprechende Aktualisierungen in der Bildlaufleiste vornehmen. Sowohl als Input als auch als Output dient die Bildlaufleiste der Abbildung einer Beziehung zu einem anderen Objekt. Die angezeigte Beziehung und die zugehörige Aktion benötigen ein benutzerdefiniertes Makro. Tabelle 316 zeigt die Eigenschaften des Modells der Bildlaufleiste.

Tabelle 316. *Eigenschaften im Service com.sun.star.awt.UnoControlScrollBarModel.*

Eigenschaft	Beschreibung
Die in der Tabelle 278 aufgeführten Eigenschaften gelten auch hier, außer TextColor, TextLineColor, FontDescriptor, FontEmphasisMark und FontRelief. Weitere Eigenschaften:	
BlockIncrement	Zählt den Wert (Long Integer) für die Weiterbewegung eines Blocks hoch oder runter.
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
LineIncrement	Zählt den Wert (Long Integer) für die Weiterbewegung einer einfachen Linie hoch oder runter.
LiveScroll	True = der Fensterinhalt wird während des Scrollens aktualisiert. False = der Fensterinhalt wird aktualisiert, wenn der Benutzer die Maustaste loslässt.
Orientation	Ausrichtung der Bildlaufleiste.Mögliche Werte: com.sun.star.awt.ScrollBarOrientation.HORIZONTAL = 0 und com.sun.star.awt.ScrollBarOrientation.VERTICAL = 1.
RepeatDelay	Der zeitliche Abstand zwischen Klickwiederholungen, in Millisekunden.
ScrollValue	Aktueller Scrollwert.
ScrollValueMax	Maximaler Scrollwert.
ScrollValueMin	Minimaler Scrollwert.
SymbolColor	Farbe (RGB) der Symbole, die als Teile der Bildlaufleiste erscheinen, zum Beispiel die Pfeilflächen.
VisibleSize	Sichtbare Größe der Bildlaufleiste. Long Integer.

Übung

Fügen Sie dem Dialog OOMESample eine Bildlaufleiste mit dem Namen ButtonScrollBar hinzu.

Fügen Sie dem Modul die globale Variable oBPosSize (Listing 566) hinzu.

Kopieren Sie die Zeilen aus dem Listing 567 hinter den bereits vorhandenen Code zur Konfiguration innerhalb der Subroutine RunOOMEDlg.

Kopieren Sie die Subroutine ScrollButton im Listing 568 in das Makromodul und verknüpfen Sie damit das Ereignis „Beim Justieren“.

Zur Illustration, wie eine Bildlaufleiste funktioniert, fügen Sie dem Beispieldialog eine horizontale Bildlaufleiste mit dem Namen ButtonScrollBar hinzu. Die ganze Arbeit leistet ein Makro. Eine Bildlaufleiste hat keine Ahnung, ob etwas waagerecht oder senkrecht angeordnet ist. Man kann also für denselben Zweck entweder eine horizontal oder eine vertikal ausgerichtete Bildlaufleiste wählen. Wenn es auch dem Benutzer schräg vorkäme, eine Leiste senkrecht zu bewegen, um etwas waagerecht zu verschieben – der Bildlaufleiste ist das völlig egal. Ich habe hier die horizontale Variante genommen, weil ich die „Schließen“-Schaltfläche waagerecht im Dialog verschieben möchte. Fügen Sie dem Modul eine globale Variable hinzu, in der die Startposition und die Größe der Schaltfläche gespeichert sind (s. Listing 566).

Listing 566. *Globale Variable für die Position und die Größe der Bildlaufleiste.*

```
Dim oBPosSize    'Ursprüngliche Position und Größe der Schaltfläche.
```

In meinem Beispieldialog ist viel freier Platz zwischen der Schließen-Schaltfläche und dem rechten Rand des Dialogs (s. Bild 148). Der Plan ist, dass die Bildlaufleiste die Schaltfläche nach rechts und links verschiebt.

Der Standardwert für das Maximum der Bildlaufleiste ist 100. In den meisten Fällen wird das als 100% des Scrollbereichs verwendet. In dem Beispieldialog habe ich aber die höchste Strecke berechnet, die von der Schließen-Schaltfläche zurückgelegt werden kann, und der Bildlaufleiste diesen Wert als maximalen Rückgabewert zugewiesen. Somit kann ich den aktuellen Wert der Leiste direkt verwenden und brauche nicht eine Position auf prozentualer Grundlage zu errechnen, was allerdings ansonsten die Regel ist.

Fügen Sie an den schon vorhandenen Konfigurationscode die Zeilen aus dem Listing 567 an, der die Startwerte für die Position und die Größe der Schließen-Schaltfläche speichert. Diese Information wird später als Bezugspunkt benötigt. Weiterhin wird berechnet, wie weit die Schaltfläche abhängig von ihrer Position und Größe sowie der Breite des Dialogs nach rechts verschoben werden kann.

Listing 567. *Speichert die Startposition der Schaltfläche und legt den Maximalwert der Bildlaufleiste fest.*

```
REM Die Startposition und die Größe der Schließen-Schaltfläche.
oBPosSize = oOOMEDlg.getControl("CloseButton").getPosSize()
REM Der maximale Wert, den die Bildlaufleiste annehmen kann, auf der Grundlage
REM der maximalen Strecke, die die Schließen-Schaltfläche zurücklegen kann.
oOOMEDlg.getModel().getByName("ButtonScrollBar").ScrollValueMax = _
     oOOMEDlg.getPosSize.Width - oBPosSize.X - oBPosSize.Width - 10
```

Verknüpfen Sie das Ereignis „Beim Justieren“ der Bildlaufleiste mit dem Makro ScrollButton im Listing 568. Diese Prozedur positioniert die Schließen-Schaltfläche abhängig vom Wert der Eigenschaft ScrollValue des Modells der Bildlaufleiste.

Listing 568. *Positionierung der Schließen-Schaltfläche beim Justieren der Bildlaufleiste.*

```
Sub ScrollButton
  Dim oModel       'Modell der Bildlaufleiste
  Dim oButton      'Schließen-Schaltfläche
  Dim newX As Long 'Neue X-Position der Schließen-Schaltfläche

  oModel = oOOMEDlg.getModel().getByName("ButtonScrollBar")
  oButton = oOOMEDlg.getControl("CloseButton")
```

```
    REM Die Bildlaufleiste wurde so eingestellt, dass ihr Maximalwert
    REM gleich dem gewünschten maximalen Verschiebeabstand ist.
    newX = oBPosSize.X + oModel.ScrollValue
    oButton.setPosSize(newX, 0, 0, 0, com.sun.star.awt.PosSize.X)
End Sub
```

Nun können Sie durch das Verschieben der Bildlaufleiste die Schließen-Schaltfläche nach rechts und links bewegen.

18.6. Mehrseitige Dialoge (V. Lenhardt)

Wenn ein Dialog mehr Steuerelemente benötigt, als auf einer Seite überschaubar dargestellt werden können, muss man entweder mehrere Dialoge konzipieren oder in einem einzigen Dialog mehrere Seiten anbieten. Letzteres ist sicher benutzerfreundlicher. Man gruppiert inhaltlich zusammengehörende Steuerelemente so, dass die einzelnen Gruppen durch Seitenwechsel aufgerufen werden. Das Schließen des Dialogs bedeutet dann, dass damit auch alle Seiten geschlossen werden.

OOo bietet zwei verschiedene Funktionalitäten für Mehrseitigkeit, die beide ihre Vor- und Nachteile haben. Das sind zum einen die Steps, deren Name daher rührt, dass sie gerne dazu verwendet werden, Benutzer Schritt für Schritt durch Einstellungsprozesse zu führen. Die andere Methode verwendet Tabs, also Karteireiter, zum einfachen Umschalten zu den diversen Steuerelementgruppen.

18.6.1. Steps

Ein Beispiel für Steps ist die OOo-Makrobibliothek ImportWizard. Probieren Sie es aus: Die Subroutine Main() im Modul Main öffnet einen Dialog, der Hilfestellung bei der Konvertierung mehrerer Dokumente bietet. Die erste Seite (Step 1) lässt Sie die Dokumenttypen auswählen, die importiert werden sollen. Auf der zweiten Seite (Step 2) können Sie das Verzeichnis wählen, in denen sich die Dokumente befinden.

Das Modell des Dialogs enthält die Eigenschaft Step, die dem Dialog die aktuelle Seite des Ablaufs mitteilt. Wenn der Step-Wert 0 ist – das ist der Standardwert –, dann werden alle definierten Steuerelemente im Dialog angezeigt. Das Modell eines jeden Kontrollelements enthält ebenfalls die Eigenschaft Step. In einem Steuerelement bestimmt der Step-Wert, auf welcher Seite es erscheint. Wenn ein Kontrollelement den Step-Wert 0 hat, wird es auf jeder Seite angezeigt. Beispielsweise sollte eine Abbrechen-Schaltfläche wohl auf allen Seiten sichtbar sein.

Ein Vorteil von Steps liegt hauptsächlich darin, dass ein solcher Dialog komplett in der IDE zusammengestellt werden kann. Wenn Sie Kontrollelemente für eine bestimmte Seite entwerfen, so ist es ratsam, die Step-Eigenschaft des Dialogs auf diese Seite einzustellen, damit im Entwurf die Steuerelemente aller anderen Seiten verborgen sind.

Ein weiterer Vorteil von Steps ist ihre Variabilität, da das Umschalten über Prozeduren geschieht, die über Schaltflächen gestartet werden. Üblicherweise werden auf einer Seite die Schaltflächen angezeigt, über die man auf die vorhergehende und auf die folgende Seite wechseln kann. Man kann den Wechsel aber auch anders gestalten, beispielsweise mit Listen oder mit Schaltflächen in einem Design, der Tabs imitiert.

Ich habe ein einfaches Beispiel mit zwei Seiten bereitgestellt (s. Listing 569 und Bild 149), das in der Wirkung mit einem Dialog vergleichbar ist, den ich im nächsten Abschnitt mit Tabs vorstelle.

Der Dialog enthält die Schaltfläche „Beenden“, deren Modell die Step-Eigenschaft 0 hat, so dass sie auf beiden Seiten sichtbar ist. Die Modelle der Schaltfläche „Seite 2 >“ sowie des Listenfelds einerseits haben die Eigenschaft Step 1, die der Schaltfläche „< Seite 1“ und der Dateiauswahl andererseits den Stepwert 2, so dass diese Elemente nur angezeigt werden, wenn das Modell des Dialogs den entsprechenden Step enthält. Die Schaltflächen „< Seite 1“ bzw. „Seite 2 >“ sind verknüpft mit den

Subroutinen SwitchToPage1 bzw. SwitchToPage2, die den Wechsel zur entsprechenden Seite vornehmen. Die Schaltfläche „Beenden“ ist verknüpft mit der Subroutine ClosePagesDlg, die den Dialog beendet und eine Auswertung der Eingaben vornimmt.

Der Dialog startet mit Step 1.

Listing 569. *Mehrseitiger Dialog mit Steps.*

```
Dim oStepsDialog     'Der Dialog StepsDialog

Sub RunStepsDlg
  DialogLibraries.loadLibrary("OOME_40")
  oStepsDialog = CreateUnoDialog(DialogLibraries.OOME_40.StepsDialog)
  oStepsDialog.execute()
End Sub

Sub SwitchToPage1
  oStepsDialog.Model.Step = 1
  oStepsDialog.getControl("ListBox1").setFocus()
End Sub

Sub SwitchToPage2
  oStepsDialog.Model.Step = 2
  oStepsDialog.getControl("FileControl1").setFocus()
End Sub

Sub ClosePagesDlg
  Dim s$
  oStepsDialog.endExecute()
  'Auswertung der Einträge:
  s = "Komponente: " & oStepsDialog.getControl("ListBox1").SelectedItem & Chr(13) _
      & "Datei: " & oStepsDialog.getControl("FileControl1").Text
  MsgBox s, "Auswahl"
End Sub
```

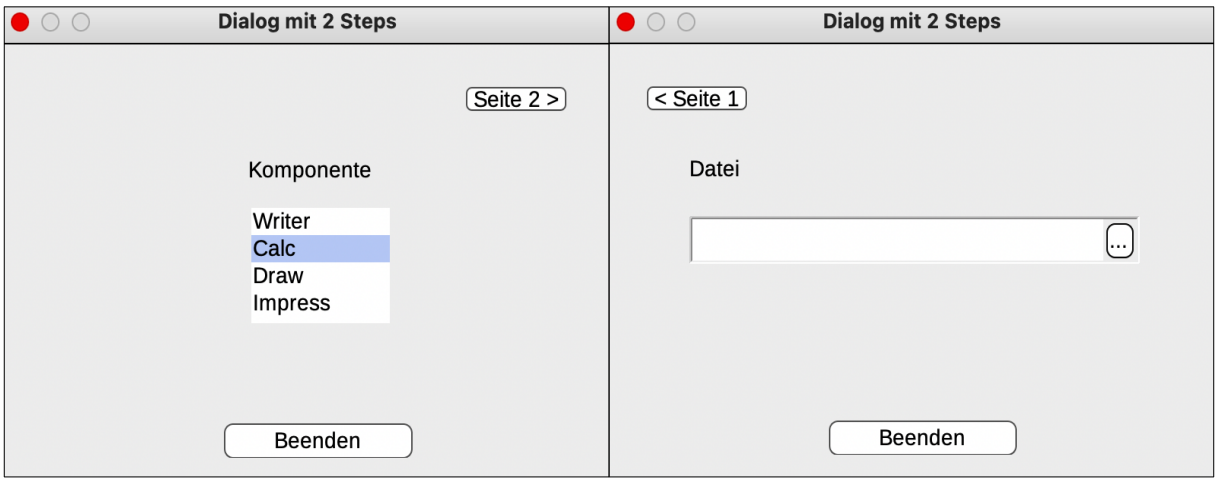


Bild 149. *Der mit Steps konzipierte Dialog StepsDialog (links Step 1, rechts Step 2).*

18.6.2. Tabs

Tabs (Karteireiter) erleichtern die Umschaltung zu den einzelnen Seiten, denn der Vorgang wird beim Klicken automatisch vom Tab-Containerobjekt organisiert. Man benötigt dazu keine selbst programmierten Prozeduren.

Tabs wurden erst mit OOo 3.4 eingeführt.

Im Gegensatz zu normalen Steuerelementen sind die Tab-Services und -Interfaces in einem eigenen Modul zusammengefasst: com.sun.star.awt.tab. Das Tab-Containerobjekt ist kein eigenes Dialogob-

jekt, es ist ein Steuerelement, das in einen Dialog eingefügt werden kann, in dem sich auch andere Steuerelemente befinden. Das Tab-Containerobjekt wiederum enthält die einzelnen Tab-Seiten-Objekte. Das Besondere daran ist, dass die einzelnen Tab-Seiten schließlich Steuerelemente aufnehmen, die dem Dialog selbst nicht direkt bekannt sind.

Tabelle 317. *Services und Interfaces im Modul com.sun.star.awt.tab.*

Service / Interface	Beschreibung
UnoControlTabPage	Service. Eine einzelne Tab-Seite.
UnoControlTabPageModel	Service. Modell einer einzelnen Tab-Seite. Erbt alle Eigenschaften vom Interface XTabPageModel.
UnoControlTabPageContainer	Service. Steuerelement TabPageContainer. Kontrolliert Tab-Seiten.
UnoControlTabPageContainerModel	Service. Modell des Steuerelements TabPageContainer. Es wird dem Modell des Dialogs hinzugefügt.
XTabPage	Interface. Weder Methoden noch Eigenschaften definiert.
XTabPageModel	Interface. Keine Methoden, nur Eigenschaften.
XTabPageContainer	Interface. Methoden für das Steuerelement TabPageContainer.
XTabPageContainerModel	Interface. Methoden für das TabPageContainer-Modell.
XTabPageContainerListener	Interface. Wird vom Interface XTabPageContainer für Informationen über den Wechsel der Tab-Seiten genutzt.

Einen Tab-Container können Sie leider nicht direkt über die Steuerelemente-Toolbox der IDE in Dialoge einbauen. Sie müssen ihn, und damit auch alle einzufügenden Steuerelemente, nachträglich per Makro definieren und in den Dialog integrieren. Eine praktische Kompromisslösung besteht darin, das Design des Dialogs ohne Tab-Container in der IDE vorzunehmen und dann nur noch die Tab-Seiten in der Prozedur hinzuzufügen, die den Dialog startet.

Ein Tab-Container verfügt über mehrere Methoden und eine Eigenschaft, die im Interface XTabPageContainer definiert sind. Die einzige Eigenschaft ist ActiveTabPageID, mit der die ID der aktiven Tab-Seite gelesen oder gesetzt wird. Die Methoden zeigt die Tabelle 318.

Tabelle 318. *Methoden im Interface com.sun.star.awt.tab.XTabPageContainer*

Methode	Beschreibung
getTabPage(index)	Gibt die Tab-Seite mit der Indexnummer als Argument zurück.
getTabPageByID(id)	Gibt die Tab-Seite mit der ID als Argument zurück.
getTabPageCount()	Gibt die Anzahl der Tab-Seiten zurück.
isTabPageActive(index)	True, wenn die Tab-Seite mit der Indexnummer als Argumen aktiv ist.
addTabPageContainerListener(listener)	Meldet einen XTabPageContainerListener an. Er bietet nur eine Methode: tabPageActivated().
removeTabPageContainerListener(listener)	Meldet den Listener ab.

Sie benötigen aber auch das Modell des TabPageContainers. Es unterstützt die allgemeinen Eigenschaften von Steuerelementen, s. Tabelle 277. Die weiteren Eigenschaften zeigt die Tabelle 319.

Zur Erinnerung: Das Modell eines Steuerelements beschreibt dessen Verhalten und die Darstellung im Dialog. Im Abschnitt 18.2. Dialoge und Steuerungsmuster wird der Zusammenhang zwischen Modell und Präsentation genauer beschrieben.

Tabelle 319. *Eigenschaften im Service com.sun.star.awt.tab.XTabPageContainerModel*

Eigenschaft	Beschreibung
BackgroundColor	Long Integer. Hintergrundfarbe (RGB).

Eigenschaft	Beschreibung
Border	Rahmentyp: 0 = kein Rahmen, 1 = 3D-Rahmen, 2 = einfacher Rahmen.
BorderColor	Long Integer. Rahmenfarbe (RGB).
DefaultControl	String. Der Servicename des Standardsteuerelements für dieses Modell.
Enabled	Boolean. True = aktiviert.
HelpText	String. Der Hilfetext.
HelpURL	String. Der URL-String zum Hilfetext.
Printable	Boolean. True = das Element wird in der Druckausgabe erscheinen.

Im Abschnitt 18.8. Das Beispiel Objektinspektor wird im Einzelnen dargelegt, wie ein kompletter Dialog zur Laufzeit erstellt wird. An dieser Stelle aber geht es nur darum, wie in einen vorhandenen Dialog Steuerelemente zur Laufzeit eingefügt werden. Der entscheidende Punkt ist, dass dieser Vorgang auf der Modellebene geschieht.

1. In das Modell des Dialogs wird das Modell des Tab-Containers eingefügt.
2. In das Modell des Tab-Containers werden die Modelle der Tab-Seiten eingefügt.
3. In die Modelle der Tab-Seiten werden die Modelle der weiteren Steuerelemente eingefügt.

Tab-Seiten werden durch den Service com.sun.star.awt.tab.UnoControlTabPage definiert. Sie verhalten sich im Wesentlichen wie Dialoge, mit denselben Interfaces, die Sie schon aus den Tabellen 272, 273, 274, 276 und 281 kennen.

Das Modell einer Tab-Seite wird im Service com.sun.star.awt.tab.UnoControlTabPageModel definiert. Interessanterweise erhält es seine Eigenschaften aus einem Interface, das keine Methoden kennt, nämlich aus com.sun.star.awt.tab.XTabPageModel, s. Tabelle 320.

Tabelle 320. *Eigenschaften im Interface com.sun.star.awt.tab.XTabPageModel.*

Eigenschaft	Beschreibung
TabPageID	Short Integer. Die ID für die Tab-Seite. Nur lesender Zugriff.
Enabled	Boolean. Die Tab-Seite ist aktiviert oder deaktiviert.
Title	String. Der Text, der für die Seite in der Tab-Leiste erscheint.
ImageURL	String. Der URL der Grafik, die für die Seite in der Tab-Leiste angezeigt wird.
ToolTip	String. Der Tooltip für die Seite, der in der Tab-Leiste zur Verfügung steht.

Im folgenden Listing 570 dient der in diesem Dokument definierte Dialog „TabsDialog“ als Grundlage. Er enthält nur ein Steuerelement, die „Beenden“-Schaltfläche, die mit der Subroutine CloseTabsDialog verknüpft ist, die den Dialog beendet und eine Auswertung der Eingaben vornimmt.

In diesen Dialog wird ein UnoControlTabPageContainer mit zwei Tab-Seiten eingefügt, die jeweils zwei Steuerelemente enthalten.

Achtung In einigen LO-Versionen funktionierte das nicht. Das Makro stoppte mit einem Laufzeitfehler in der Zeile

```
oTabPageModel1.insertByName("Label1", oFixedTextModel1)
```

Da der Bug mittlerweile behoben ist, verzichte ich hier auf die Vorstellung eines Workarounds, der sich ohnehin auf unveröffentlichte Services stützte.

Listing 570. *Mehrseitiger Dialog mit Tabs.*

```
Dim oTabsDlg      'Der Dialog TabsDialog

Sub RunTabsDialog
  Dim oDialogModel           'Das Modell des Dialogs
```

```
Dim oTabPageContainerModel 'Das Modell des Tab-Containers
Dim oTabPageModel(1 To 2)  'Die Modelle der Tabseiten
Dim oListBoxModel          'Das Modell des Listenfelds
Dim oFixedTextModel1       'Das Modell des ersten Beschriftungsfelds
Dim oFixedTextModel2       'Das Modell des zweiten Beschriftungsfelds
Dim oFileControlModel      'Das Modell der Dateiauswahl
Dim i%

DialogLibraries.loadLibrary("OOME_40")
'Der Dialog wurde ohne das Tab-Element in der IDE entworfen.
oTabsDlg = CreateUnoDialog(DialogLibraries.OOME_40.TabsDialog)
oDialogModel = oTabsDlg.getModel()
'UnoControlTabPageContainerModel ist der Service des Modells des Tab-Containers
oTabPageContainerModel = oDialogModel.CreateInstance _
                         ("com.sun.star.awt.tab.UnoControlTabPageContainerModel")

'Die Eigenschaften des Modells des Tab-Containers
With oTabPageContainerModel
  .Name = "Tab"      :  .PositionX = 5
  .PositionY = 5     :  .Width = 170
  .Height = 100      :  .TabIndex = 0
  .Border = 1        :  .BackgroundColor = RGB(240, 240, 240)
End With

'Der Tab-Container wird in den Dialog eingefügt.
'Das geschieht über das Modell.
oDialogModel.insertByName("Tab", oTabPageContainerModel)

'Die Modelle der Tab-Seiten werden vom Dialog-Modell erzeugt.
'Ihre Eigenschaften werden festgelegt.
'Sie werden in das Container-Modell eingefügt - in dieser Reihenfolge.
For i = 1 To 2
  'Die createInstanceWithArguments-Argumente werden immer als Array angegeben.
  'Hier wird eine Zahl (Short Integer) als ID-Nummer der Tab-Seite erwartet.
  oTabPageModel(i) = oDialogModel.createInstanceWithArguments _
                     ("com.sun.star.awt.tab.UnoControlTabPageModel", Array(i))
  oTabPageModel(i).Name = "Page" & i
  oTabPageModel(i).Title = "Seite " & i
  oTabPageContainerModel.insertByIndex(i - 1, oTabPageModel(i))
Next

'Aktiviert die Tab-Seite 1.
oTabsDlg.getControl("Tab").ActiveTabPageID = oTabPageModel(1).TabPageID

'Die Modelle der Steuerelemente für die einzelnen Seiten werden
'von den Tab-Seiten erzeugt, und ihre Eigenschaften werden festgelegt.

'Das Beschriftungsfeld für die Tab-Seite 1.
oFixedTextModel1 = oTabPageModel(1).createInstance _
                   ("com.sun.star.awt.UnoControlFixedTextModel")
With oFixedTextModel1
  .PositionX = 60     :  .PositionY = 15
  .Width = 42         :  .Height = 8
  .Label = "Komponente"
End With
```

```
    'Das Listenfeld für die Tab-Seite 1.
    oListBoxModel = oTabPageModel(1).createInstance _
                    ("com.sun.star.awt.UnoControlListBoxModel")
    With oListBoxModel
      .PositionX = 60           :  .PositionY = 30
      .Width = 45               :  .Height = 38
      .StringItemList = Array("Writer", "Calc", "Draw", "Impress")
      .MultiSelection = False   :  .SelectedItems = Array(1)
    End With

    'Das Beschriftungsfeld für die Tab-Seite 2.
    oFixedTextModel2 = oTabPageModel(2).createInstance _
                       ("com.sun.star.awt.UnoControlFixedTextModel")
    With oFixedTextModel2
      .PositionX = 10     :  .PositionY = 15
      .Width = 42         :  .Height = 8
      .Label = "Datei"
    End With

    'Die Dateiauswahl für die Tab-Seite 2.
    oFileControlModel = oTabPageModel(2).createInstance _
                        ("com.sun.star.awt.UnoControlFileControlModel")
    With oFileControlModel
      .PositionX = 10     :  .PositionY = 30
      .Width = 140        :  .Height = 15
    End With

    'Fügt die Modelle der Steuerelemente in die Modelle der Tab-Seiten ein.
    oTabPageModel(1).insertByName("Label1", oFixedTextModel1)
    oTabPageModel(1).insertByName("ListBox1", oListBoxModel)
    oTabPageModel(2).insertByName("Label2", oFixedTextModel2)
    oTabPageModel(2).insertByName("FileControl1", oFileControlModel)

    oTabsDlg.getControl("Tab").Controls(0).setFocus()
    oTabsDlg.execute()
End Sub

Sub CloseTabsDialog
    'Ist über die IDE mit der "Beenden"-Schaltfläche verknüpft.
    Dim oContainer   'Der Tab-Container als Steuerelement des Dialogs
    Dim oPage1       'Die erste Tab-Seite als Steuerelement des Containers
    Dim oPage2       'Die zweite Tab-Seite als Steuerelement des Containers
    Dim s$           'Ausgabestring

    oTabsDlg.endExecute()  'Schließt den Dialog.

    'Auswertung der Einträge:
    oContainer = oTabsDlg.getControl("Tab")
    oPage1 = oContainer.getTabPage(0)
    oPage2 = oContainer.getTabPage(1)
    'oder
    'oPage1 = oContainer.getTabPageByID(1) : oPage2 = oContainer.getTabPageByID(2)
    'bzw.
    'oPage1 = oContainer.Controls(0)       : oPage2 = oContainer.Controls(1)

    s = "Komponente: " & oPage1.getControl("ListBox1").SelectedItem & Chr(10) _
        & "Datei: " & oPage2.getControl("FileControl1").Text
```

```
    MsgBox s, "Auswahl"
End Sub
```

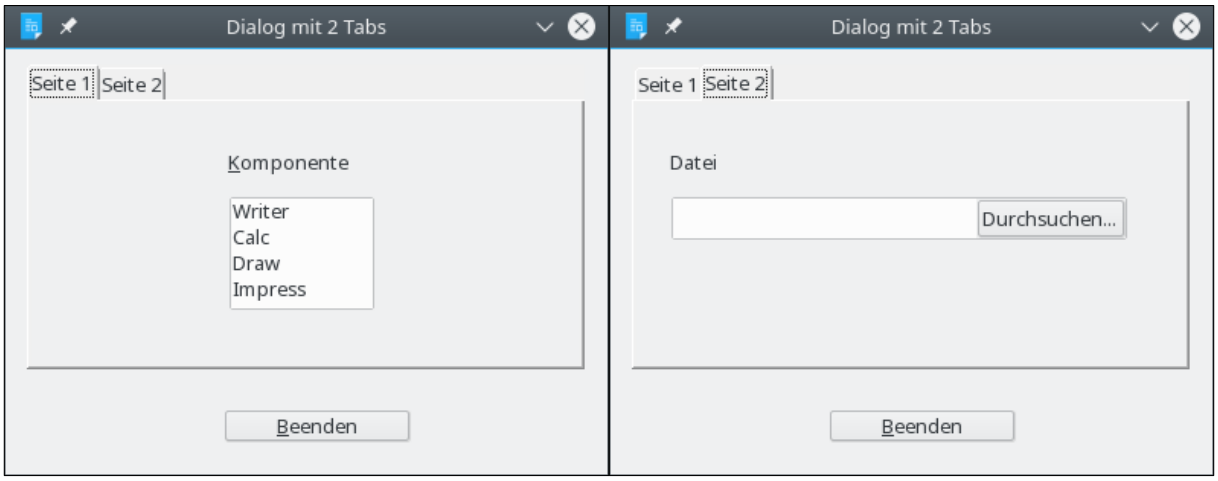


Bild 150. *Der mit Tabs konzipierte Dialog TabsDialog.*

18.7. Modale und Nichtmodale Dialoge (V. Lenhardt)

Wenn ein Dialog mit CreateUnoDialog erzeugt wurde, existiert er in seiner Umgebung, und man kann auf seine Bestandteile zugreifen, lesend und schreibend, auch wenn er nicht sichtbar ist. Er lebt so lange wie seine Umgebung, sei es eine Prozedur oder die laufende Anwendung, je nachdem, wo er definiert wurde. Der Nutzer kann allerdings nur mit einem sichtbaren Dialog kommunizieren.

Die bisher vorgestellten Dialoge wurden allesamt mit der Methode execute() gestartet. Der Fokus geht auf den Dialog über und kann nicht auf andere Fenster gesetzt werden. Er blockiert weitere Aktivitäten der Hauptanwendung (oder des Ausgangsfensters) so lange, bis er mit endExecute() geschlossen wird. Manipulationen an den Elementen eines Dialogs können in dieser Zeit nur von Ereignissen des Dialogs selber vorgenommen werden. Eine OK- beziehungsweise Abbrechen-Schaltfläche (oder mit einem Element verknüpfte Prozedur) sorgt für den entsprechenden Rückgabewert -1 oder 0. Ein solcher Dialog wird „modal“ genannt. Sie kennen das Verhalten auch von einer MsgBox, die erst geschlossen werden muss, damit der Makrofluss weitergehen kann.

Sie kennen aber auch Dialoge, die es erlauben, den Cursor woanders hin zu setzen, zum Beispiel den Suchen-und-Ersetzen-Dialog. Während dieser Dialog auf dem Bildschirm zu sehen ist, verdeckt er zwar einen Teil des darunter liegenden Textfensters, man kann aber den Cursor in den Text setzen, um Markierungen oder Startpositionen festzulegen. Solche Dialoge nennt man „nichtmodal“ (englisch modeless).

Dialoge sind nichtmodal, wenn sie nicht mit execute() gestartet und mit endExecute() geschlossen, sondern mit der Dialogmethode setVisible(True) sichtbar und mit setVisible(False) wieder unsichtbar gemacht werden. Es gibt dann auch keine Rückgabewerte. Entsprechende OK- oder Abbrechen-Schaltflächen sind unwirksam, ebenso das Abbrechen-Kreuzchen in der oberen Dialogecke. Da der Dialog den Programmfluss nicht blockiert, können Manipulationen an dessen Elementen aus dem laufenden Makro heraus vorgenommen werden.

18.7.1. Countdown als nichtmodaler Dialog

Als Beispieldialog zeigt das folgende Listing 571 einen Countdown im Sekundentakt, den der Nutzer abbrechen kann. Das lässt sich mit wait oder MsgBox nicht realisieren. Der Dialog existiert nur innerhalb der Function CountDownDialog. Er wird darin definiert und zugleich mit dem Ende der Funktion auch wieder entfernt. Er enthält nur ein Beschriftungsfeld „TimeLeftLabel“ zur Anzeige der Restzeit und eine Befehlsfläche „StopButton“ zum Abbrechen des Countdowns, die mit dem Sub

SetStopButtonIsPressed verknüpft ist. Für den eigentlichen Countdown habe ich zwei verschiedene Arbeitsweisen bereitgestellt, die je nach Prozessorlast etwas unterschiedliche Eigenarten aufweisen. Wenn Sie wollen, testen Sie, was Ihnen besser zusagt.

Listing 571. *Countdown als nichtmodaler Dialog.*

```
Dim bStopButtonIsPressed As Boolean

Sub SetStopButtonIsPressed
  REM Wird aufgerufen, wenn die Stop-Schaltfläche aktiviert wird.
  bStopButtonIsPressed = True
End Sub

Sub StartCountDownDialog
  If CountDownDialog(10) Then
    MsgBox "Countdown beendet"
  Else
    MsgBox "Countdown abgebrochen"
  End If
End Sub

Function CountDownDialog(iSeconds%) As Boolean
  REM iSeconds - Countdownzeit in Sekunden

  Dim oCountDownDlg      'Die Variable des Dialogobjekts
  Dim i%                 'Countdown-Zähler
  Dim oLabelModel        'Das Label zur Restzeitanzeige

  bStopButtonIsPressed = False

  DialogLibraries.loadLibrary("OOME_40")
  oCountDownDlg = CreateUnoDialog(DialogLibraries.OOME_40.CountdownDialog)
  oCountDownDlg.setVisible(True)  'Macht den Dialog nichtmodal sichtbar.
  oLabelModel = oCountDownDlg.getControl("TimeLeftLabel").Model

  CountDownWithNow(iSeconds, oLabelModel)
'  CountDownWithWait(iSeconds, oLabelModel)

  oCountDownDlg.setVisible(False)  'Macht den Dialog wieder unsichtbar.
  CountDownDialog = Not bStopButtonIsPressed  'Rückgabewert
End Function

REM Führt den Countdown im Dialog durch. Prüft über wiederholte Now-Abfragen
REM den Ablauf von jeweils einer Sekunde, bis die gewünschte Sekundenzahl
REM erreicht ist. Gleichzeitig wird getestet, ob die Stop-Befehlstaste
REM gedrückt wurde.
Sub CountDownWithNow(iLength, oLabel)
  REM iLength - Countdownzeit in Sekunden
  REM oLabel  - Modell des Dialog-Beschriftungsfelds zur Restanzeige

  Dim dOneSec As Date    '1 Sekunde als Datumswert
  Dim dTimecheck As Date 'Vergleichszeitpunkt
  Dim sSecText$          'Benennung im Singular oder Plural
  Dim i%                 'Countdown-Zähler
  sSecText = " Sekunden"
  dOneSec = 1/86400      '1 Tag hat 86400 Sekunden.
  dTimecheck = Now()

  For i = iLength To 1 Step -1
    If i = 1 Then sSecText = " Sekunde"
```

```
    oLabel.Label = "Noch " & i & sSecText
    dTimecheck = dTimecheck + dOneSec
    Do While Now() < dTimecheck And Not bStopButtonIsPressed Loop
    If bStopButtonIsPressed Then Exit For
  Next
End Sub

REM Alternativer Countdown. Misst nicht den Zeitverlauf, sondern unterbricht
REM den Programmablauf durch eine Reihe von jeweils gleich langen
REM Wait-Aufrufen, deren Gesamtdauer die gewünschte Zeitdauer
REM in Sekunden ergibt. Nach jeder Wait-Periode wird getestet,
REM ob die Stop-Befehlstaste gedrückt wurde.
REM Ich habe als Periode eine Zehntelsekunde gewählt, denn je länger sie
REM dauert, desto größer wird die Verzögerung zwischen einem Stop-Befehl
REM und dem Sub-Ende.
Sub CountDownWithWait(iLength, oLabel)
  REM iLength - Countdownzeit in Sekunden
  REM oLabel  - Modell des Dialog-Beschriftungsfelds zur Restanzeige

  Dim iFrequency%         'Anzahl der Stop-Abfragen pro Sekunde
  Dim iMilliSecs%         'Dauer der Unterbrechungen in Millisekunden
  Dim sSecText$           'Benennung im Singular oder Plural
  Dim i%                  'Countdown-Zähler
  sSecText = " Sekunden"
  iFrequency = 10
  iMilliSecs = 1000 \ iFrequency

  For i = iLength * iFrequency To 1 Step -1
    If i Mod iFrequency = 0 Then
      If i = iFrequency Then sSecText = " Sekunde"
      oLabel.Label = "Noch " & i \ iFrequency & sSecText
    End If
    Wait(iMilliSecs)
    If bStopButtonIsPressed Then Exit For
  Next
End Sub
```

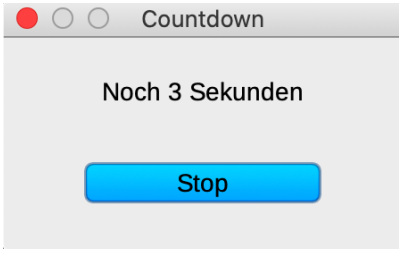


Bild 151. *Ein Countdown als nichtmodaler Dialog.*

18.7.2. Countdown als modaler Dialog mit asynchronem Callback

Aber auch in einem modalen Dialog lässt sich ein Countdown realisieren. Doch da ein solcher Dialog während seiner Anzeige den Rest der Anwendung blockiert, können Dialoginhalte nur über Ereignisse des Dialogs geändert werden und nicht aus dem Programmfluss des laufenden Makros heraus. Um den Zeitfortschritt anzuzeigen, benötigt man einen parallel ablaufenden Prozess.

Im Abschnitt 10.12.5. Asynchroner Callback (A. Heier) wird der Service AsyncCallback und seine Funktionsweise vorgestellt. Mit diesem Hilfsmittel lässt sich ein Countdown sehr einfach und effektiv verwirklichen. Das folgende Makro zählt nach einem wait-Aufruf von jeweils einer Sekunde das

TimeLeftLabel des Dialogs innerhalb eines jeden zum Dialog parallel laufenden Callback-notify-Strangs herunter. Da der Nutzer in dieser Zeit direkt auf den Dialog zugreifen kann, wird keine Aufsplittung in kleinere wait-Abschnitte benötigt. Ein Abbruch wirkt also unmittelbar. Das Aussehen und die Nutzerkommunikation sind identisch mit denen des Dialogs im Listing 571.

Listing 572. *Countdown-Dialog mit asynchronem Callback.*

```
Dim oCBDlg                'Countdown-Dialog
Dim IsStopped As Boolean  'Abbruchsignal

Sub CountDownWithCallback
  Dim iLimit As Integer  'Sekunden bis zum Ende des Countdowns
  IsStopped = False
  iLimit = 10

  oCBDlg = CreateUnoDialog(DialogLibraries.OOME_40.CountdownCBDialog)
  oCBDlg.getControl("TimeLeftLabel").Text = "Noch " & iLimit & " Sekunden"

  CreateCallback(iLimit)

  oCBDlg.execute()
  'Wenn der Dialog über das Abbruch-Kreuzchen beendet wurde,
  'ist das Abbruchsignal IsStopped noch false.
  If Not IsStopped Then
    IsStopped = True
    MsgBox "Dialog abgebrochen"
  End If
End Sub

REM Startet einen Prozeduraufruf als eigenständigen Programmablaufstrang
Sub CreateCallback(iReps As Integer)
  Dim oASync
  Dim oCallback
  oASync = CreateUnoService ("com.sun.star.awt.AsyncCallback")
  oCallback = CreateUnoListener ("CBCountDown_", "com.sun.star.awt.XCallback")
  'Meldet die Prozedur CBCountDown_notify zur Ausführung an, wenn die
  'Laufzeitumgebung dafür bereit ist.
  oASync.addCallback(oCallback, iReps)
End Sub

Sub CBCountDown_notify(iRep As Integer)
  Dim sSecText As String
  sSecText = " Sekunden"
  Wait(1000)
  If Not IsStopped Then
    iRep = iRep - 1
    If iRep = 1 Then sSecText = " Sekunde"
    oCBDlg.getControl("TimeLeftLabel").Text = "Noch " & iRep & sSecText
    If iRep = 0 Then
     Wait(200)  'Damit die 0 als Abschluss kurz sichtbar ist.
      CountDownTerminated
    Else
      'Dies ist kein rekursiver Aufruf, sondern der Start eines neuen
      'parallelen notify-Strangs. Die Sub-Prozedur wird beendet.
      CreateCallback(iRep)
    End If
  End If
End Sub
```

```
REM Der Countdown ist bis zum Ende durchgelaufen.
Sub CountDownTerminated
  oCBDlg.endExecute()
  IsStopped = True
  MsgBox "Countdown beendet"
End Sub

REM Der Countdown wurde über die Stop-Befehlsfläche beendet.
Sub CountDownStopped
  oCBDlg.endExecute()
  IsStopped = True
  MsgBox "Countdown abgebrochen"
End Sub
```

18.8. Das Beispiel Objektinspektor

Dieser Abschnitt behandelt einen Objektinspektor, mit dem ich gerne Objekte inspiziere, um herauszufinden, was ich mit ihnen anstellen kann. Der hier vorgestellte Code hat sich über die Jahre im Hinblick auf spezifische Aufgabenstellungen gewandelt. Sicherlich sind wesentliche Verbesserungen möglich, doch für meine Zwecke reicht es.

Der Objektinspektor erwartet ein zu inspizierendes Objekt als optionales Argument. Wenn es ein UNO-Objekt ist, enthält es Debug-Eigenschaften, die die unterstützten Eigenschaften, Methoden, Services und Interfaces auflisten.

In diesem Beispiel wird ein Dialog ohne die Basic-IDE erzeugt. Die Information aus der Inspektion wird in einem mehrzeiligen Texteingabefeld ausgegeben. Das Textfeld hat einen Vorteil gegenüber dem Listenfeld: Man kann daraus Text in die Zwischenablage kopieren. Beim Listenfeld geht das nicht. Es kommt häufig vor, dass ich ein Objekt inspiziere und dann die enthaltenen Eigenschaften, Methoden und Interfaces zur späteren Verwendung als Referenz in die Zwischenablage kopiere.

Optionsfelder steuern den Inhalt des Textfelds. Man kann wechseln zwischen der Ausgabe von Eigenschaften, Methoden, Interfaces und Services sowie allgemeinen Inspektionsinformationen wie dem Datentyp des Objekts (s. Bild 152).

18.8.1. Dienstfunktionen und -Subroutinen

Man braucht eine Reihe von nützlichen Helferlein zur Inspektion eines beliebigen Objekts in OOo. Diese Dienstroutinen haben keinerlei Zugriff auf den Dialog oder seine Kontrollelemente, so dass ich sie von der Hauptfunktionalität getrennt habe. Die hier genutzten Techniken und Methoden sollten Ihnen schon bekannt vorkommen, daher habe ich sie nicht im Einzelnen kommentiert. All diese Routinen finden Sie im Modul ObjInspector.

Leerraum in einem String erkennen und entfernen

Ein Textzeichen, das als freie Fläche erscheint, wird Leerraum, auch weißes Zeichen genannt. In diesem Buch werden beispielsweise die einzelnen Wörter durch ein einzelnes Leerzeichen getrennt. Das Zeilenendezeichen kann Leerraum zwischen den Zeilen produzieren. Bei der Inspektion eines Objekts enthalten die Informationsstrings häufig am Anfang oder am Ende weiße Zeichen unterschiedlicher Art. Die Funktionen des Listing 573 ermitteln die verschieden Leerraumzeichen und entfernen sie aus dem String.

Listing 573. *Leerraum in einem String erkennen und entfernen.*

```
REM Dienstfunktionen und -Methoden
```

```
'************************************************************************
'** Ist das betreffende Zeichen Leerraum? Die Antwort lautet ja, wenn
'** das Zeichen ein Tab, CR, LF, einfaches oder geschütztes Leerzeichen ist.
'** Es handelt sich um die ASCII-Werte 9, 10, 13, 32 und 160.
'************************************************************************
Function IsWhiteSpace(iChar As Integer) As Boolean
  Select Case iChar
    Case 9, 10, 13, 32, 160
      IsWhiteSpace = True
    Case Else
      IsWhiteSpace = False
    End Select
End Function
'************************************************************************
'** Sucht das erste Zeichen von der Position i% an, das kein Leerraum ist.
'** Wenn es keines gibt, ist der Rückgabewert größer als die Stringlänge.
'************************************************************************
Function FirstNonWhiteSpace(ByVal i%, s$) As Integer
  If i <= Len(s) Then
    Do While IsWhiteSpace(Asc(Mid$(s, i, 1)))
      i = i + 1
      If i > Len(s) Then
        Exit Do
      End If
    Loop
  End If
  FirstNonWhiteSpace = i
End Function
'************************************************************************
'** Entfernt Leerraum am Anfang und Ende eines Strings.
'** Der übergebene String wird verändert und zurückgegeben.
'** Entfernt wird jede Art von Leerraum, nicht nur einfache Leerzeichen.
'************************************************************************
Function TrimWhite(s As String) As String
  s = Trim(s)
  Do While Len(s) > 0
    If Not IsWhiteSpace(Asc(s)) Then Exit Do
    s = Right(s, Len(s) - 1)
  Loop
  Do While Len(s) > 0
    If Not IsWhiteSpace(Asc(Right(s, 1))) Then Exit Do
    s = Left(s, Len(s) - 1)
  Loop
  TrimWhite = s
End Function
```

Einfache Objekte in einen String konvertieren

Die Methode CStr() konvertiert die meisten Standarddatentypen in einen String. Doch nicht alle Variablen lassen sich leicht mit CStr() in einen String verwandeln, daher versucht es die Funktion ObjToString() (s. Listing 574) mit einer intelligenten Konvertierung. Die Variablen vom Typ Objekt und Variant bereiten die größten Probleme, denn sie können die Werte Empty oder Null haben – diese beiden Fälle werden ermittelt und entsprechend beschrieben. Variablen vom Typ Objekt können nicht so einfach in einen String umgewandelt werden, also geschieht es auch nicht. Die Standardtypen jedoch werden konvertiert. Sehr lange Strings werden nach 100 Zeichen abgeschnitten.

Listing 574. *Ausgabe eines Objekts als String.*

```
Function ObjToString(oInsObj, Optional arraySep$, _
            Optional maxDepth%, Optional CRReplace As Boolean) As String

  Dim iMaxDepth% ' Maximalzahl der darzustellenden Arrayelemente.
  Dim sArraySep$ ' Trennzeichen zwischen Arrayelementen.
  Dim s$         ' Ergebnisstring.
  Dim bCRReplace As Boolean

  ' Optionale Argumente
  If IsMissing(CRReplace) Then
    bCRReplace = False
  Else
    bCRReplace = CRReplace
  End If

  If IsArray(oInsObj) Then
    If IsMissing(arraySep) Then
      sArraySep = Chr$(10)
    Else
      sArraySep = arraySep
    End If

    If IsMissing(maxDepth) THen
      iMaxDepth = 100
    Else
      iMaxDepth = maxDepth
    End If

    s = ObjArrayToString(oInsObj, sArraySep, iMaxDepth)
  Else
    ' Kein Array.
    Select Case VarType(oInsObj)
      Case 0
        s = "Leer"
      Case 1
        s = "Null"
      Case 2, 3, 4, 5, 7, 8, 11, 16 To 23, 33 To 37
        s = CStr(oInsObj)
      Case Else
        If IsUnoStruct(oInsObj) Then
          s = "[Ein UnoStruct kann nicht als String dargestellt werden.]"
        Else
          Dim sImplName$ : sImplName = GetImplementationName(oInsObj)
          If sImplName = "" Then
              s = "[" & TypeName(oInsObj) & _
                " kann nicht als String dargestellt werden.]"
          Else
            s = "[" & TypeName(oInsObj) & " (" & sImplName & _
                ") kann nicht als String dargestellt werden.]"
          End If
        End If
    End Select
    If bCRReplace Then
      s = Replace(s, Chr$(10), "|", 1, -1, 1)
```

```
      s = Replace(s, Chr$(13), "|", 1, -1, 1)
    End If
    If Len(s) > 100 Then s = Left(s, 100) & "...."
  End If
  ObjToString = s
End Function
```

Arrays (ein- oder zweidimensional) werden erkannt und separat konvertiert.

Listing 575. *Konvertiert ein Array in einen String.*

```
'************************************************************************
'** Konvertiert ein Array von Objekten in einen String, falls möglich.
'**
'** oInsObj    - Zu inspizierendes Objekt.
'** sArraySep - Trennzeichen zwischen Arrayelementen.
'** maxDepth   - Maximalzahl der darzustellenden Arrayelemente.
'************************************************************************
Function ObjArrayToString(oInsObj, sArraySep$, Optional maxDepth%) As String
  Dim iMaxDepth%
  Dim iDimensions%

  If IsMissing(maxDepth) Then
    iMaxDepth = 100
  Else
    iMaxDepth = maxDepth
  End If

  iDimensions = NumArrayDimensions(oInsObj)
  If iDimensions = 1 Then
    ObjArrayToString = ObjArray_1_ToString(oInsObj, iMaxDepth%, sArraySep)
  ElseIf iDimensions = 2 Then
    ObjArrayToString = ObjArray_2_ToString(oInsObj, iMaxDepth%, sArraySep)
  Else
    ObjArrayToString = "[Die Konvertierung eines " & iDimensions & _
                       "-dimensionalen Arrays ist nicht möglich.]"
  End If
End Function

'************************************************************************
'** Konvertiert ein eindimensionales Array von Objekten in einen String.
'**
'** oInsObj     - Zu inspizierendes Objekt.
'** iMaxDepth   - Maximalzahl der darzustellenden Arrayelemente.
'** sSep        - Trennzeichen zwischen Arrayelementen.
'************************************************************************
Function ObjArray_1_ToString(oInsObj, iMaxDepth%, sSep$) As String
  Dim iDim_1%
  Dim iMax_1%
  Dim iCounter%
  Dim s$
  Dim sTempSep$

  sTempSep = ""
  iDim_1 = LBound(oInsObj, 1)
  iMax_1 = UBound(oInsObj, 1)
  Do While (iCounter < iMaxDepth And iDim_1 <= iMax_1)
    iCounter = iCounter + 1
    s = s & sTempSep & "[" & iDim_1 & "] " _
          & ObjToString(oInsObj(iDim_1), ", ")
```

```
    sTempSep = sSep
    iDim_1 = iDim_1 + 1
  Loop
  ObjArray_1_ToString = s
End Function

'************************************************************************
'** Konvertiert ein zweidimensionales Array von Objekten in einen String.
'**
'** oInsObj    - Zu inspizierendes Objekt.
'** iMaxDepth  - Maximalzahl der darzustellenden Arrayelemente.
'** sSep       - Trennzeichen zwischen Arrayelementen.
'************************************************************************
Function ObjArray_2_ToString(oInsObj, iMaxDepth%, sSep$) As String
  Dim iDim_1%, iDim_2%
  Dim iMax_1%, iMax_2%
  Dim iCounter%
  Dim s$
  Dim sTempSep$

  sTempSep = ""
  iDim_1 = LBound(oInsObj, 1)
  iMax_1 = UBound(oInsObj, 1)
  iMax_2 = UBound(oInsObj, 2)
  Do While (iCounter < iMaxDepth And iDim_1 <= iMax_1)
    iCounter = iCounter + 1
    iDim_2 = LBound(oInsObj, 2)
    Do While (iCounter < iMaxDepth And iDim_2 <= iMax_2)
      s = s & sTempSep & "[" & iDim_1 & ", " & iDim_2 & "] " _
            & ObjToString(oInsObj(iDim_1, iDim_2), ", ")
      sTempSep = sSep
      iDim_2 = iDim_2 + 1
    Loop
    iDim_1 = iDim_1 + 1
  Loop
  ObjArray_2_ToString = s
End Function

'************************************************************************
'** Anzahl der Dimensionen eines Arrays.
'** A(4) gibt 1 zurück, A(3, 4) gibt 2 zurück.
'**
'** oInsObj    - Zu inspizierendes Objekt.
'************************************************************************
Function NumArrayDimensions(oInsObj) As Integer
  Dim i%  : i = 0
  If IsArray(oInsObj) Then
    On Local Error Goto DebugBoundsError:
    Do While (i% >= 0)
      LBound(oInsObj, i% + 1)
      UBound(oInsObj, i% + 1)
      i% = i% + 1
    Loop
    DebugBoundsError:
    On Local Error Goto 0
  End If
```

```
    NumArrayDimensions = i
  End Function
```

Es ist ganz nützlich, den internen Namen des Objekts zu kennen. Es muss aber beachtet werden, dass es Objekte gibt, die diese Methode nicht unterstützen.

Listing 576. *Ermittlung des internen Objektnamens (falls es ein UNO-Service ist).*

```
'************************************************************************
'** Ermittlung des internen Namens eines Objekts, falls existent.
'**
'** oInsObj    - Zu inspizierendes Objekt.
'************************************************************************
Function GetImplementationName(oInsObj) As String
  On Local Error GoTo DebugNoSet
  Dim oTmpObj
  oTmpObj = oInsObj
  GetImplementationName = oTmpObj.getImplementationName()
  DebugNoSet:
End Function
```

Objektinspektion mit Basic-Methoden

Die Funktion ObjInfoString() (s. Listing 577) erzeugt einen String, der ein Objekt beschreibt, mit detaillierten Informationen über den Typ des Objekts. Wenn das Objekt ein Array ist, werden dessen Dimensionen angegeben. Falls möglich wird auch der UNO-Implementationsname ermittelt. Weitere UNO-bezogene Informationen – wie zum Beispiel Eigenschaften oder Methoden – werden jedoch nicht ermittelt.

Listing 577. *Ermittlung von Informationen zu einem Objekt.*

```
'************************************************************************
'** Gibt einen String zurück mit nützlichen Informationen zu einem Objekt.
'** Enthalten sind detaillierte Informationen über den Typ des Objekts.
'** Wenn das Objekt ein Array ist, werden dessen Dimensionen aufgeführt.
'** Falls möglich wird auch der UNO-Implementationsname ermittelt.
'************************************************************************
Function ObjInfoString(oInsObj) As String
  Dim s As String

  REM Typname und Variablentyp können immer ermittelt werden.
  s = "TypeName = " & TypeName(oInsObj) & Chr$(10) &_
    "VarType = " & VarType(oInsObj) & Chr$(10)

  REM Die Werte NULL und EMPTY testen.
  If IsNull(oInsObj) Then
    s = s & "IsNull = True"
  ElseIf IsEmpty(oInsObj) Then
    s = s & "IsEmpty = True"
  Else
    If IsObject(oInsObj) Then
      s = s & "Implementationsname = " & GetImplementationName(oInsObj)
      s = s & Chr$(10) & "IsObject = True" & Chr$(10)
    End If
    If IsUnoStruct(oInsObj) Then s = s & "IsUnoStruct = True" & Chr$(10)
    If IsDate(oInsObj) Then s = s & "IsDate = True" & Chr$(10)
    If IsNumeric(oInsObj) Then s = s & "IsNumeric = True" & Chr$(10)
    If IsArray(oInsObj) Then
      On Local Error Goto DebugBoundsError:
      Dim i%, sTemp$
```

```
        s = s & "IsArray = True" & Chr$(10) & "Bereich = ("
        Do While (i% >= 0)
          i% = i% + 1
          sTemp$ = LBound(oInsObj, i%) & " To " & UBound(oInsObj, i%)
          If i% > 1 Then s = s & ", "
          s = s & sTemp$
        Loop
        DebugBoundsError:
        On Local Error Goto 0
        s = s & ")" & Chr$(10)
      End If
    End If

    s = s & "Wert : " & Chr$(10) & ObjToString(oInsObj) & Chr$(10)
    ObjInfoString = s
  End Function
```

Sortierung eines Arrays

Viele Objekte enthalten so viele Eigenschaften und setzen so viele Methoden ein, dass es schwierig wird, einen bestimmten Eintrag zu finden, wenn sie im Textfeld ausgegeben werden. Es wird einfacher, wenn die Einträge sortiert ausgegeben werden.

Angenommen, ich habe zwei Arrays. Das erste enthält eine Liste von Namen und das zweite eine Liste der zugehörigen Altersangaben. Das dritte Element im Altersarray bezieht sich also auf das dritte Element des Namenarrays. In der Informatik nennt man so etwas „parallele Arrays“. Im Objektbrowser kommen solche parallelen Arrays vor. Bei den Eigenschaften ist es so, dass der Typ der Eigenschaft im ersten Array gespeichert wird und der Name der Eigenschaft im zweiten Array. Bei der Auflistung der unterstützten Methoden ist es genauso.

Wenn die Daten für die Inspektion entstehen, werden parallele Arrays erzeugt und verwaltet. Ich will die Information sortieren, kann aber nicht ein Array ohne Berücksichtigung der anderen Arrays sortieren. Die übliche Lösung für dieses Problem besteht darin, ein Integer-Array der Indexwerte zu erstellen, die dann zur Indexierung der Arrays genutzt werden. Anstatt der echten Daten wird einfach bei der Sortierung das Indexarray neu arrangiert. Als Beispiel soll das Array oItems()=("B", "C", "A") sortiert werden. Das Indexarray (0, 1, 2) wird nach der Sortierung zu (2, 0, 1): in dessen Abfolge steht oItems(2) also am Anfang, dann folgt oItems(0) und schließlich oItems(1), also „A B C“. Die Funktion SortMyArray() im Listing 578 sortiert ein Array über ein Indexarray.

Listing 578. *Indirekte Sortierung eines Arrays durch Indexverschiebung.*

```
'************************************************************************
'** Sortiert das Array oItems() durch Neuordnung des Arrays iIdx().
'** Es sei oItems ="B", "C", "A"), die Ausgabe sei iIdx() = (2, 0, 1).
'** Das heißt, die Reihenfolge der Einträge soll so sein:
'** erst oItems(2), dann oItems(0), zuletzt oItems(1).
'************************************************************************
Sub SortMyArray(oItems(), iIdx() As Integer)
  Dim i As Integer        'Äußere Indexvariable.
  Dim j As Integer        'Innere Indexvariable.
  Dim temp As Integer     'Temporäre Variable zum Vertauschen zweier Werte.
  Dim bChanged As Boolean 'Wird True, wenn sich etwas ändert.
  For i = LBound(oItems()) To UBound(oItems()) - 1
    bChanged = False
    For j = UBound(oItems()) To i + 1 Step -1
      If oItems(iIdx(j)) < oItems(iIdx(j - 1)) Then
        temp = iIdx(j)
```

```
            iIdx(j) = iIdx(j - 1)
            iIdx(j - 1) = temp
            bChanged = True
          End If
        Next
        If Not bChanged Then Exit For
      Next
    End Sub
```

18.8.2. Einen Dialog zur Laufzeit erzeugen

Der Objektinspektor definiert Variablen für den Dialog und häufig referenzierte Objekte als Private. Der Dialog enthält einen Fortschrittsbalken, der aktualisiert wird, während Informationen in das Textfeld einfließen. Auch das inspizierte Objekt wird als globale Variable geführt, denn es wird von Ereignisbeobachtern überwacht, die ansonsten keinen Zugang zu dem Objekt hätten (s. Listing 579).

Listing 579. *Globale Variablen im Modul ObjInspector.*

```
Private oDlg           'Der dargestellte Dialog.
Private oProgress      'Das Modell des Fortschrittsbalkens.
Private oTextEdit      'Das Modell des Textfelds, das die Informationen ausgibt.
Private oObjects(100)  'Die zu inspizierenden Objekte.
Private Titles(100)    'Titel.
Private iDepth%        'Aktuell genutztes Objekt/Titel.
```

Leider können Dialoge, die in der IDE entwickelt wurden, nur innerhalb von Basic erzeugt und aufgerufen werden. Man kann jedoch in jeder Programmiersprache, die auf die UNO-API zugreifen kann, Dialoge zur Laufzeit erzeugen und benutzen. Die Dialogfunktionalität sollte seit der Version OOo 2.0 so weit verbessert sein, dass man Dialoge in der IDE entwickeln und dann mit anderen Sprachen als Basic verwenden kann. Die Subroutine Inspect() im Listing 580 erzeugt einen Dialog mit all seinen Kontrollelementen, und zwar ohne die IDE. Folgende Schritte sind dazu erforderlich:

1. Erzeugen Sie mit CreateUnoService("com.sun.star.awt.UnoControlDialogModel") das Modell eines Dialogs und legen Sie dann die Eigenschaften des Modells fest.
2. Erzeugen Sie mit der Methode createInstance(name) des Dialogmodells ein Modell für jedes im Dialog eingesetzte Kontrollelement. Legen Sie die Eigenschaften des jeweiligen Modells fest und fügen Sie es mit der Dialogmodell-Methode insertByName(name, oModel) in das Dialogmodell ein. Dieser Schritt ist wichtig! Um ein Kontrollelement in einen Dialog einzufügen, müssen Sie mit dem Dialogmodell ein Kontrollmodell erzeugen, das dann in das Dialogmodell eingefügt wird. Es sind keine Kontrollelemente, die Sie erzeugen oder steuern, sondern nur Modelle.
3. Erzeugen Sie einen Dialog mit CreateUnoService("com.sun.star.awt.UnoControlDialog").
4. Weisen Sie dem Dialog mit dessen Methode setModel() das Dialogmodell zu.
5. Erzeugen Sie mit CreateUnoListener(name) die erforderlichen Beobachter. Verknüpfen Sie diese Beobachter mit den richtigen Objekten, normalerweise mit Kontrollelementen.
6. Erzeugen Sie mit CreateUnoService("com.sun.star.awt.Toolkit") ein Fensterwerkzeug-Objekt.
7. Weisen Sie den Dialog mit seiner Methode createPeer(oWindow, null) an, ein entsprechendes Benutzungsfenster zu erzeugen.
8. Führen Sie den Dialog aus.

Listing 580. *Erzeugt den Dialog, die Kontrollelemente und die Beobachter.*

```
Sub Inspect(Optional oInsObj, Optional title$)
  Dim oDlgModel            'Das Modell des Dialogs.
  Dim oModel               'Das Modell eines Kontrollelements.
```

```
Dim oListener            'Ein neu erzeugtes Beobachterobjekt.
Dim oControl             'Referenz auf ein Kontrollelement.
Dim iTabIndex As Integer 'Der aktuelle Tab-Index beim Erzeugen der Kontrollelemente.
Dim iDlgHeight As Long   'Die Höhe des Dialogs.
Dim iDlgWidth As Long    'Die Breite des Dialogs.
Dim sTitle$

iDepth = LBound(oObjects)
sTitle = ""
If Not IsMissing(title) Then sTitle = title

REM Wenn kein Objekt übergeben wurde, wird ThisComponent genommen.
If IsMissing(oInsObj) Then
  oObjects(iDepth) = ThisComponent
  If IsMissing(title) Then sTitle = "ThisComponent"
Else
  oObjects(iDepth) = oInsObj
  If IsMissing(title) Then sTitle = "Obj"
End If
Titles(iDepth) = sTitle

iDlgHeight = 300
iDlgWidth  = 350

REM Erzeugt das Modell des Dialogs.
oDlgModel = CreateUnoService("com.sun.star.awt.UnoControlDialogModel")
SetProperties(oDlgModel, Array("PositionX", 50, "PositionY", 50,_
   "Width", iDlgWidth, "Height", iDlgHeight, "Title", sTitle))

CreateInsertControl(oDlgModel, iTabIndex, "PropButton",_
   "com.sun.star.awt.UnoControlRadioButtonModel",_
   Array("PositionX", 10, "PositionY", 10, "Width", 55, "Height", 15,_
     "Label", "Eigenschaften"))

CreateInsertControl(oDlgModel, iTabIndex, "MethodButton",_
   "com.sun.star.awt.UnoControlRadioButtonModel",_
   Array("PositionX", 75, "PositionY", 10, "Width", 55, "Height", 15,_
     "Label", "Methoden"))

CreateInsertControl(oDlgModel, iTabIndex, "ServiceButton",_
   "com.sun.star.awt.UnoControlRadioButtonModel",_
   Array("PositionX", 140, "PositionY", 10, "Width", 55, "Height", 15,_
     "Label", "Services"))

CreateInsertControl(oDlgModel, iTabIndex, "ObjectButton",_
   "com.sun.star.awt.UnoControlRadioButtonModel",_
   Array("PositionX", 205, "PositionY", 10, "Width", 55, "Height", 15,_
     "Label", "Objekt"))

CreateInsertControl(oDlgModel, iTabIndex, "EditControl",_
   "com.sun.star.awt.UnoControlEditModel",_
   Array("PositionX", 10, "PositionY", 25, "Width", iDlgWidth - 20,_
     "Height", iDlgHeight - 75, "HScroll", True, "VScroll", True,_
     "MultiLine", True, "HardLineBreaks", True))

REM Speichert das Modell des Textfelds in einer globalen Variablen.
```

```
oTextEdit = oDlgModel.getByName("EditControl")

CreateInsertControl(oDlgModel, iTabIndex, "Progress",_
   "com.sun.star.awt.UnoControlProgressBarModel",_
   Array("PositionX", 10, "PositionY", iDlgHeight - 45,_
     "Width", iDlgWidth - 20, "Height", 15, "ProgressValueMin", 0,_
     "ProgressValueMax", 100))

REM Speichert eine Referenz auf den Fortschrittsbalken.
oProgress = oDlgModel.getByName("Progress")

REM Beachten Sie, dass ich den Typ auf OK setze, so dass ich zum Schließen
REM des Dialogs keinen Ereignisbeobachter benötige.
CreateInsertControl(oDlgModel, iTabIndex, "OKButton",_
   "com.sun.star.awt.UnoControlButtonModel",_
   Array("PositionX", Clng(25), "PositionY", iDlgHeight - 20,_
     "Width", 65, "Height", 15, "Label", "Schließen",_
     "PushButtonType", com.sun.star.awt.PushButtonType.OK))

CreateInsertControl(oDlgModel, iTabIndex, "InspectButton",_
   "com.sun.star.awt.UnoControlButtonModel",_
   Array("PositionX", Clng(iDlgWidth / 2 - 50), "PositionY", iDlgHeight - 20,_
     "Width", 65, "Height", 15, "Label", "Inspektion Auswahl",_
     "PushButtonType", com.sun.star.awt.PushButtonType.STANDARD))

CreateInsertControl(oDlgModel, iTabIndex, "BackButton",_
   "com.sun.star.awt.UnoControlButtonModel",_
   Array("PositionX", Clng(iDlgWidth / 2 + 50), "PositionY", iDlgHeight - 20,_
     "Width", 65, "Height", 15, "Label", "Inspektion zurück",_
     "Enabled", False, _
     "PushButtonType", com.sun.star.awt.PushButtonType.STANDARD))

REM Erzeugung des Dialogs und Zuweisung des Modells.
oDlg = CreateUnoService("com.sun.star.awt.UnoControlDialog")
oDlg.setModel(oDlgModel)

REM Der Beobachter für alle Optionsfelder.
oListener = CreateUnoListener("radio_", "com.sun.star.awt.XItemListener")
oControl = oDlg.getControl("PropButton")
oControl.addItemListener(oListener)
oControl = oDlg.getControl("MethodButton")
oControl.addItemListener(oListener)
oControl = oDlg.getControl("ServiceButton")
oControl.addItemListener(oListener)
oControl = oDlg.getControl("ObjectButton")
oControl.addItemListener(oListener)
oControl.getModel().State = 1

oListener = CreateUnoListener("ins_", "com.sun.star.awt.XActionListener")
oControl = oDlg.getControl("InspectButton")
oControl.addActionListener(oListener)

oListener = CreateUnoListener("back_", "com.sun.star.awt.XActionListener")
oControl = oDlg.getControl("BackButton")
oControl.addActionListener(oListener)

REM Nun wird der Dialog mit den Standardeinstellungen versorgt.
DisplayNewObject()
```

```
  REM Erzeugt ein Fenster und gibt es dem Dialog zur Benutzung.
  Dim oWindow
  oWindow = CreateUnoService("com.sun.star.awt.Toolkit")
  oDlg.createPeer(oWindow, null)

  REM Schließlich wird der Dialog aufgerufen.
  oDlg.execute()
End Sub
```

Um den Code im Listing 580 kurz zu halten, habe ich zwei Dienstprozeduren eingesetzt zum Setzen der Eigenschaften, zur Erzeugung des Modells eines Kontrollelements und zum Einfügen des Kontrollmodells in das Dialogmodell. Beachten Sie, dass das Kontrollelement selbst niemals erzeugt wird – nur sein Modell. Listing 581 zeigt die Methoden zum Erzeugen des Kontrollmodells und zum Setzen der Eigenschaften. Die Eigenschaften werden mit der Methode setPropertyValue gesetzt, Sie können sie aber auch in Basic direkt zuweisen, wenn Sie wollen.

Listing 581. *Erzeugt das Modell eines Kontrollelements und fügt es in das Dialogmodell ein.*

```
'*************************************************************************
'** Erzeugt ein Kontrollelement des Typs sType mit dem Namen sName
'** und fügt es in das Dialogmodell oDlgModel ein.
'**
'** oDlgModel - Modell des Dialogs für das Kontrollelement.
'** index     - Tabindex, der hochgezählt wird.
'** sName     - Name des Kontrollelements.
'** sType     - Der komplette Servicename des Kontrollmodells.
'** props     - Array der Name-Werte-Paare der Eigenschaft.
'*************************************************************************
Sub CreateInsertControl(oDlgModel, index%, sName$, sType$, props())
  Dim oModel

  oModel = oDlgModel.createInstance(sType$)
  SetProperties(oModel, props())
  SetProperties(oModel, Array("Name", sName$, "TabIndex", index%))
  oDlgModel.insertByName(sName$, oModel)

  REM Ändert den Wert außerhalb,
  REM weil die Übergabe nicht mit dem Wert (ByValue) erfolgt.
  index% = index% + 1
End Sub
```

Das folgende Makro setzt viele Eigenschaften in einem einzigen Aufruf.

Listing 582. *Setzt viele Eigenschaften in einem Schwung.*

```
REM Setzt Eigenschaften generisch mit Hilfe eines Arrays von Name-Wert-Paaren.
Sub SetProperties(oModel, props())
  Dim i As Integer
  For i = LBound(props()) To UBound(props()) Step 2
    oModel.setPropertyValue(props(i), props(i + 1))
  Next
End Sub
```

18.8.3. Beobachter

Optionsfelder

Jedes Mal, wenn ein anderes Optionsfeld ausgewählt wird, startet die Prozedur radio_itemStateChanged() (der Beobachter wurde im Listing 580 zugeordnet). Die Ereignisprozedur ruft die Subroutine DisplayNewObject() auf, die ihrerseits die für die Anforderung eines neuen Eigenschaftstyps notwendige Arbeit erledigt (s. Listing 583). Ich habe mich für eine getrennte Prozedur entschieden, damit sie unabhängig von der Ereignisbehandlung genutzt werden kann. Zum Beispiel rufe ich DisplayNewObject() auf, bevor der Dialog angezeigt wird, so dass der Dialog vor dem ersten Erscheinen die wesentlichen Informationen enthält.

Listing 583. *Die Ereignisprozedur ist sehr einfach: sie ruft DisplayNewObject() auf.*

```
REM Diese Prozedur unterstützt das Interface com.sun.star.awt.XItemListener
REM für den Optionsfeldbeobachter. Sie wird aufgerufen, wenn ein Optionsfeld
REM ausgewählt wird.
Sub radio_itemStateChanged(oItemEvent)
  DisplayNewObject()
End Sub

Sub DisplayNewObject()
  REM Setzt den Fortschrittsbalken zurück!
  oProgress.ProgressValue = 0
  oTextEdit.Text = ""

  On Local Error GoTo IgnoreError:
  If IsNull(oObjects(iDepth)) Or IsEmpty(oObjects(iDepth)) Then
    oTextEdit.Text = ObjInfoString(oObjects(iDepth))
  ElseIf oDlg.getModel().getByName("PropButton").State = 1 Then
    ProcessStateChange(oObjects(iDepth), "p")
  ElseIf oDlg.getModel().getByName("MethodButton").State = 1 Then
    ProcessStateChange(oObjects(iDepth), "m")
  ElseIf oDlg.getModel().getByName("ServiceButton").State = 1 Then
    ProcessStateChange(oObjects(iDepth), "s")
  Else
    oTextEdit.Text = ObjInfoString(oObjects(iDepth))
  End If
  oProgress.ProgressValue = 100

  IgnoreError:
End Sub
```

Inspektion Auswahl

Die Ereignisbehandlung für die Schaltfläche „Inspektion Auswahl“ geht vom aktuell markierten Text aus und versucht, auf dieser Grundlage ein neues Objekt zu inspizieren. Ich habe erwogen, den Ablauf benutzerfreundlicher zu gestalten, aber es genügt mir so, wie es ist. Wenn Sie einen Text markieren, wird eine Eigenschaft dieses Namens inspiziert. Wenn Sie eine Zahl markieren, wird angenommen, dass das aktuelle Objekt ein Array ist, und es wird das Objekt an dem entsprechenden Index inspiziert.

Listing 584. *Inspektion eines neuen Elements.*

```
Sub Ins_actionPerformed(oActionEvent)
  Dim oControl
  Dim oSel
  Dim sText$
  Dim v
  Dim oOldObj
```

```
  Dim sOldTitle$
  Dim sNewTitle$

  If iDepth = UBound(oObjects) Then
    Print "Leider schon zu tief verschachtelt."
    Exit Sub
  End If

  oControl = oDlg.getControl("EditControl")
  sText = oControl.getSelectedText()
  sOldTitle = oDlg.Title
  sNewTitle = sOldTitle & "/" & sText

  If IsNumeric(sText) Then
    If NumArrayDimensions(oObjects(iDepth)) <> 1 Then
      Print "Eine Zahl kann nur für ein eindimensionales Array gewählt werden."
    Else
      oOldObj = oObjects(iDepth)
      iDepth = iDepth + 1
      oDlg.getControl("BackButton").getModel().Enabled = True
      oObjects(iDepth) = oOldObj(sText)
      oDlg.Title = sNewTitle
      Titles(iDepth) = sNewTitle
      DisplayNewObject()
    End If
  Else
    v = RunFromLib(GlobalScope.BasicLibraries, _
                   "xyzzylib", "TestMod", oObjects(iDepth), sText, False, False)
    If IsNull(v) Or IsEmpty(v) Then
      '
    Else
      iDepth = iDepth + 1
      oDlg.getControl("BackButton").getModel().Enabled = True
      oObjects(iDepth) = v
      oDlg.Title = sNewTitle
      Titles(iDepth) = sNewTitle
      DisplayNewObject()
    End If
  End If
End Sub
```

Inspektion zurück

Bereits inspizierte Objekte werden in einem Stapel verwaltet. Das „zurück“-Element ist das letzte Element auf dem Stapel.

Listing 585. *Inspektion eines vorherigen Elements.*

```
Sub back_actionPerformed(oActionEvent)
  If iDepth <= LBound(oObjects) Then
    Exit Sub
  End If
  iDepth = iDepth - 1
  If iDepth = LBound(oObjects) Then
     oDlg.getControl("BackButton").getModel().Enabled = False
  End If
```

```
    oDlg.Title = Titles(iDepth)
    DisplayNewObject()
End Sub
```

18.8.4. Die Debug-Information ermitteln

Wenn im Listing 583 Eigenschaften, Methoden oder Services gewünscht sind, wird die Subroutine ProcessStateChange() aufgerufen (s. Listing 586). Andernfalls wird dem Textfeld eine einfache Objektinformation übergeben.

Auch hier in der Subroutine ProcessStateChange() ist wenig Programmlogik zu finden. Der Hauptzweck des Listing 586 ist, die Liste der unterstützten Services dem Inhalt des Textfelds im Anschluss an die aus der Eigenschaft dbg_supportedInterfaces ermittelten Interfaces anzuhängen.

Listing 586. *Baut den Informationstext über das Objekt auf.*

```
'************************************************************************
'** Baut den Text auf. Für einen Service werden die Interfaces und Services
'** in getrennten Gruppen angezeigt, so dass man sie leichter finden kann.
'************************************************************************
Sub ProcessStateChange(oInsObj, sPropType$)
  Dim oItems()
  BuildItemArray(oInsObj, sPropType$, oItems())
  If sPropType$ = "s" Then
    Dim s As String
    On Local Error Resume Next
    s = "** INTERFACES **" & Chr$(10) & Join(oItems, Chr$(10))
    s = s & Chr$(10) & Chr$(10) & "** SERVICES **" & Chr$(10) & _
      Join(oInsObj.getSupportedServiceNames(), Chr$(10))
    oTextEdit.Text = s
  Else
    oTextEdit.Text = Join(oItems, Chr$(10))
  End If
End Sub
```

Der größte Teil des für diesen Dialog wichtigen Codes steckt in der Subroutine BuildItemArray(). Zuallererst werden die „dbg_“-Eigenschaften abgerufen, jeweils ein einziger String mit der Liste der Eigenschaften, der Methoden und der Interfaces. Dann wird der String in seine einzelnen Elemente aufgetrennt, die dann sortiert werden. Zur Ausgabe von Eigenschaften wird zusätzlich der Wert jeder unterstützten Eigenschaft über das Interface XPropertySetInfo abgefragt. So kann man neben dem Namen einer Eigenschaft auch ihren Wert sehen. Das PropertySetInfo-Objekt unterstützt die meisten der von der Eigenschaft dbg_properties zurückgegebenen Eigenschaften, aber eben nicht alle. Die Zurückgabe der einzelnen Daten erfolgt als Stringarray im übergebenen Array oItems(). Das Bild 152 zeigt ein Beispiel.

Listing 587. *Baut ein Array von Eigenschaften, Methoden oder Services auf.*

```
'************************************************************************
'** Diese Routine zergliedert die Strings, die von dbg_Methods,
'** dbg_properties und dbg_supportedInterfaces kommen.
'** Die interessanten Daten stehen nach dem ersten Doppelpunkt.
'**
'** Daher wird alles bis zum ersten Doppelpunkt weggeworfen.
'** Der Rest des Strings wird jeweils am Trennstring, der die Einträge
'** voneinander trennt, in Einzelelemente aufgeteilt.
'**
'** Der Teilstring "Sbx" wird überall entfernt, damit die Ausgabe
'** besser lesbar ist. Falls dieser String in einem Methodennamen
'** tatsächlich vorkommen sollte, wird er dennoch entfernt. Das mag zwar
'** nicht die sicherste Methode sein, aber mir ist dieser Fall noch nie
```

```
'** vorgekommen.
'**
'** oItems() enthält am Ende alle einzelnen Teilelemente.
'*************************************************************************
Sub BuildItemArray(oInsObj, sType$, oItems())
  On Error Goto BadErrorHere
  Dim s As String           'Die zu zergliedernde Ursprungsliste.
  Dim sSep As String        'Der jeweils in den Strings verwendete Trenner.
  Dim iCount%               '
  Dim iPos%                 '
  Dim sNew$                 '
  Dim i%                    '
  Dim j%                    '
  Dim sFront() As String    'Der vordere Teil eines kompletten Elements.
  Dim sMid() As String      'Der mittlere Teil eines kompletten Elements.
  Dim iIdx() As Integer     'Hilfsarray zum Sortieren paralleler Arrays.
  Dim nFrontMax As Integer  'Größte vorkommende Länge eines Vorderteils.
  Dim sArraySep$            : sArraySep = ", "
  Dim iArrayMaxDepth%       : iArrayMaxDepth = 10

  nFrontMax = 0

  REM Erste Frage: was soll inspiziert werden?
  s = ""
  On Local Error Resume Next
  If sType$ = "s" Then
    REM dbg_supportedInterfaces gibt Interfaces zurück und
    REM getSupportedServiceNames() gibt Services zurück.
    s = oInsObj.dbg_supportedInterfaces
    sSep = Chr$(10)
  ElseIf sType$ = "m" Then
    s = oInsObj.Dbg_Methods
    sSep = ";"
  ElseIf sType$ = "p" Then
    s = oInsObj.dbg_properties
    sSep = ";"
  Else
    s = ""
    sSep = ""
  End If

  REM Die dbg_Variablen enthalten einleitende Informationen,
  REM die wir nicht gebrauchen können. Also werden sie entfernt.
  REM Wir kümmern uns nur um das, was nach dem Doppelpunkt kommt.
  iPos = InStr(1, s, ":") + 1
  If iPos > 0 Then s = TrimWhite(Right(s, Len(s) - iPos))

  REM Alle Datentypen haben das Präfix Sbx.
  REM All diese Teilstrings "Sbx" werden entfernt.
  s = Join(Split(s, "Sbx"), "")

  REM Wenn der Trennstring NICHT Chr$(10) ist,
  REM werden alle Vorkommen von Chr$(10) entfernt.
  If Asc(sSep) <> 10 Then s = Join(Split(s, Chr$(10)), "")

  REM Teilt den String am Trennstring in Einzelteile
```

```
REM und aktualisiert den Fortschrittsbalken.
oItems() = Split(s, sSep)
oProgress.ProgressValue = 20

REM Dimensionierung der Arrays für die unterschiedlichen Textteile.
REM Der String sieht normalerweise aus wie "SbxString getName()".
REM sFront() enthält den Datentyp, wenn er existiert, andernfalls "".
REM sMid() enthält den Rest.
ReDim sFront(UBound(oItems)) As String
ReDim sMid(UBound(oItems))   As String
ReDim iIdx(UBound(oItems))   As Integer
REM Initialisiert das Indexarray und entfernt Leerzeichen
REM vom Anfang und vom Ende eines jeden Strings.
For i = LBound(oItems()) To UBound(oItems())
  oItems(i) = Trim(oItems(i))
  iIdx(i) = i
  j = InStr(1, oItems(i), " ")
  If (j > 0) Then
    REM Wenn der String mehr als nur ein Wort enthält, wird das erste Wort
    REM in sFront() gespeichert und der Rest des Strings in sMid().
    sFront(i) = Mid$(oItems(i), 1, j)
    sMid(i) = Mid$(oItems(i), j + 1)
    If j > nFrontMax Then nFrontMax = j
  Else
    REM Wenn der String nur aus einem Wort besteht, ist sFront() leer,
    REM und der String wird in sMid() gespeichert.
    sFront(i) = ""
    sMid(i) = oItems(i)
  End If
Next

oProgress.ProgressValue = 40
REM Sortiert die relevanten Namen. Das Array selbst bleibt unverändert,
REM aber die Indexwerte des Arrays iIdx() erlauben einen sortierten Durchlauf.
SortMyArray(sMid(), iIdx())
oProgress.ProgressValue = 50

REM Es geht hier um Eigenschaften, also soll auch der Wert
REM jeder einzelnen Eigenschaft ermittelt werden.
If sType$ = "p" Then
  Dim oPropInfo  'Das Objekt PropertySetInfo.
  Dim oProps     'Array der Eigenschaften.
  Dim oProp      'com.sun.star.beans.Property
  Dim v          'Der Wert einer einzelnen Eigenschaft.
  Dim bHasPI As Boolean
  Dim bUsePI As Boolean
  Dim bConvertReturns As Boolean

  bConvertReturns = True
  bHasPI = False
  bUsePI = False
  On Error Goto NoPropertySetInfo
  oPropInfo = oInsObj.getPropertySetInfo()
  bHasPI = True
  NoPropertySetInfo:

  On Error Goto BadErrorHere
```

```
      For i = LBound(sMid()) To UBound(sMid())
        If bHasPI Then
          bUsePI = oPropInfo.hasPropertyByName(sMid(i))
        End If

        If bUsePI Then
          v = oInsObj.getPropertyValue(sMid(i))
        Else
          v = RunFromLib(GlobalScope.BasicLibraries, _
                  "xyzzylib", "TestMod", oInsObj, sMid(i), False, False)
        End If
        sMid(i) = sMid(i) & " = " & _
              ObjToString(v, sArraySep, iArrayMaxDepth, bConvertReturns)
      Next
    End If
    oProgress.ProgressValue = 60

    nFrontMax = nFrontMax + 1
    iCount = LBound(oItems())
    REM Nun wird das Array der Elemente in sortierter Folge erstellt.
    REM Manchmal wird ein Service mehrmals gelistet.
    REM Diese Routine entfernt dublette Vorkommen desselben Service.
    For i = LBound(oItems()) To UBound(oItems())
      sNew = sFront(iIdx(i)) & " " & sMid(iIdx(i))
      'Wenn Sie den Raum zwischen dem vorderen Teil und dem Rest mit Leerzeichen
      'füllen wollen, damit der Rest ausgerichtet wird, lassen Sie die nächsten
      'Zeilen unkommentiert. Das wirkt allerdings nur richtig, wenn die Schriftart
      'eine feste Zeichenbreite hat.
      'sNew = sFront(iIdx(i))
      'sNew = sNew & Space(nFrontMax - Len(sNew)) & sMid(iIdx(i))
      If i = LBound(oItems()) Then
        oItems(iCount) = sNew
      ElseIf oItems(iCount) <> sNew Then
        iCount = iCount + 1
        oItems(iCount) = sNew
      End If
    Next
    oProgress.ProgressValue = 75
    ReDim Preserve oItems(iCount)
    Exit Sub
  BadErrorHere:
    MsgBox "Fehler " & Err & ": " & Error & Chr(13) & "In Zeile : " & Erl
  End Sub
```

Der Objektinspektor ist unten abgebildet. Inspiziert wurde ThisComponent, ich habe die Eigenschaften anzeigen lassen und darin „ChapterNumberingRules“ markiert. Dann habe ich auf **Inspektion Auswahl** geklickt. Der Pfad steht im Dialogtitel.

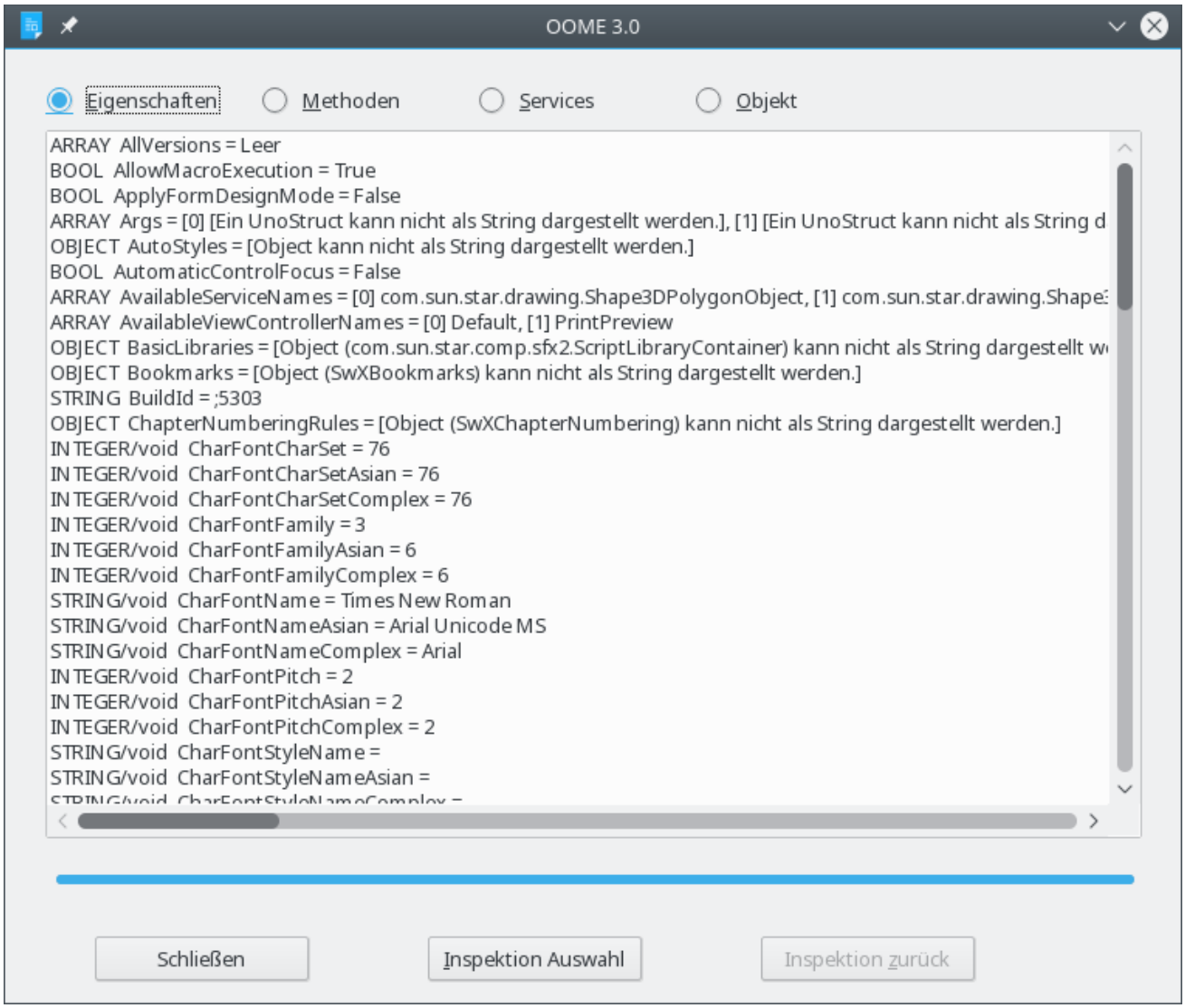


Bild 152. *Der Dialog des Objektinspektors.*

Mit dem Objektinspektor können Sie Objekte untersuchen, wenn Sie nicht ganz sicher sind, was Sie mit ihnen anfangen können. Vielleicht möchten Sie den Inspektor in eine Bibliothek auf Anwendungsebene kopieren, so dass Sie von allen Ihren Makros aus darauf zugreifen können.

18.8.5. Werte für Eigenschaften ermitteln

Die meisten UNO-Objekte haben ein PropertySetInfo-Objekt, mit dessen Hilfe Sie sehen können, welche Eigenschaften existieren, auf die Eigenschaften zugreifen und deren Werte setzen können. Seltsamerweise sind nicht alle Eigenschaften über dieses Interface verfügbar, doch über die Eigenschaft dbg_properties kommen Sie daran. Wenn das Objekt PropertySetInfo nicht in der Lage ist, den Wert einer Eigenschaft zu holen, dann kommt die Routine RunFromLib ins Spiel.

Das folgende Makro erzeugt eine Bibliothek, ein Modul und eine Funktion, die dann aufgerufen wird. Die Namen der Bibliothek und des Moduls werden übergeben. Wenn das Modul existiert, wird es gelöscht und mit einer einzigen Funktion ersetzt. Es geht nur darum, auf eine Eigenschaft unter ihrem Namen zuzugreifen und deren Wert zurückzugeben.

Listing 588. *Erzeugt eine Bibliothek, ein Modul und eine Funktion, die dann aufgerufen wird.*

```
'*************************************************************************
'** Erzeugt ein Modul, das eine Funktion enthält, die den Wert einer
'** Eigenschaft zurückgibt oder die eine Methode eines Objekts aufruft.
'**
```

```
'** Nicht alle Eigenschaften sind über das Objekt PropertySetInfo erreichbar,
'** und mir ist kein anderer Weg bekannt, eine Objektmethode aufzurufen.
'**
'** oLibs  - Der verwendetet Bibliothekscontainer.
'** sLName - Name der Bibliothek.
'** sMName - Name des Moduls.
'** oObj   - Objekt des Interesses.
'** sCall  - Name der aufzurufenden Methode oder Eigenschaft.
'** bClean - Falls True, wird das Modul gelöscht.
'**           Die Bibliothek wird nur gelöscht, wenn sie erzeugt wurde.
'** bIsMthd- Falls True, wird eine Methode und keine Eigenschaft erzeugt.
'** x      - Falls vorhanden, erster Parameter für den Methodenaufruf.
'** y      - Falls vorhanden, zweiter Parameter für den Methodenaufruf.
'**
'*************************************************************************
Function RunFromLib(oLibs, sLName$, sMName$, oObj, sCall$, _
                    bClean As Boolean, bIsMthd As Boolean, _
                    Optional x, Optional y)
  Dim oLib     'Die Bibliothek, in der die neue Funktion laufen soll.
  Dim s$       'Allgemeine Stringvariable.
  Dim bAddedLib As Boolean

  REM Falls die Bibliothek nicht existiert, wird sie erzeugt.
  bAddedLib = False
  If Not oLibs.hasByName(sLName) Then
    bAddedLib = True
    oLibs.createLibrary(sLName)
  End If
  oLibs.loadLibrary(sLName)
  oLib = oLibs.getByName(sLName)

  If oLib.hasByName(sMName) Then
    oLib.removeByName(sMName)
  End If

  s = "Option Explicit" & Chr$(10) & _
      "Function MyInsertedFunc(ByRef oObj"

  If Not IsMissing(x) Then s = s & ", x"
  If Not IsMissing(y) Then s = s & ", y"
  s = s & ")" & Chr$(10) & "On Local Error Resume Next" & Chr$(10)
  s = s & "MyInsertedFunc = oObj." & sCall

  If bIsMthd Then
    s = s & "("
    If Not IsMissing(x) Then s = s & "x"
    If Not IsMissing(y) Then s = s & ", y"
    s = s & ")"
  End If

  s = s & Chr$(10) & "End Function"

  oLib.insertByName(sMName, s)

  If IsMissing(x) Then
    RunFromLib = MyInsertedFunc(oObj)
```

```
    ElseIf IsMissing(y) Then
      RunFromLib = MyInsertedFunc(oObj, x)
    Else
      RunFromLib = MyInsertedFunc(oObj, x, y)
    End If

    If bClean Then
      oLib.removeByName(sMName)
      If bAddedLib Then
        oLibs.removeLibrary(sLName)
      End If
    End If
  End Function
```

Achtung Beim Debuggen des Codes zur Objektinspektion kam es vor, dass das System einen seltsamen Zustand einnahm. Das vorhandene Modul konnte nicht gelöscht werden. Ich glaube, es hatte damit zu tun, dass eine Ereignisbehandlung aufgerufen wurde, in der Haltepunkte gesetzt waren, und dann der Dialog gekillt wurde, während die Haltepunkte noch aktiv waren. Was aber nun wichtig ist, wenn das Modul nicht gelöscht werden konnte: Beenden Sie den Objektinspektor und entfernen Sie die Bibliothek mit dem Namen „xyzzlib“ mit Hilfe der Makroverwaltung.

18.9. Fazit

Die in diesem Kapitel skizzierten Methoden sollten Ihnen eine Starthilfe zur Verwendung von Dialogen und Kontrollelementen sein. Die Methoden des Zugriffs und der Nutzung von Kontrollelementen in Dialogen sind dem Zugriff auf Kontrollelemente in Formularen sehr ähnlich, so dass Sie auch über ein gutes Rüstzeug zur Nutzung von Formularen verfügen. Wenn ein Kontrollelement in einem Dokument und nicht in einem Dialog gespeichert ist, dann ist es in einem Formular gespeichert.

19. Informationsquellen

Das Innenleben von OOo ist sehr ausladend, nicht vollständig dokumentiert und im Dauerzustand der Veränderung, weil laufend neue Funktionalitäten hinzukommen und Fehler berichtigt werden. In diesem Kapitel werden Ihnen Informationsquellen vorgestellt, mit deren Hilfe Sie Ihre eigenen Lösungen finden können.

Bei jeder Art von Problemlösung ist es höchst wichtig, ein Grundverständnis des Problems zu haben, und zu wissen, wie Sie an die wesentlichen Informationen kommen, die Ihnen bei dem Problem und den Lösungsmöglichkeiten fehlen. Sie könnten schnelleren und verlässlicheren Code schreiben, wenn Sie alles über OOo wüssten. Das ist aber schlichtweg unmöglich bei dem gewaltigen Umfang des Produkts. Auch wenn Sie das alles wüssten und mit den Änderungen in den fortlaufenden OOo-Versionen Schritt halten könnten, wäre es wahrscheinlich produktiver, einen Teil der dafür erforderlichen Gedächtnisleistung dazu zu verwenden, sich die Telefonnummern Ihrer engsten Freunde zu merken, oder Ihren Hochzeitstag oder lustige Erlebnisse mit Ihren Eltern – oder daran zu denken, Ihren Freunden, Verwandten und Eltern dieses Buch zu empfehlen.

Es gibt viele ausgezeichnete Informationsquellen über OOo, den jeweiligen Zwecken entsprechend. Das Wissen darüber, welche Information an welcher Stelle zur Verfügung steht, kann viel Zeit und Mühen bei der Problemlösung sparen.

19.1. Die in OOo eingebauten Hilfetexte

Verachten Sie nicht die ausgezeichnete Hilfe, die im Programmpaket von OOo mit dabei ist (s. Bild 153). Obwohl es vielleicht so aussieht, als würde ich dauernd über fehlerhafte oder fehlende Punkte in den Hilfetexten klagen, so sind diese Stellen in Wirklichkeit doch äußerst dünn gesät. Es ist halt wichtig, auf Unzulänglichkeiten hinzuweisen, so dass man auf das eine oder andere unvermeidliche Schlagloch vorbereitet ist. (Na, was wäre dieses Buch denn wert, wenn ich nicht hier und da mal eine eigene Erkenntnis einfließen ließe. Schließlich hält mich noch ein Alltagsjob auf Trab – dieses Buch soll doch für alle Beteiligten keine Zeitverschwendung sein!).

Die Hilfeseiten umfassen alle Komponenten von OOo: Writer, Calc, Basic, Draw, Math und Impress. Über die Aufklappliste in der Ecke links oben wählen Sie aus, zu welcher Komponente Sie Hilfe benötigen. Um die Hilfeseiten zu Basic zu sehen, wählen Sie „OpenOffice Basic“ aus. Das geschieht automatisch, wenn Sie das Hilfefenster aus der Basic-IDE heraus öffnen. Die mitgelieferte Hilfe bietet eine Menge aktueller Informationen über Basic-Syntax und -Routinen.

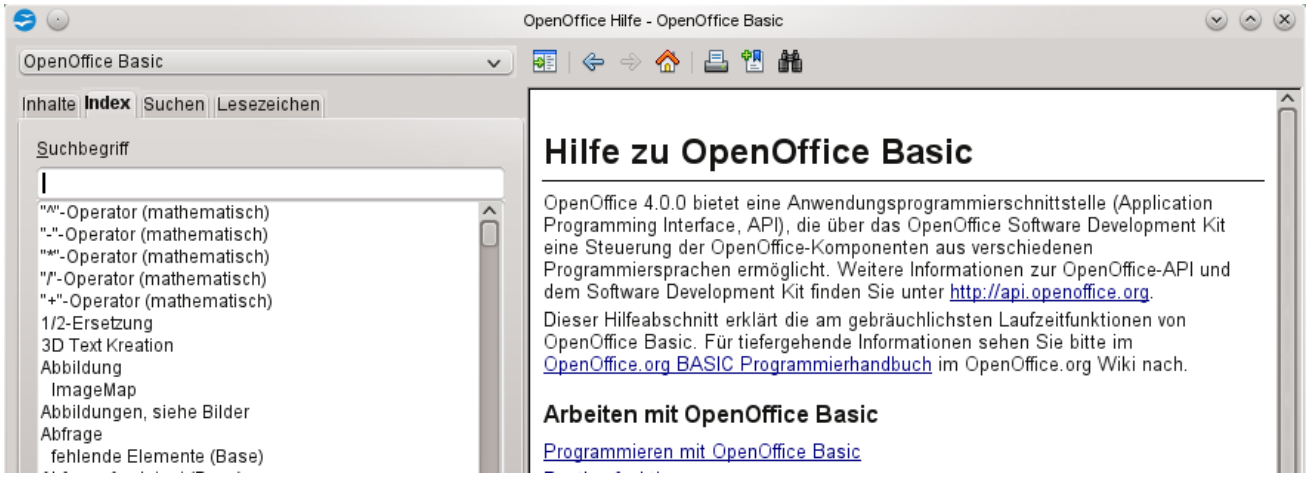


Bild 153. *Die mitgelieferte Basic-Hilfe ist sehr gut.*

19.2. In OOo mitgelieferte Makros

OOo stellt viele Makros in Bibliotheken bereit. Gönnen Sie sich die Zeit, diese Makros zu erkunden. Sie werden viele wunderbare Beispiele finden. Zum Beispiel enthält die Bibliothek Gimmicks das Modul ReadDir, das ein komplettes Verzeichnis einliest und die Informationen als Baumstruktur in einem Draw-Dokument ausgibt. Auch die Bibliothek Tools bietet etliche ausgezeichnete Beispiele. Schauen Sie einmal in das Modul Debug und probieren zum Beispiel WriteDbgInfo und PrintDbgInfo aus.

Wenn Sie nicht wissen, wie Sie ein Problem lösen können, sollten Sie immer zuerst nach einer Antwort suchen, bevor Sie darangehen, ein völlig neues Konzept zu erfinden. Die Makros in OOo sind eine gute erste Anlaufstelle. Ich habe einmal in einer Newsgroup danach gefragt, wie man in einem Dialog die Größe eines Kontrollelements ermittelt. Der Antwortende zeigte mir die Lösung, wies mich aber auch darauf hin, im Modul ModuleControls der Bibliothek Tools nachzusehen. Die mit OOo mitgelieferten Makros enthielten also die Lösung für mein Problem.

Tipp

So können Sie die mit OOo mitgelieferten Bibliotheken und Module erreichen:

Öffnen Sie den Makrodialog mit **Extras | Makros | Makros verwalten | Basic**.

Die auf der Anwendungsebene verfügbaren Makros finden Sie im Bibliothekscontainer **OpenOffice Makros**.

19.3. Websites

Es ist nicht nur einmal vorgekommen, dass ich wohl wusste, was ich zur Problemlösung benötigte, aber die Details nicht im Einzelnen kannte. Dazu sind funktionierende Beispiele unersetzlich. Im Internet gibt es zahlreiche Hilfequellen, und es werden immer mehr. Die folgende Liste von Adressen kann Ihnen eine Hilfe sein.

Tipp

Ich habe mich bemüht, die Links und Beschreibungen korrekt anzugeben, aber die OpenOffice-Websites sind dauernder Veränderung und Verbesserung unterworfen.

19.3.1. Referenzmaterial

http://www.openoffice.org/api/ – für LO https://api.libreoffice.org/ – ist die Site, mit der ich die meiste Zeit verbringe. Hier finde ich die meisten Interfaces und Services aufgelistet und den grundlegenden Developer‘s Guide. Dieses Handbuch hat den Ruf, schwer verständlich zu sein, aber die Autoren arbeiten daran, den Inhalt auch für weniger technisch orientierte Leser zugänglicher zu machen.

http://www.odfauthors.org/: Dokumentation und Bücher.

http://wiki.openoffice.org/wiki/Documentation: Das Apache-Dokumentationsprojekt.

19.3.2. Makrobeispiele

http://www.pitonyak.org/oo.php ist meine persönliche Website, die ich schon eingerichtet hatte, bevor ich den Plan fasste, ein Buch zu schreiben. Die Site bietet viele Dokumente und Links. „Andrew-Macro“ enthält vielfältige Beispiele. Der direkte Link zu meinem Makrodokument ist http://www.pitonyak.org/AndrewMacro.odt.

http://kienlein.com/pages/oo.html mit vielen ausgezeichneten Makros, mit Kurzbeschreibungen in Deutsch.

http://www.darwinwars.com/lunatic/bugs/oo_macros.html ist ein guter Ausgangspunkt auf der Suche nach Writer-Makros. Der Autor der Site ist professioneller Schriftsteller und verwendet die Makros selbst – also funktionieren sie!

19.3.3. Verschiedenes

http://www.openoffice.org/ (englisch) ist der Hauptlink zu AOO (die Links in http://www.openoffice.org/de/ führen leider in der Mehrheit zu den englischsprachigen Sites). Diese Site bietet eine Menge an Informationen, zum Beispiel Links zu Mailinglisten, die Sie durchsuchen können, und zum Fehlerverwaltungssytem (bug-tracking) IssueZilla. Sie können sich dort anmelden (kostet nichts) und weitere Berechtigungen erhalten, so die Möglichkeit, Fehler zu melden und Mailinglisten zu abonnieren.

Die englische Homepage von LO ist https://www.libreoffice.org/, die deutsche https://de.libreoffice.org/. Die Beiträge dort sind zu einem großen Teil wirklich in deutscher Sprache vorhanden.

http://www.openoffice.org/development/index.html ist der Link zum Entwicklerprojekt. Die Site bietet den Zugriff auf eine Vielzahl von Links und Quellen.

http://forum.openoffice.org/ ist das offizielle OpenOffice-Forum.

http://www.oooforum.org/ enthält viele Hilfeforen, auch speziell für die einzelnen OOo-Komponenten, einschließlich Makroprogrammierung.

http://ooodocs.sourceforge.net/ enthält viele Hilfedateien, in Kategorien untergliedert, inklusive eines Abschnitts über Makroprogrammierung.

http://www.openoffice.org/documentation/ ist die Homepage für das Dokumentationsprojekt. Sie enthält Dokumentationen zu vielerlei Themen. Zwar sind die How-tos umgezogen zu http://wiki.services.openoffice.org/wiki/Documentation/How_Tos/, aber es gibt immer noch http://www.openoffice.org/documentation/HOW_TO/various_topics/, in dem Sie unter anderem ein ganz ausgezeichnetes, wenn auch betagtes, interaktives Dokument finden können, in dem gezeigt wird, wie Kontrollelemente, die Makros aufrufen, in Dokumente eingefügt werden: How_to_use_basic_macros.sxw.

http://sourceforge.net/project/showfiles.php?group_id=43716 enthält zahlreiche Mustervorlagen und Beispiele.

Deutschsprachige Dokumentationen zur Makroprogrammierung sind noch rar gesät. Allerdings werden die Foren sehr gut administriert und hochkarätig frequentiert.

- http://de.openoffice.info/ ist das deutsche Forum rund um AOO und LibreOffice. Es gibt ein Unterforum „Makros und allgemeine Programmierung“.
- http://www.libreoffice-forum.de/ ist das Forum rund um das LibreOffice Softwarepaket. Auch dort gibt es ein Unterforum „LibreOffice Programmierung“.

19.4. http://www.openoffice.org/api/ oder http://api.libreoffice.org

Für den ernsthaften Makroprogrammierer sind http://www.openoffice.org/api/ , bzw. http://api.libreoffice.org außerordentlich wichtige Sites. Auf der Suche nach Problemlösungen verbringe ich dort die meiste Zeit. Es gibt dort einen Link auf den Basic-Programmier-Guide für StarOffice 8, der aber zur Zeit ins Leere führt. Doch auch die Links zu den älteren Guides der Versionen 7 und 6 sind noch heute nutzbar und bieten viele Beispiele.

http://wiki.openoffice.org/wiki/Documentation/DevGuide/OpenOffice.org_Developers_Guide enthält den Guide für die Entwickler. Er ist immer auf dem aktuellen Stand. Die Zielgruppe dieses Dokuments ist die der professionellen Entwickler. Eine große Menge an Informationen ist hier zusammengetragen, und es bedarf schon eines gewissen Talents, das zu finden, was man sucht, einfach nur, weil die Datenmenge so riesig ist. Die Programmierung mit Basic deckt nur einen kleinen Teil des Dokuments ab. Wenn Sie die vollständige Geschichte haben wollen, oder wenigstens das, was der Vollständigkeit nahe kommt, schauen Sie in den Entwickler-Guide. (Aber seien Sie auf die *vollständige* Geschichte gefasst ...).

Die meiste Zeit verbringe ich dort, wo ich die einzelnen Module durchsuchen kann, in denen die verfügbaren Interfaces und Services aufgelistet sind. Ein guter Ausgangspunkt für AOO ist http://www.openoffice.org/api/docs/common/ref/com/sun/star/module-ix.html, denn dort sind alle Hauptmodule versammelt. Beachten Sie das „module-ix.html“ am Ende der Adresse. Wenn Sie gerade eine Site über einen Service oder ein Interface betrachten, können Sie in der Adresse normalerweise einfach anstelle des Namens der Site den Namen „module-ix.html“ einsetzen. Angenommen, Sie informieren sich gerade über den Service TextCursor:
http://www.openoffice.org/api/docs/common/ref/com/sun/star/text/TextCursor.html.

Sie wollen nun die Services sehen. Ändern Sie „TextCursor.html“ in „module-ix.html“ um, und Sie erhalten eine Auflistung aller Definitionen, Services, enthaltenen Module, Konstanten und Interfaces des Moduls com.sun.star.text.

https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star.html ist die Startadresse für den Überblick über die Module in LO. Die URL-Ergänzungen für die einzelnen Informationsseiten sind nicht so einfach strukturiert, so dass Sie eher über die Links im Inhalt navigieren wollen.

Beginnen Sie Ihre Recherche der verschiedenen Module auf der jeweiligen Hauptsite, wo Sie ein Gefühl für die einzelnen Module und ihre Verwendungszwecke bekommen können. Weil die Module zusammengehörende Funktionalitäten bündeln, schaue ich manchmal hier nach, wenn ich einen Service finden muss, der ein bestimmtes Problem löst, ich aber nicht weiß, welcher Service es ist.

Zur Suche nach bestimmten bekannten Elementen bietet sich der Index in AOO an (http://www.openoffice.org/api/docs/common/ref/index-files/index-1.html). LO stellt auf jeder IDL-Seite ein Such-Eingabefeld zur Verfügung, das interaktiv ohne Berücksichtigung von Groß- und Kleinschreibung alle Treffer vom Beginn des Suchbegriffs an auflistet – sehr praktisch.

19.5. Mailinglisten und Foren

Für OpenOffice-Benutzer gibt es eine ganze Reihe von Mailinglisten. Informationen über die jeweiligen Listen erhalten Sie auf folgenden Sites:

- http://www.openoffice.org/mail_list.html
- http://www.libreoffice.org/get-help/mailing-lists/

Wenn Sie in einer Mailingliste eine Frage stellen, gehen die Antworten nicht an Sie persönlich, sondern an die Liste. Daher müssen Sie sich in der Liste anmelden, sonst erhalten Sie keine Antworten. Wie sie sich anmelden (subskribieren), ist jeweils genau beschrieben. Obwohl auch die Entwickler mitlesen und Fragen beantworten, ist eine Mailingliste kein „Produktsupport“ mit Anspruch auf eine garantierte Hilfeleistung. Wenn andere Benutzer oder Entwickler eine Lösung haben oder vielleicht auch nur den Ansatz für eine Lösung, werden sie eine Antwort posten (Slang für einen Beitrag in einem Forum oder in einer Mailingliste). Wenn die Antwort viel Arbeit und Zeit benötigt, ist die Wahrscheinlichkeit groß, dass gar keine Antwort kommt. Die Mailingliste für die Entwicklung von LO ist für die Leute gedacht, die sich mit der Weiterentwicklung von LO befassen, also werden Sie in dieser Liste mit einer Frage über Makros keine Aufmerksamkeit finden.

Der erste Schritt vor der Fragestellung ist also die Subskription der Liste. Denken Sie immer daran, dass die Leute, die auf Ihre Frage eingehen, nicht dafür bezahlt werden. Es ist auch gute Sitte, auf die Fragen anderer zu antworten, wenn Sie die Lösung kennen. Im Folgenden finden Sie ein paar Tipps, wie Sie in Mailinglisten mit sicheren Antworten rechnen können.

Bevor Sie eine Frage stellen, versuchen Sie erst einmal, die Lösung selbst herauszufinden. Suchen Sie im Internet, durchsuchen Sie die Archive der Mailinglisten, suchen Sie im Entwickler-Guide, suchen Sie in Google, lesen Sie die FAQs. Inspizieren Sie die von Methoden zurückgegebenen Objekte. Die Mitglieder der Benutzergemeinde freuen sich, Menschen zu helfen, die sich selbst helfen können und die ihr Wissen der Gemeinschaft zurückgeben.

Geben Sie Ihrer Anfrage eine aussagekräftige Betreffzeile. Ich zum Beispiel erhalte jeden Tag Hunderte von E-Mails, so ist es mehr als wahrscheinlich, dass ich eine Nachricht überspringe, die ein Be-

treff wie „Bitte um Hilfe“ oder „Dringend“ oder „Makroproblem“ trägt. Ein besseres Betreff wäre „Wie kann ich ein Makro von der Befehlszeile starten?“ oder „Suchmakro findet keine Attribute“. Dann kann ich mich auf Fragen konzentrieren, deren Beantwortung ich wahrscheinlich schon kenne oder die mich interessieren. Wir alle erhalten viel zu viele E-Mails, viel mehr als wir brauchen. Nichtssagende oder irreführende Betreffzeilen laden oft zum lockeren Gebrauch der Löschtaste ein.

Achten Sie auf die Formulierung Ihrer Frage, damit sie auch klar verstanden wird. „Mein Makro läuft in einen Laufzeitfehler“ ist ziemlich unklar. Hilfreich wären ein paar Codezeilen, ebenso der Wortlaut der Fehlermeldung. Der Umstand, dass Ihr Sortier-Makro nicht wie gewünscht sortiert, mag zwar irgendwie spezifisch sein, aber wenn ich das kommentieren soll, brauche ich mehr Informationen, vielleicht ein paar Zeilen Quelltext. Beschreiben Sie das Problem präzise und geben Sie auch an, welche Lösungen Sie versucht haben. Konkrete Fehlermeldungen und alles, was Sie an Hintergrundinformationen und möglichen Problemursachen haben, sind sehr willkommen.

Erwarten Sie nicht, dass jemand für Sie die Arbeit macht. „Mein Kunde zahlt mir 2000 € für die Konvertierung dieses Makros. Kann das jemand umsonst für mich machen?“ wird sehr wahrscheinlich keine Antwort nach sich ziehen. Ich habe schon Makros aus anderen Systemen übersetzt, weil sie interessant waren und einen breiten Anwendungsbereich umfassten und weil ich sie anschließend der Benutzergemeinde öffentlich zugänglich machen durfte. Wenn Sie wirklich in der Bredouille sind, wird sich sicher jemand in der Liste bereit erklären, die Arbeit gegen Bezahlung zu erledigen.

Sprechen Sie keine Einzelperson an, auch wenn sie sachkundig ist. Viele können Ihre Frage beantworten, und über den Wissenspool erhält man sehr oft eine bessere Lösung. Adressieren Sie also Ihr Anliegen an die Liste und nicht an eine Einzelperson, außer Sie haben einen sehr triftigen Grund dafür. Die letzte Frage an die Liste, zu deren Beantwortung ich beitragen konnte, erhielt circa 10 Folgenachrichten von vier verschiedenen Personen.

Es ist auch schlechter Stil, darum zu bitten, die Antworten auf die Frage an eine andere E-Mailadresse zu schicken. Wenn es Ihnen nicht wert ist, in der Liste nach einer Antwort zu suchen, warum sollte es einem anderen überhaupt wert sein, Zeit für die Beantwortung Ihrer Frage aufzuwenden? Noch einmal, denken Sie daran, dass die Leute freiwillig dabei sind! Es kommt hinzu, dass die Antwort auf Ihr Problem – eine Online-Diskussion, die zum Verständnis Ihrer Programmsituation und ihrer möglichen Lösung führt – auch für andere in der Gemeinschaft wichtig sein kann. Auf diese Weise vermehrt und verbessert sich das Wissen im Pool von Mal zu Mal.

Wenn Sie eine Lösung zu einer Frage kennen, posten Sie sie. Damit helfen Sie anderen. Ein höfliches „Danke“ ist auch immer angebracht. Höflichkeit und die Bereitschaft zu teilen sind für die Entwicklung einer funktionierenden Gemeinschaft wichtig, zum allseitigen Nutzen für Sie und die anderen Mitglieder. (Oje, das klingt wie ein Werbespot für eine öffentliche Einrichtung, ist aber dennoch richtig.)

Ein Hinweis noch zur Netiquette in deutschsprachigen Foren und Mailinglisten. In allen ist es üblich, dass man sich duzt. Diese Form der Anrede wird wie auch in den Gewerkschaften und der SPD als Zeichen der Zugehörigkeit zu einer Gemeinschaft Gleichgesinnter verstanden, in diesem Fall zur Gemeinschaft der Nutzer einer Open-Source-Anwendung, nämlich OpenOffice. Bei Nichtbeachtung kann es sein, dass man mehr oder weniger diskret auf den üblichen Umgangston, das Du, hingewiesen wird.

Dass der Übersetzer dieses Buches den Leser siezt, liegt daran, dass es ihm nach der Lektüre so vieler Bücher in seinem Leben, in denen fast ausnahmslos die förmliche Anrede gewählt wurde, nicht gelingt, sich von den Usancen im Medium Buch zu lösen. Aber seien Sie sicher, lieber Leser, im Usenet duze ich Sie. (Das gilt natürlich auch für die Leserin.)

19.6. Die Suche nach Antworten

Wenn ich etwas über ein bestimmtes Thema wissen möchte, befrage ich den Guide für Entwickler und das alte StarOffice-Tutorial und schließlich noch Google (oder DuckDuckGo) mit so etwas wie „cursor OpenOffice“. Wenn ich die Suche auf die Site „http://www.openoffice.org/api/docs“ beschränken möchte, schreibe ich „site:http://www.openoffice.org/api/docs cursor“. Die Resultate enthalten normalerweise ein Interface oder einen Service. Daran anschließend kann ich eine Inspektion vornehmen:

```
MsgBox  vObj.dbg_methods
MsgBox  vObj.dbg_supportedInterfaces
MsgBox  vObj.dbg_properties
```

LibreOffice hat eine ganz ähnliche Site: „http://api.libreoffice.org“, die Sie ebenso durchsuchen können.

Wenn Sie nicht wissen, was Sie mit einem Objekt tun können, inspizieren Sie es. Lassen Sie sich zum Beispiel die Eigenschaften dbg_methods, dbg_supportedInterfaces und dbg_properties anzeigen. Wenn Sie immer noch nicht weiterkommen, können Sie das Internet oder andere Dokumente nach den eingebundenen Services, Methoden und Eigenschaften durchsuchen.

Tipp Ich lasse die Methoden und Eigenschaften nicht mit Print oder MsgBox ausgeben, sondern verwende X-Ray oder den Objektinspektor (s. Abschnitt 18.8. Das Beispiel Objektinspektor).

Dehnen Sie die Suche auch auf eine Schnellsuche nach Beispielcode aus, der Ihr aktuelles Problem vielleicht lösen kann. Das könnte ein Idealfall sein, denn Ihr Arbeitsaufwand würde sich stark reduzieren. Erst wenn alles nichts gebracht hat, wenden Sie sich um Hilfe an eine Mailingliste.

19.7. Fazit

In vielen ausgezeichneten Informationsquellen wird erläutert, wie man Code für OOo schreibt. Machen Sie sich mit diesen Quellen vertraut, und Sie sparen eine Menge Zeit. Und wenn Sie regelmäßig durch die Mailinglisten stöbern, bleiben Sie auf dem Laufenden. Die Mitgliedschaft in einer Mailingliste oder einem Forum bietet die Möglichkeit, Lösungen für Ihre Probleme zu erhalten und zugleich andere von Ihrer Sachkenntnis profitieren zu lassen.

Anhang 1. Verzeichnis der Abbildungen

Anhang 2. Verzeichnis der Tabellen

2. Die Grundlagen

3. Sprachstrukturen

4. Numerische Routinen

5. Array-Routinen

6. Datums- und Uhrzeit-Routinen

7. String-Routinen

8. Dateiroutinen

9. Diverse weitere Routinen

10. Universal Network Objects (UNO)

15. Tabellendokumente

16. Zeichnungs- und Präsentationsdokumente

17. Verwaltung der Bibliotheken

18. Dialoge und Steuerelemente

Anhang 3. Verzeichnis der Listings

4. Numerische Routinen

5. Array-Routinen

6. Datums- und Uhrzeit-Routinen

7. String-Routinen

8. Dateiroutinen

9. Diverse weitere Routinen

10. Universal Network Objects (UNO)

14. Textdokumente

15. Tabellendokumente

16. Zeichnungs- und Präsentationsdokumente

17. Verwaltung der Bibliotheken

18. Dialoge und Steuerelemente