StarOffice™ 7 Office Suite

A Sun™ ONE Software Offering

Basic Programmierhandbuch

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A. 650-960-1300

Part No. 817-3924-10
2003, Revision A

Copyrights and Trademarks

Inhalt

Einleitung

Dieses Handbuch führt in die Programmierung mit StarOffice Basic 6.0 ein und zeigt, welche Anwendungsmöglichkeiten sich durch die Verwendung von StarOffice Basic in StarOffice eröffnen. Es wendet sich an Programmierer, die bereits Erfahrung mit einer anderen Programmiersprache gesammelt haben.

Eine Fülle von Beispielen gibt dem Leser wertvolle Hilfen für die Entwicklung eigener Programme an die Hand.

Speziell für Programmierer, die bisher mit Microsoft Visual Basic for Applications oder einer älteren Version von StarOffice Basic gearbeitet haben, stehen eine Reihe von Migrations-Tipps zur Verfügung. Diese sind durch ein kleines Symbol am Seitenrand hervorgehoben. Im Anhang dieses Buches findet sich ein Index zum Quereinstieg für Nutzer dieser Programmiersprachen.

Über StarOffice Basic

Die Programmiersprache StarOffice Basic wurde speziell für StarOffice entwickelt und ist fest in das Office-Paket integriert.

Wie der Name bereits nahe legt, handelt es sich bei StarOffice Basic um eine Programmiersprache aus der Basic-Familie. Wer bisher mit anderen Basic-Sprachen gearbeitet hat – insbesondere mit Visual Basic oder Visual Basic for Applications (VBA) von Microsoft – wird sich schnell in StarOffice Basic zurecht finden. Die Basiskonstrukte von StarOffice Basic sind über weite Teile kompatibel zu den genannten Sprachen von Microsoft.

Die Programmiersprache StarOffice Basic lässt sich in vier Komponenten aufteilen:

- **Die Sprache StarOffice Basic**: Sie legt die elementaren Sprachkonstrukte fest, etwa für Variablendeklarationen, Schleifen und Funktionen.
- **Die Laufzeitbibliothek**: Sie stellt Standard-Funktionen zur Verfügung, die keinen direkten Bezug zu StarOffice besitzen. Dazu zählen beispielsweise Funktionen für das Bearbeiten von Zahlen, Zeichenfolgen, Datumswerten und Dateien.
- **Das StarOffice API (Application Programming Interface)**: Es gestattet den Zugriff auf StarOffice-Dokumente und erlaubt es, diese anzulegen, zu speichern, zu ändern und zu drucken.
- **Der Dialog-Editor**: Er dient zum Erzeugen eigener Dialogfenster und bietet Möglichkeiten, diese mit Steuerelementen und Ereignis-Handlern zu versehen.

Die Kompatibilität zwischen StarOffice Basic und VBA bezieht sich auf die Sprache StarOffice Basic sowie die Laufzeitbibliothek. Das StarOffice API und der Dialog-Editor sind*nicht* zu VBA kompatibel. (Eine Vereinheitlichung dieser Schnittstellen hätte viele der in StarOffice vorhandenen Konzepte unmöglich gemacht.)

An wen sich StarOffice Basic wendet

Der Anwendungsbereich von StarOffice Basic beginnt dort, wo die Standardfunktionen von StarOffice enden. So lassen sich in StarOffice Basic Routine-Aufgaben automatisieren, Verbindungen zu anderen Programmen – etwa einem Datenbankserver – herstellen und komplexe Tätigkeiten in Form von vorgegebenen Skripten auf Knopfdruck ausführen.

StarOffice Basic bietet vollständigen Zugriff auf alle Funktionen von StarOffice, kann alle unterstützten Dokumenttypen modifizieren und bietet Möglichkeiten zur Erstellung eigener Dialogfenster.

Wie sich StarOffice Basic nutzen läßt

StarOffice Basic kann ohne zusätzliche Programme und Hilfsmittel von jedem StarOffice-Anwender verwendet werden. Bereits in der Standardinstallation verfügt es über alle Komponenten, die für die Erstellung eigener Basic-Makros notwendig sind. Dazu zählen:

- **Die integrierte Entwicklungsumgebung** (Integrated Development Environment, IDE), die einen Editor für die Eingabe und das Testen von Makros zur Verfügung stellt.
- **Der Interpreter**, der für das Ausführen von StarOffice Basic-Makros notwendig ist.
- **Die Schnittstellen** zu den verschiedenen StarOffice-Anwendungen, die direkte Zugriffe auf die Office-Dokumente gestatten.

Aufbau dieses Handbuchs

Die ersten Kapitel führen in die Arbeit mit StarOffice Basic ein:

- Kapitel 2: Die Sprache StarOffice Basic
- Kapitel 3: Die Laufzeitbibliothek von StarOffice Basic
- Kapitel 4: Einführung in das StarOffice API

Aufgrund ihres allgemeinen Charakters sollten diese Kapitel von allen Lesern gelesen oder zumindest überflogen werden.

Es folgen Kapitel, die auf einzelne Teilbereiche des StarOffice API eingehen und die daher selektiv gelesen werden können:

- Kapitel 5: Die Arbeit mit StarOffice-Dokumenten
- Kapitel 6: Textdokumente
- Kapitel 7: Tabellendokumente
- Kapitel 8: Zeichnungen und Präsentationen

- Kapitel 9: Diagramme (Charts)
- Kapitel 10: Datenbankzugriff
- Kapitel 11: Dialoge
- Kapitel 12: Formulare

Weiterführende Informationen

Die vorgestellten Funktionsbereiche des StarOffice API wurden nach ihrem Praxisnutzen für den StarOffice Basic-Programmierer ausgewählt. Im Regelfall wird dabei nur ein Ausschnitt der angesprochenen Schnittstellen vorgestellt. Wer sich ein detaillierteres Bild dazu verschaffen möchte, sei auf die API-Referenz verwiesen, die im Internet unter der Adresse

```
http://api.openoffice.org/common/ref/com/sun/star/module-ix.html
```

zu finden ist.

Der Developer's Guide beschreibt das StarOffice API detaillierter als dieses Buch, wendet sich jedoch primär an Java- und C++-Programmierer. Wer mit der StarOffice Basic-Programmierung prinzipiell vertraut ist, findet im Developer's Guide viele Zusatzinformationen, mit deren Hilfe er sein Wissen über StarOffice Basic und die Programmierung von StarOffice vertiefen kann. Der Developer's Guide ist im Internet unter der Adresse

```
http://api.openoffice.org/DevelopersGuide/DevelopersGuide.html
```

zu finden.

Programmierer, die anstelle von StarOffice Basic direkt mit Java oder C++ arbeiten möchten, sollten gleich zu Beginn auf den StarOffice Developer's Guide zurück greifen. Die StarOffice-Programmierung mit Java oder C++ ist jedoch deutlich komplexer ist als die mit StarOffice Basic.

KAPITEL 2

Die Sprache StarOffice Basic

StarOffice Basic gehört zur Familie der Basic-Sprachen. Viele Teile von StarOffice Basic sind identisch mit Microsoft Visual Basic for Applications und Microsoft Visual Basic. Wer mit diesen Sprachen bereits gearbeitet hat, findet sich schnell in StarOffice Basic zurecht.

Aber auch Programmierer anderer Sprachen – etwa von Java, C++ oder Delphi – dürften keine großen Schwierigkeiten haben, sich in StarOffice Basic einzuarbeiten. Längst sind die Zeiten vorbei, in denen Basic nur rudimentäre Kontrollstrukturen wie `GoTo` und `GoSub` bot. Bei StarOffice Basic handelt es sich um eine voll entwickelte prozedurale Programmiersprache.

Auch von den Vorteilen der objektorientierten Programmierung können StarOffice Basic-Programmierer profitieren, denn eine Schnittstelle in StarOffice Basic gestattet es, externe Objektbibliotheken von StarOffice Basic aus zu nutzen. Das gesamte StarOffice API basiert auf derartigen Schnittstellen, die später näher erläutert werden.

Dieses Kapitel gibt einen Überblick zu den Sprachkonstrukten von StarOffice Basic. Es beschreibt den Rahmen, von StarOffice Basic an dem sich alle Anwendungen und Bibliotheken orientieren.

Ein StarOffice Basic-Programm im Überblick

StarOffice Basic ist eine so genannte Interpreter-Sprache. StarOffice verfügt zwar über einen Compiler, der die einzelnen Programmzeilen in Zwischencode übersetzt und in einzelne Funktionen unterteilt. Im Gegensatz zu klassischen Compiler-Sprachen wie C++ oder Turbo Pascal erzeugt StarOffice Basic jedoch keine ausführbaren Dateien, die eigenständig ablaufen können.

Sobald ein StarOffice Basic-Programm erstellt ist, lässt es sich per Knopfdruck ausführen. StarOffice prüft dazu den Programmcode zunächst auf offensichtliche Fehler. Dann führt es den vorbereiteten Zwischencode Zeile für Zeile aus.

Programmzeilen

Die zeilenorientierte Arbeitsweise des Basic-Interpreters führt zu einem wichtigen Unterschied zwischen Basic und anderen Programmiersprachen. Während die Position von Zeilenumbrüchen innerhalb des Quelltextes beispielsweise in Java, C++ und Delphi unerheblich ist, bilden die Zeilen eines Basic-Programms jeweils eine eigene, für sich abgeschlossene Einheit. Funktionsaufrufe, mathematische Ausdrücke und andere Sprachelemente wie Funktions- und Schleifenköpfe müssen stets in der Zeile abgeschlossen werden, in der sie beginnen.

Reicht der Platz dafür nicht aus oder führt dies zu Bandwurmzeilen, so besteht die Möglichkeit, mehrere Zeilen durch das Anfügen von Unterstrichen _ miteinander zu verbinden. Das Beispiel

```
LongExpression = (Expression1 * Expression2) + _
      (Expression3 * Expression4) + _
      (Expression5 * Expression6) + _
      (Expression7 * Expression8)
```

verbindet vier Zeilen eines mathematischen Ausdrucks. Der Unterstrich muss immer das letzte Zeichen in den betreffenden Zeilen sein, ansonsten erzeugt StarOffice Basic eine Fehlermeldung. Ihm dürfen auch keine Leerzeichen oder Tabulatorzeichen folgen.

Neben dem Verbinden einzelner Zeilen bietet StarOffice Basic die Möglichkeit eine Zeile in mehrere Bereiche zu teilen, so dass Raum für mehrere Ausdrücke entsteht. Dazu ist ein Doppelpunkt als Trennzeichen zu verwenden. So lassen sich beispielsweise die Zuweisungen

```
a = 1
a = a + 1
a = a + 1
```

in einer Zeile der Form

```
a = 1  :  a = a + 1  :  a = a + 1
```

darstellen.

Kommentare

Neben dem auszuführenden Programmcode kann ein StarOffice Basic-Programm auch Kommentare enthalten, die einzelne Programmteile erklären und wichtige Informationen liefern, mit deren Hilfe sich der Programmcode zu einem späteren Zeitpunkt, etwa im Rahmen einer Fehlersuche, rekapitulieren lässt.

StarOffice Basic bietet zwei Möglichkeiten zum Einfügen von Kommentaren in den Programmcode:

- Sämtliche Zeichen, die auf einen einfachen Anführungsstrich folgen, gelten als Kommentar:

  ```
  Dim A ' Das ist ein Kommentar für Variable A
  ```

- Das Schlüsselwort `Rem` kennzeichnet einen Kommentar.

  ```
  Rem Dieser Kommentar wurde durch das Schlüsselwort Rem eingeleitet.
  ```

Ein Kommentar umfasst prinzipiell alle Zeichen bis zum Ende der Zeile. Die Folgezeile interpretiert StarOffice Basic hingegen wieder als reguläre Anweisung. Bei mehrzeiligen Kommentaren muss daher jede Zeile als Kommentar gekennzeichnet werden:

```
Dim B       ' Dieser Kommentar für die Variable B ist relativ lang
            ' und erstreckt sich auf mehrere Zeilen. Das
            ' Kommentarzeichen muss daher in jeder Zeile wiederholt
            ' werden.
```

Bezeichner

Ein StarOffice Basic-Programm enthält im Regelfall viele dutzende, hunderte oder gar tausende Namen für Variablen, Konstanten, Funktionen und vieles mehr. Derartige Namen werden auch *Bezeichner* genannt. Bei der Wahl eines Namens als Bezeichner sind einige Regeln zu beachten:

- Bezeichner dürfen nur aus lateinischen Buchstaben, Ziffern und Unterstrichen bestehen.
- Das erste Zeichen eines Bezeichners muss ein Buchstabe oder ein Unterstrich _ sein.
- Bezeichner dürfen keine Sonderzeichen wie ä â î ß enthalten.
- Die Maximallänge eines Bezeichners beträgt 255 Zeichen.
- Es findet keine Unterscheidung zwischen Groß- und Kleinschreibung statt. Der Bezeichner `EineTestVariable` definiert beispielsweise die gleiche Variable wie `einetestVariable` und `EINETESTVARIABLE`.

  Von dieser Regel gibt es jedoch eine Ausnahme: Bei Konstanten des UNO-API wird zwischen Groß- und Kleinschreibung unterschieden! (Mehr zum Thema UNO finden Sie in Kapitel 4.)

Die Regeln für den Aufbau von Bezeichnern unterscheiden sich zwischen VBA und StarOffice Basic. So verbietet StarOffice Basic im Gegensatz zu VBA beispielsweise die Verwendung von Sonderzeichen in Bezeichnern. (Diese können in internationalen Projekten Probleme bereiten.)

Hier einige Beispiele für richtige und falsche Bezeichner:

```
Surname          ' Korrekt
Surname5         ' Korrekt (Ziffer 5 steht nicht an erster Stelle)
First Name       ' Falsch (Leerzeichen nicht erlaubt)
DéjàVu           ' Falsch (Buchstaben wie é, à nicht erlaubt)
5Surnames        ' Falsch (erstes Zeichen darf keine Ziffer sein)
First,Name       ' Falsch (Komma und Punkt sind nicht erlaubt)
```

Arbeiten mit Variablen

Implizite Variablendeklaration

Basic-Sprachen tendieren dazu, es dem Programmierer so einfach wie möglich zu machen. So folgt auch StarOffice Basic den heute üblichen Basic-Sprachen und gestattet es, eine Variable ohne explizite Deklaration durch einfache Benutzung zu erzeugen. Eine Variable existiert, sobald sie das erste Mal verwendet wird. Je nach bereits vorhandenen Variablen kann der Aufruf

```
a = b + c
```

daher bis zu drei neue Variablen erzeugen.

Aus gutem Grunde gilt es jedoch als unsauber Variablen in einem Programm ohne explizite Deklaration einzuführen. Führt doch ein einfacher Tippfehler zum Erzeugen einer neuen Variable. Anstelle einer Fehlermeldung initialisiert der Interpreter die neue Variable einfach mit 0 und lässt diese neben der bereits vorhandenen Variable parallel existieren. Das Auffinden derartiger Fehler kann sehr schwierig sein.

Explizite Variablendeklaration

Um die genannten Fehler durch eine implizite Variablendeklaration zu vermeiden bietet StarOffice Basic einen Schalter namens

```
Option Explicit
```

an. Er muss in der ersten Programmzeile eines jeden Moduls aufgeführt werden und sorgt für eine Fehlermeldung, falls eine der verwendeten Variablen nicht deklariert ist. `Option Explicit` sollte in keinem Basic-Modul fehlen.

Der Befehl für eine explizite Variablendeklaration lautet in seiner einfachsten Form:

```
Dim MyVar
```

Das genannte Beispiel deklariert eine Variable mit dem Namen `MyVar` und dem Typ Variant. Ein Variant ist eine Universalvariable, die alle denkbaren Werte aufnehmen kann, etwa Zeichenfolgen, ganze Zahlen, Fließkommazahlen und Boolesche Werte. Hier einige Beispiele zur Verwendung einer Variant-Variable:

```
MyVar = "Hello World"              ' Zuweisung einer Zeichenfolge
MyVar = 1                          ' Zuweisung einer ganzen Zahl
MyVar = 1.0                        ' Zuweisung einer Fließkommazahl
MyVar = True                  ' Zuweisung eines Booleschen Wertes
```

Die oben deklarierte Variable kann sogar in ein- und demselben Programm für verschiedene Typen verwendet werden. Das bringt zweifelsohne eine Menge Flexibilität mit sich. Im Regelfall empfiehlt es sich aber dennoch, Variablen auf einen bestimmten Typ zu beschränken. Mit der Typ-Festlegung einer Variablen kann StarOffice Basic bereits frühzeitig herausfinden, ob eine Variable in einem bestimmten Kontext falsch verwendet wird, und durch die Generierung einer entsprechenden Fehlermeldung warnen.

Eine typgebunde Variablendeklaration erfolgt über eine Anweisung im Stile:

```
Dim MyVar As Integer    ' Deklaration einer Variable mit dem Typ Integer
```

Die genannte Variable hat den Typ Integer und kann ganzzahlige Werte aufnehmen. Alternativ zur Typfestlegung über den Ausdruck `As Integer` besteht die Möglichkeit, den Typ durch ein nachfolgendes Typ-Deklarationszeichen festzulegen:

```
Dim MyVar%              ' Deklaration einer Variablen mit dem Typ Integer
```

Die beiden Variablendeklarationen sind inhaltlich identisch.

Die `Dim`-Anweisung kann nicht nur eine, sondern mehrere Variablendeklarationen in einer Zeile aufnehmen:

```
Dim MyVar1, MyVar2
```

Soll den Variablen dabei ein fester Typ zugeordnet werden, so muss dieser für jede Variable getrennt aufgeführt werden:

```
Dim MyVar1 As Integer, MyVar2 As Integer
```

Fehlt die Typ-Angabe bei einer der Variablen, legt StarOffice Basic diese als Variant an. Im Beispiel

```
Dim MyVar1, MyVar2 As Integer
```

wird so aus `MyVar1` ein Variant und aus `MyVar2` ein Integer.

Die nachfolgenden Abschnitte listen auf, welche Variablentypen in StarOffice Basic zur Verfügung stehen, für welche Anwendungszwecke sie gedacht sind und wie sie sich deklarieren lassen.

Zeichenfolgen

Zeichenfolgen bilden zusammen mit den Zahlen die wichtigsten Basistypen von StarOffice Basic. Eine Zeichenfolge (String) besteht aus einer Kette aneinander gereihter Einzelzeichen. Intern speichert der Computer die Zeichenfolgen als Aneinanderreihung von Zahlen ab, wobei jede Zahl für ein bestimmtes Zeichen steht.

Vom ASCII-Zeichensatz zu Unicode

Welche Zahl welches Zeichen einer Zeichenfolge repräsentiert, legt ein so genannter Zeichensatz fest. Dahinter verbirgt sich eine Tabelle, die die Zahlen verschiedenen Zeichen beziehungsweise Symbolen zuordnet. Soll eines der Zeichen ausgegeben werden, so entscheidet der Computer anhand dieser Tabelle, welches Zeichen er auf dem Drucker oder Bildschirm darstellt.

ASCII-Zeichensatz

In der Anfangszeit der Personal Computer kam insbesondere der ASCII-Zeichensatz zum Einsatz (American Standard Code for Information Interchange). Er ordnet den Zahlen von 0 bis 127 die gängigen lateinischen Buchstaben, die arabischen Ziffern, einige Sonderzeichen (etwa Klammern, Punkt und Komma) sowie Steuerzeichen zu (zum Beispiel das Zeilenumbruchzeichen).

Der ASCII-Zeichensatz hat dazu beigetragen, den Datenaustausch zwischen verschiedenen Programmen und Systemen zu vereinfachen. Er kam jedoch schnell an seine Grenzen. So fehlen beispielsweise eine ganze Reihe der in Europa verwendeten Sonderzeichen wie â, ä und î. Ganz zu schweigen von einer Unterstützung anderer Schriftzeichen, etwa aus dem Kyrillischen.

ANSI-Zeichensatz

Mit Windows setzte Microsoft auf den ANSI-Zeichensatz, der nach und nach um zusätzliche Zeichen erweitert wurde. Auch hier gab es jedoch proprietäre Erweiterungen. So führte Microsoft zum Beispiel nachträglich das Zeichen 128 ein, welches das Euro-Symbol repräsentiert.

Codepages

Den lange überfälligen internationalen Standard lieferten die ISO-8859-Zeichensätze (*Codepages*). Die ersten 128 Zeichen der ISO-Zeichensätze entsprechen dem ASCII-Zeichensatz.

Der ISO-Standard führte eine Reihe verschiedener Zeichensätze (*Codepages*) ein, die jeweils auf dem betreffenden System eingestellt werden müssen.

Die Codepages boten in vielen Regionen der Welt die einzige Möglichkeit, die landesspezifischen Buchstaben darzustellen. Sie führten allerdings zu dem unschönen Nebeneffekt, dass ein und derselbe Zeichenwert in verschiedenen Ländern für unterschiedliche Buchstaben steht.

Unicode

1991 wurde das Unicode-Konsortium gegründet mit dem Ziel, die verschiedenen Zeichensätze zusammenzufassen und einen Standard zu schaffen, mit dem sich möglichst alle Schriftsprachen der Welt darstellen lassen.

Das Unicode-Konsortium entschied zunächst, die Länge eines Zeichens von einem Byte auf zwei Byte zu erhöhen. Diese Version 2.0 von Unicode wird mittlerweile von vielen Programmen unterstützt – dazu zählen auch StarOffice und StarOffice Basic.

In der Unicode-Version 3.0 vom September 1999 wurde der Zeichensatz nochmals erweitert. Ein Zeichen ist nun bis zu 4 Byte lang.

String-Variablen

Zeichenfolgen werden in StarOffice Basic in Form von String-Variablen abgelegt. Sie können Zeichenfolgen mit einer Länge von bis zu 65535 Zeichen enthalten. Intern legt StarOffice Basic für jedes Zeichen den zugehörigen Unicode-Wert ab. Der benötigte Arbeitsspeicher für eine String-Variable hängt davon ab, wie lang die Zeichenfolge ist.

Beispiel-Deklaration einer String-Variablen:

```
Dim Variable As String
```

Das Typ-Deklarationszeichen für String-Variablen lautet $, so dass folgende Deklaration der oben stehenden entspricht:

```
Dim Variable$
```

Achten Sie beim Portieren von VBA-Anwendungen darauf, dass diese die in StarOffice Basic existierende Maximallänge von Zeichenfolgen einhalten (65535 Zeichen).

Angabe expliziter Zeichenfolgen

Soll einer Zeichenfolgen-Variable eine explizite Zeichenfolge zugeordnet werden, so muss diese in doppelten, oben stehenden Anführungszeichen aufgeführt werden.

```
Dim MyString As String
MyString = "Das ist ein Test"
```

Falls die Zeichenfolge auf zwei Zeilen aufgeteilt werden muss, ist diese in zwei Zeichenfolgen aufzuteilen, die dann über ein Plus-Zeichen miteinander verkettet werden:

```
Dim MyString As String
MyString = "Diese Zeichenfolge ist so lang, dass sie " + _
           "auf zwei Zeilen aufgeteilt wurde."
```

Soll eine Zeichenfolge das Zeichen " enthalten, dann muss es an der betreffenden Stelle zweimal hintereinander aufgeführt werden:

```
Dim MyString As String
MyString = "ein ""-Anführungszeichen."    ' ergibt: ein "-Anführungszeichen
```

Zahlen

StarOffice Basic unterstützt fünf Basistypen zur Verarbeitung von Zahlen:

- Integer
- Long Integer
- Float
- Double
- Currency

Grob unterteilt man die Zahlen in ganzzahlige Variablentypen, zu denen die Typen *Integer* und *Long Integer* gehören und Fließkommatypen wie *Float* und *Double*, die ein Komma enthalten können.

Integer-Variablen

Variablen des Typs *Integer* können ganze Zahlen zwischen -32768 und 32767 aufnehmen. Beim Zuweisen einer Integer-Variablen an eine Fließkommazahl, wird diese auf die nächste ganze Zahl gemäß der kaufmännischen Rundungsregeln auf- beziehungsweise abgerundet. Berechnungen mit Integer-Variablen sind sehr schnell. Sie eignen sich unter anderem als Zählvariablen für Schleifen. Eine Integer-Variable belegt 2 Byte Arbeitsspeicher. Das Typ-Deklarationszeichen für Integer-Variablen lautet **%**.

Beispiel-Deklarationen einer Integer-Variablen:

```
Dim Variable As Integer
Dim Variable%
```

Long Integer-Variablen

Variablen des Typs *Long Integer* können ganze Zahlen zwischen -2147483648 und 2147483647 aufnehmen. Beim Zuweisen von Fließkommazahlen an eine Long Integer-Variable wird diese auf die nächste ganze Zahl gemäß der kaufmännischen Rundungsregeln auf- beziehungsweise abgerundet. Berechnungen mit Long Integer-Variablen sind sehr schnell. Sie eignen sich unter anderem als Zählvariablen für Schleifen mit vielen Durchläufen. Eine Long Integer-Variable belegt 4 Byte Arbeitsspeicher. Das Typ-Deklarationszeichen lautet **&**.

Beispiel-Deklarationen einer Long Integer-Variablen:

```
Dim Variable as Long
Dim Variable&
```

Single-Variablen

Variablen vom Typ *Single* sind Fließkommazahlen. Sie können positive und negative Werte zwischen 3,402823 x 10^{38} und 1,401298 x 10^{-45} aufnehmen. Eine Single-Variable belegt intern 4 Byte Arbeitsspeicher. Das Typ-Deklarationszeichen lautet !.

Ursprünglich wurden Single-Variablen konzipiert, um den Rechner bei der Ausführung mathematischer Operationen nicht so stark wie beim Umgang mit den genaueren Double-Variablen zu belasten. Eventuelle Geschwindigkeitsvorteile sind bei aktuellen Rechnersystemen allerdings nicht mehr feststellbar, so dass man aus Gründen der Genauigkeit im Regelfall auf Double-Variablen zurückgreifen wird.

Beispiel-Deklarationen einer Single-Variablen:

```
Dim Variable as Single
Dim Variable!
```

Double-Variablen

Variablen vom Typ *Double* sind Fließkommazahlen. Sie können positive und negative Werte zwischen 1,79769313486232 x 10^{308} und 4,94065645841247 x 10^{-324} aufnehmen. Double-Variablen sind geeignet für genaue Berechnungen. Eine Double-Variable belegt intern 8 Byte Arbeitsspeicher. Das Typ-Deklarationszeichen lautet #.

Beispiel-Deklarationen einer Double-Variablen:

```
Dim Variable As Double
Dim Variable#
```

Währungsvariablen (Currency)

Währungsvariablen (*Currency*-Variablen) unterscheiden sich von den anderen Variablen durch die interne Behandlung der Werte. Eine Currency-Variable belegt 8 Byte Arbeitsspeicher. Die Position des Kommas ist fest, weshalb Währungsvariablen auch nicht als Fließkommavariablen gelten. Es existieren stets genau vier Nachkommastellen. Für den Teil vor dem Komma stehen bis zu 15 Ziffern zur Verfügung. Die Werte einer Currency-Variable müssen zwischen -922337203685477,5808 und +922337203685477,5807 liegen. Das Typ-Deklarationszeichen für Currency-Variablen lautet @.

Währungsvariablen sind insbesondere für kaufmännische Kalkulationen gedacht, bei denen durch die Verwendung von Fließkomma-Zahlen unvorhersehbare Rundungsfehler entstehen können.

Beispiel-Deklaration einer Currency-Variablen:

```
Dim Variable As Currency
Dim Variable@
```

Angabe expliziter Zahlen

Zahlen können unterschiedlich dargestellt werden. So kommen je nach Sprache beispielsweise unterschiedliche Dezimaltrennzeichen zum Einsatz. Teilweise erfolgt die Angabe in der wissenschaftlichen Exponentialschreibweise. Und in einigen Fällen kommt anstelle des gängigen Dezimalsystems eine andere Basis zum Einsatz. Hier daher die Grundregeln für die Darstellungen von Zahlen innerhalb von StarOffice Basic:

Ganze Zahlen

Am einfachsten ist der Umgang mit ganzen Zahlen. Sie werden ohne Tausender-Trennzeichen innerhalb des Quelltextes aufgeführt:

```
Dim A As Integer
Dim B As Float

A = 1210
B = 2438
```

Den Zahlen kann sowohl ein Plus als auch ein Minus als Vorzeichen vorangestellt werden (mit oder ohne Leerzeichen dazwischen):

```
Dim A As Integer
Dim B As Float

A = + 121
B = - 243
```

Dezimalzahlen

Bei der Eingabe von Dezimalzahlen ist ein *Punkt* als Dezimaltrennzeichen zu verwenden. Mit dieser Festlegung wird sichergestellt, dass sich Quelltexte ohne Konvertierung von einem Land in ein anderes Land übertragen lassen.

```
Dim A As Integer
Dim B As Integer
Dim C As Float

A = 1223.53       ' wird gerundet
B = - 23446.46    ' wird gerundet
C = + 3532.76323
```

Auch bei Dezimalzahlen ist die Verwendung der Vorzeichen Plus und Minus gestattet (ebenfalls mit und ohne Leerzeichen).

Wird eine Kommazahl, wie im Beispiel zu sehen, einer Integer-Variablen zugewiesen, so rundet StarOffice Basic die Zahl auf den nächsten ganzzahligen Wert auf beziehungsweise ab.

Exponentialschreibweise

StarOffice Basic gestattet die Angabe von Zahlen in Exponentialschreibweise. Dies ermöglicht eine einfache Darstellung von sehr großen und sehr kleinen Zahlen. Die Zahl 1,5e-10 beispielsweise, steht für die Zahl $1,5 \times 10^{-10}$ (0,00000000015). Der Basiswert und der Exponent sind dabei durch den Buchstaben `e` (egal ob klein oder groß geschrieben) zu trennen. Der Exponent kann optional mit einem Plus als Vorzeichen aufgeführt werden.

Hier einige Beispiele richtiger und falscher Zahlangaben in Exponentialschreibweise:

```
Dim A As Double

A = 1.43E2        ' Korrekt
A = + 1.43E2      ' Korrekt (Leerzeichen zwischen Plus und Basiszahl)
A = - 1.43E2      ' Korrekt (Leerzeichen zwischen Minus und Basiszahl)
A = 1.43E-2       ' Korrekt (Negativer Exponent)

A = 1.43E -2      ' Fehlerhaft (Leerzeichen innerhalb der Zahl verboten)
A = 1,43E-2       ' Fehlerhaft (Komma als Dezimaltrennzeichen verboten)
A = 1.43E2.2      ' Fehlerhaft (Exponent muss ganzzahlig sein)
```

Bei der ersten und der dritten fehlerhaften Darstellung ist darauf zu achten, dass in diesen Fällen kein Fehler generiert wird. Sie führen jedoch zu falschen Werten. Der Ausdruck

```
A = 1.43E -2
```

wird interpretiert als 1,43 minus 2, was dem Wert -0,57 entspricht. Gemeint war jedoch der Wert $1,43 * 10^{-2}$ (entspricht 0,0143). Bei dem Wert

```
A = 1.43E2.2
```

ignoriert StarOffice Basic den Nachkomma-Teil des Exponenten und interpretiert den Ausdruck als

```
A = 1.43E2
```

Hexadezimalwerte

In einigen Fällen ist es sinnvoll, Zahlen im Hexadezimal- anstatt des üblichen Dezimalsystems darzustellen. Das Hexadezimalsystem (16er System) hat den Vorteil, dass zwei Ziffern jeweils exakt einem Byte entsprechen, was einen maschinennahen Umgang mit Zahlen gestattet. Als Ziffern kommen beim Hexadezimalsystem die Ziffern 0 bis 9 sowie die Buchstaben A bis F zum Einsatz. Ein A steht für die Dezimalzahl 10, ein F für die Dezimalzahl 15. StarOffice Basic gestattet die Eingabe ganzzahliger Hexadezimalwerte. Ihnen ist die Zeichenfolge `&H` voran zu stellen, damit sie als Hexadezimal erkannt werden.

```
Dim A As Long

A = &HFF    ' Hexadezimalwert FF, entspricht dem Dezimalwert 255
A = &H10    ' Hexadezimalwert 10, entspricht dem Dezimalwert 16
```

Oktalwerte

Neben dem Hexadezimalsystem versteht StarOffice Basic das Oktalsystem (8er-System), das mit den Ziffern 0 bis 7 arbeitet. StarOffice Basic gestattet die Angabe ganzzahliger Oktalwerte durch das Voranstellen der Ziffernfolge `&O`. Hier einige Beispiele:

```
Dim A As Longer

A = &O77    ' Oktalwert 77, entspricht dem Dezimalwert 63
A = &O10    ' Oktalwert 10, entspricht dem Dezimalwert 8
```

Wahr und Falsch

Boolesche Variablen

Booleschen Variablen können genau zwei Werte enthalten: `True` (Wahr) und `False` (Falsch). Sie eignen sich für binäre Angaben, die genau einen der genannten Zustände einnehmen können. Ein Boolescher Wert wird intern in einem 2-Byte-Integerwert gespeichert. Hat dieser den Wert 0, so entspricht dies dem Booleschen Wert `False`. Alle anderen Werte entsprechen dem Booleschen Wert `True`. Für Boolesche Variablen existiert kein Typ-Deklarationszeichen. Die Deklaration ist ausschließlich über den Zusatz *As Boolean* möglich.

Beispiel-Deklaration einer Booleschen Variablen:

```
Dim Variable As Boolean
```

Datums- und Zeitangaben

Date-Variablen

Date-Variablen können Datums- und Zeitwerte enthalten. Bei der Speicherung von Datumswerten verwendet StarOffice Basic ein internes Format, das Vergleiche und mathematische Operationen erlaubt. Für Date-Variablen existiert kein Typ-Deklarationszeichen. Die Deklaration ist ausschließlich über den Zusatz *As Date* möglich.

Beispiel-Deklaration einer Date-Variablen:

```
Dim Variable As Date
```

Datenfelder

Neben einfachen Variablen (*Skalare*) unterstützt StarOffice Basic auch Datenfelder (*Arrays*). Ein Datenfeld enthält mehrere Variablen, die über einen Index adressiert werden.

Einfache Arrays

Die Deklaration eines Arrays ähnelt der einer einfachen Variablendeklaration. Es schließt sich jedoch eine Klammer an, die Angaben zur Anzahl der Elemente enthält. Der Ausdruck

```
Dim MyArray(3)
```

deklariert ein Array mit 4 Variablen vom Datentyp Variant, nämlich `MyArray(0)`, `MyArray(1)`, `MyArray(2)` und `MyArray(3)`.

Neben dem Universal-Datentyp Variant sind auch typspezifische Datenfeld-Deklarationen möglich. Die Zeile

```
Dim MyInteger(3) As Integer
```

beispielsweise deklariert ein Array mit 4 Integer-Variablen.

In den vorstehenden Beispielen beginnt der Index für das Array stets mit dem Standard-Startwert 0. Alternativ ist es auch möglich, bei der Datenfeld-Deklaration einen Gültigkeitsbereich mit Start- und Endwert anzugeben. So deklariert der Beispielausdruck

```
Dim MyInteger(5 To 10)
```

ein Datenfeld mit 6 Integer-Werten, die sich über die Indizes 5 bis 10 ansprechen lassen.

Bei den Indizes muss es sich nicht zwangsläufig um positive Werte handeln. Das folgende Beispiel zeigt eine ebenfalls korrekte Deklaration mit negativen Datenfeld-Grenzen.

```
Dim MyInteger(-10 To -5)
```

Es deklariert ein Integer-Datenfeld mit 6 Werten, die mit den Indizes -10 bis -5 adressierbar sind.

Beim Festlegen von Datenfeld-Indizes gibt StarOffice Basic dem Programmierer freie Hand. Er muss lediglich drei Grenzwerte beachten:

- der kleinste mögliche Index beträgt -32768.
- der größte mögliche Index beträgt 32767.
- die maximale Anzahl von Elementen (innerhalb einer Datenfeld-Dimension) beträgt 16368.

In VBA gelten teilweise andere Grenzwerte für Datenfeld-Indizes und die maximal je Dimension mögliche Anzahl der Elemente. Die dort gültigen Werte sind der jeweiligen VBA-Dokumentation zu entnehmen.

Vorgabewert für Startindex

Wie im vorigen Absatz gezeigt, beginnt der Start-Index eines Datenfeldes im Normalfall mit dem Wert 0. Alternativ besteht die Möglichkeit, den Startindex für sämtliche Datenfeld-Deklarationen über den Aufruf

```
Option Base 1
```

auf den Wert 1 umstellen. Der betreffende Aufruf sollte sich im Kopf des betreffenden Modules befinden und gilt dann für alle darin enthaltenen Array-Deklarationen. Keine Auswirkungen hat die Definition jedoch auf die über das StarOffice API definierten UNO-Sequenzen, deren Index in *jedem Falle* mit 0 beginnt. Aus Gründen der Übersichtlichkeit sollte daher auf den Einsatz von Option Base 1 nach Möglichkeit verzichtet werden.

Bei der Verwendung von Option Base 1 in StarOffice Basic ist zu beachten, dass dies keinen Einfluss auf die Anzahl der Elemente innerhalb des Arrays hat. Lediglich der Startindex verschiebt sich. Die Deklaration

```
Option Base 1
' ...
Dim MyInteger(3)
```

erzeugt 4 Integer-Variablen, die sich über die Ausdrücke `MyInteger(1)`, `MyInteger(2)`, `MyInteger(3)` und `MyInteger(4)` ansprechen lassen. Diesbezüglich unterscheidet sich StarOffice Basic von anderen Basic-Programmiersprachen wie VBA:

> Bei StarOffice Basic hat der Ausdruck Option Base 1 im Gegensatz zu VBA keinen Einfluss auf die Anzahl der Elemente eines Arrays. Lediglich der Startindex verschiebt sich in StarOffice Basic. Während die Deklaration `MyInteger(3)` in VBA drei Integer-Werte mit den Indizes 1 bis 3 erzeugt, generiert die gleiche Deklaration in StarOffice Basic vier Integer-Werte mit den Indizes 1 bis 4.

Mehrdimensionale Datenfelder

Neben eindimensionalen Datenfeldern unterstützt StarOffice Basic auch den Umgang mit mehrdimensionalen Datenfeldern. Die betreffenden Dimensionen werden durch Kommata voneinander getrennt. Das Beispiel

```
Dim MyIntArray(5, 5)
```

definiert ein Integer-Array mit zwei Dimensionen mit jeweils 6 Indizes (adressierbar über die Indizes 0 bis 5). Das gesamte Array kann somit insgesamt 6 × 6 = 36 Integer-Werte aufnehmen.

Theoretisch lassen sich in StarOffice Basic Arrays mit mehreren hundert Dimensionen definieren. In der Praxis begrenzt der verfügbare Arbeitsspeicher die Anzahl der Dimensionen.

Dynamische Größenänderungen von Datenfeldern

Die vorstehenden Beispiele basieren auf Datenfeldern mit vorgegebener Größe. Es gibt jedoch Anwendungen, in denen die Festlegung auf eine bestimmte Anzahl von Elementen zu unflexibel ist. Man denke sich beispielsweise eine Funktion, die sämtliche Begriffe eines Textes suchen soll, die mit dem Buchstaben A beginnen. Da zunächst unklar ist, wie viele entsprechende Begriffe in dem Text vorhanden sind, muss man die Feldgrenzen nachträglich verändern können. Der zugehörige Aufruf lautet:

```
ReDim MyArray(10)
```

Das Beispiel ändert das Array so, dass es 11 Werte aufnehmen kann.

VBA unterscheidet im Gegensatz zu StarOffice Basic zwischen statischen und dynamischen Arrays. Dynamische Arrays müssen in VBA über einen Aufruf Dim MyArray() dimensioniert werden. Nur in diesem Fall ist eine nachträgliche Größenänderung des Arrays möglich. In StarOffice Basic lassen sich hingegen alle Arrays über ReDim ändern.

Die Größe eines Arrays lässt sich beliebig oft anpassen:

```
Dim MyArray(4) As Integer           ' Deklaration mit fünf Elementen

' ...

ReDim MyArray(10) As Integer        ' Vergrößerung auf 11 Elemente

' ...

ReDim MyArray(20) As Integer        ' Vergrößerung auf 21 Elemente
```

Bei der Größenanpassung eines Arrays kann man zur Angabe von Indizes alle oben erläuterten Möglichkeiten verwenden. Dies umfasst die Möglichkeiten zur Deklaration mehrdimensionaler Datenfelder und zur Angabe expliziter Start- und Endwerte. Mit einer Größenänderung des Datenfeldes geht sein alter Inhalt *komplett verloren*. Wer die alten Werte erhalten möchte, muss bei der Größenänderung ein zusätzliches Schlüsselwort namens `Preserve` angeben:

```
Dim MyArray(10) As Integer                      ' Festlegen der initialen
                                                ' Größe

' ...

ReDim Preserve MyArray(20)    As Integer        ' Vergrößerung des
                                                ' Datenfeldes, bei
                                                ' gleichbleibendem Inhalt
```

Es sorgt dafür, dass die alten Werte des Arrays erhalten bleiben. Dem Einsatz von `Preserve` sind dabei kaum Grenzen gesetzt. Lediglich die Anzahl der Dimensionen und der Typ der Variablen müssen bei der Verwendung von `Preserve` gleich bleiben.

Während sich in VBA bei Verwendung von `Preserve` nur die obere Grenze der letzten Dimension eines Datenfeldes verändern lässt, erlaubt StarOffice Basic in Verbindung mit `Preserve` auch die Änderung anderer Dimensionen sowie der unteren Datenfeld-Grenzen.

Bei einer `ReDim`-Anweisung mit `Preserve` muss der gleiche Datentyp wie bei der ursprünglichen Datenfeld-Deklaration verwendet werden. Ansonsten erzeugt StarOffice Basic eine Fehlermeldung.

Gültigkeitsbereiche und Lebensdauern von Variablen

Eine Variable hat in StarOffice Basic nur eine begrenzte Lebensdauer und einen begrenzten Gültigkeitsbereich, von dem aus sie für andere Programmfragmente sichtbar und verwendbar ist. Wie lange die Variable tatsächlich erhalten bleibt und von wo aus sie verfügbar ist, hängt von ihrere definierten Position und ihrem Typ ab.

Lokale Variablen

Lokale Variablen stehen innerhalb einer Funktion oder Prozedur:

```
Sub Test
      Dim MyInteger As Integer

      ' ...
End Sub
```

Sie sind so lange gültig, wie die zugehörige Funktion oder Prozedur ausgeführt wird. Nach dem Ende der Funktion setzt StarOffice Basic den Inhalt der Variablen auf einen `Null`-Wert zurück. Eventuelle errechnete Ergebnisse gehen damit verloren.

Wer die alten Werte erhalten möchte, muss die Variable als `Static` definieren:

```
Sub Test
      Static MyInteger As Integer

      ' ...

End Sub
```

Im Gegensatz zu VBA achtet StarOffice Basic darauf, dass der Name einer lokalen Variablen nicht gleichzeitig für eine Globale beziehungsweise Private Variable im Kopf des Modules verwendet wird. Beim Portieren von VBA-Anwendungen nach StarOffice Basic sind etwaige doppelte Variablennamen gegebenenfalls zu ändern.

Öffentliche Variablen

Öffentliche Variablen werden im Kopfbereich eines Modules definiert. Sie stehen allen Modulen der betreffenden Bibliothek zur Verfügung:

Modul A:

```
Dim A As Integer

Sub Test
      Flip
      Flop
End Sub

Sub Flip
      A = A + 1
End Sub
```

Modul B:

```
Sub Flop
      A = A - 1
End Sub
```

So enthält die Variable `A` in dem vorstehenden Beispiel nach dem Aufruf der Funktion `Test` den gleichen Wert wie vor dem Aufruf. Die Funktion `Flip` zählt zunächst den Wert 1 hinzu, die Funktion `Flop` zieht diesen Wert dann wieder ab. Beide Funktionen greifen dabei auf ein und dieselbe Variable `A` zurück, da es sich durch die Deklaration mit Hilfe des Schlüsselwortes `Dim` um eine öffentliche Variable handelt.

Um dies zu unterstreichen, lässt sich anstelle des Schlüsselwortes `Dim` auch das Schlüsselwort `Public` bei der Deklaration verwenden:

```
Public A As Integer
```

Die Lebensdauer öffentlicher Variablen ist beschränkt auf die Ausführung des zugehörigen Makros. Nach dessen Ablauf werden die Variablen automatisch zurück gesetzt .

Globale Variablen

Globale Variablen entsprechen von Ihrer Funktionsweise her weitgehend den öffentlichen Variablen. Ihre Werte bleiben jedoch auch dann erhalten, wenn das zugehörige Makro bereits beendet wurde. Die Deklaration von globalen Variablen erfolgt im Kopfbereich eines Moduls über das Schlüsselwort `Global`:

```
Global A As Integer
```

Private Variablen

StarOffice Basic-Module können neben `Public`-Variablen auch `Private`-Variablen enthalten. Wie das Beispiel

```
Private MyInteger As Integer
```

zeigt, werden diese durch das Schlüsselwort `Private` definiert. `Private`-Variablen stehen ausschließlich in dem Modul zur Verfügung, in dem sie definiert wurden. Enthalten mehrere Module eine `Private`-Variable mit ein und demselben Namen, so erzeugt StarOffice Basic dafür dennoch zwei verschiedene Variablen. Dies zeigt das nachfolgende Beispiel. Es besteht aus den Modulen `A` und `B`, die jeweils eine `Private`-Variable `C` definieren. Die Funktion `Test` setzt zunächst die `Private`-Variable in Modul `A` und dann die `Private`-Variable in Modul `B`.

Modul A:

```
Private C As Integer

Sub Test
      SetModuleA              ' Setzt die Variable C aus Modul A
      SetModuleB              ' Setzt die Variable C aus Modul B

      ShowVarA                ' Zeigt die Variable C aus Modul A an (= 10)
      ShowVarB                ' Zeigt die Variable C aus Modul B an (= 20)
End Sub

Sub SetModuleA
      A = 10
End Sub

Sub ShowVarA
      MsgBox C                ' Zeigt die Variable C aus Modul A an.
End Sub
```

Modul B:

```
Private C As Integer

Sub SetModuleB
      A = 20
End Sub

Sub ShowVarB
      MsgBox C                ' Zeigt die Variable C aus Modul B an.
End Sub
```

Konstanten

Die Deklaration einer Konstante erfolgt über das Schlüsselwort `Const`:

```
Const A = 10
```

Für den Fall, dass die Konstante einen bestimmten vorgegebenen Typ annehmen soll, lässt sich dieser ebenfalls bei der Deklaration angeben:

```
Const B As Double = 10
```

Operatoren

StarOffice Basic versteht die gängigen mathematischen, logischen und vergleichenden Operatoren.

Mathematische Operatoren

Die mathematischen Operatoren sind auf sämtliche Zahlen anwendbar. Der Operator + kann darüber hinaus zur Verkettung von Zeichenfolgen verwendet werden.

`+` Addition von Zahlen und Datumswerten, Verketten von Zeichenfolgen

`-` Subtraktion von Zahlen und Datumswerten

`*` Multiplikation von Zahlen

`/` Division von Zahlen

`\` Division von Zahlen mit ganzzahligem Ergebnis (wird kaufmännisch gerundet)

`^` Potenzieren von Zahlen

`MOD` Modulo-Operation (Berechnung des Restes einer Division)

Logische Operatoren

Logische Operatoren gestatten eine Verknüpfung nach den Regeln der Booleschen Algebra. Werden sie auf Boolesche Werte angewendet, so liefert die Verknüpfung direkt das gewünschte Ergebnis. Beim Einsatz im Zusammenhang mit Integer- und Long Integer-Werten erfolgt eine bitweise Verknüpfung.

`AND` Und-Verknüpfung

`OR` Oder-Verknüpfung

`XOR` Exklusive Oder-Verknüpfung

`NOT` Negation

`EQV` Equivalent-Test (beide Teile `True` oder `False`)

`IMP` Implikation (falls erster Ausdruck richtig, muss auch zweiter richtig sein)

Vergleichsoperatoren

Die Vergleichsoperatoren können auf sämtliche elementaren Variablentypen (Zahlen, Datumsangaben, Zeichenfolgen und Boolesche Werte) angewendet werden.

=	Gleichheit von Zahlen, Datumswerten und Zeichenfolgen
<>	Ungleichheit von Zahlen, Datumswerten und Zeichenfolgen
>	Größer-Prüfung für Zahlen, Datumswerte und Zeichenfolgen
>=	Größer-Gleich-Prüfung für Zahlen, Datumswerte und Zeichenfolgen
<	Kleiner-Prüfung für Zahlen, Datumswerte und Zeichenfolgen
<=	Kleiner-Gleich-Prüfung für Zahlen, Datumswerte und Zeichenfolgen

Der in VBA vorhandene Vergleichsoperator `Like` wird von StarOffice Basic nicht unterstützt.

Verzweigungen

Verzweigungen erlauben es, einen Programmbereich nur dann auszuführen, wenn eine bestimmte Bedingung erfüllt ist.

If...Then...Else

Die bekannteste und am meisten verwendete Verzweigung ist die `If`-Anweisung. Das Beispiel

```
If A > 3 Then
      B = 2
End If
```

zeigt eine einfache `If`-Anweisung. Die Zuweisung `B = 2` wird ausgeführt, wenn die Variable `A` größer als der Wert drei ist. Neben dem einfachen `If`-Befehl existiert eine weitere Variante des Befehls, bei der auch ein alternativer Ausführungszweig angegeben werden kann. Im Beispiel

```
If A > 3 Then
      B = 2
Else
      B = 0
End If
```

erhält die Variable `B` den Wert 2 zugewiesen, wenn `A` größer als 3 ist. Sonst erhält `B` den Wert 0.

Für komplexere Abfragen gestattet StarOffice Basic das Kaskadieren mehrerer `If`-Anweisungen:

```
If A = 0 Then
      B = 0
ElseIf A < 3 Then
      B = 1
Else
      B = 2
End If
```

Falls die Variable `A` den Wert Null hat, erhält `B` ebenfalls den Wert `0`. Ist `A` kleiner als `3` (aber ungleich Null), erhält `B` den Wert `1`. In allen übrigen Fällen (`A` ist größer oder gleich `3`) erhält `B` den Wert `2`.

Select...Case

Die `Select....Case`-Anweisung ist eine Alternative zur oben aufgeführten kaskadierten `If`-Anweisung. Sie wird eingesetzt, wenn ein Wert auf verschiedene Bedingungen hin überprüft werden soll:

```
Select Case DayOfWeek
Case 1:
      NameOfWeekday = "Sunday"
Case 2:
      NameOfWeekday = "Monday"
Case 3:
      NameOfWeekday = "Tuesday"
Case 4:
      NameOfWeekday = "Wednesday"
Case 5:
      NameOfWeekday = "Thursday"
Case 6:
      NameOfWeekday = "Friday"
Case 7:
      NameOfWeekday = "Saturday"
End Select
```

Das Beispiel ermittelt den Namen eines Wochentages für dessen Nummer. Enthält die Variable `DayOfWeek` den Wert `1`, so wird der Variable `NameOfWeekday` der Wert `Sunday` zugewiesen, enthält `DayOfWeek` den Wert `2`, so wird NameOf`Weekday` der Wert `Monday` zugewiesen usw.

Der `Select`-Befehl ist dabei keineswegs auf einfache 1:1-Zuordnungen beschränkt. So ist es möglich, auch Vergleichsoperatoren oder Listen von Ausdrücken innerhalb eines `Case`-Zweiges anzugeben. Das nachfolgende Beispiel listet die wichtigsten Syntax-Varianten auf:

```
Select Case Var
Case 1 To 5

      ' ... Var liegt zwischen den Zahlen 1 und 5

Case 6, 7, 8

      ' ... Var ist 6, 7 oder 8

Case Var > 8 And Var < 11

      ' ... Var ist größer als 8 und kleiner als 11

Case Else

      ' ... alle übrigen Fälle

End Select
```

Schleifen

Schleifen gestatten es, einen Block mit Anweisungen mehrmals hintereinander auszuführen. Man unterscheidet zwischen Schleifen mit einer vorgegebenen Zahl an Durchläufen und Schleifen mit einer offenen Zahl an Durchläufen.

For...Next

Die `For`-Schleife ist eine Schleife mit einer festen Zahl an Durchläufen. Der Schleifenkopf legt fest, wie oft die Schleife durchschritten wird. Eine `For`-Schleife benötigt eine Zählvariable, in der StarOffice Basic vermerkt, um welchen Durchlauf es sich handelt. Im Beispiel

```
Dim I

For I = 1 To 10

      ' ... Innenteil der Schleife

Next I
```

lautet die Zählvariable `I`. Die Zuweisung `I = 1` legt fest, dass `I` beim ersten Durchlauf den Wert `1` bekommt. Der Ausdruck `To 10` wiederum bestimmt, dass die Schleife so lange durchlaufen wird, bis `I` den Wert `10` hat. Da StarOffice Basic `I` bei jedem Durchlauf um den Wert `1` erhöht, wird der innere Teil der Schleife damit exakt 10 mal ausgeführt.

In einigen Fällen macht es Sinn, mit einer anderen Schrittweite als 1 zu arbeiten. Diese wird gegebenenfalls durch den Zusatz `Step` innerhalb des Schleifenkopfes angegeben. Im Beispiel

```
Dim I

For I = 1 To 10 Step 0.5

      ' ... Innenteil der Schleife

Next I
```

wird die Schleife bei jedem Durchlauf um 0,5 statt um 1 erhöht. Die Anzahl der Schleifendurchläufe beträgt damit 19.

Auch negative Schrittweiten sind erlaubt, wie folgendes Beispiel zeigt:

```
Dim I

For I = 10 To 1 Step -1

      ' ... Innenteil der Schleife

Next I
```

In diesem Fall beginnt die Zählvariable I bei 10 und reduziert sich in ganzzahligen Schritten auf 1.

Die Anweisung `Exit For` gestattet den vorzeitigen Abbruch einer `For`-Schleife. Das Beispiel

```
Dim I

For I = 1 To 10

      If I = 5 Then
            Exit For
      End If

      ' ... Innenteil der Schleife

Next I
```

zeigt eine `For`-Schleife, die beim fünften Durchlauf durch einen expliziten Aufruf von `Exit For` abbricht.

Die in VBA vorhandene Schleifenvariante `For Each...Next` wird in StarOffice Basic nicht unterstützt.

Do...Loop

Mit `Do...Loop` bietet StarOffice Basic eine Schleife, die nicht an eine feste Anzahl von Durchläufen gebunden ist. `Do...Loop` wird so lange ausgeführt, bis eine bestimmte Bedingung erfüllt beziehungsweise nicht mehr erfüllt ist. Von der Schleife existieren vier Varianten, die nachfolgend vorgestellt werden. Der Ausdruck `A > 10` steht dabei stellvertretend für eine beliebige Bedingung.

1. Die `Do While...Loop`-Variante

   ```
   Do While A > 10
       ' ... Schleifenrumpf
   Loop
   ```

   prüft vor jedem Durchlauf, ob die Bedingung noch erfüllt ist und führt nur dann den Schleifenrumpf aus.

2. Die `Do Until...Loop`-Variante

   ```
   Do Until A > 10
         ' ... Schleifenrumpf
   Loop
   ```

   prüft ebenfalls zu Beginn der Schleife, führt den Schleifenrumpf jedoch so lange aus, bis die Bedingung *nicht mehr* erfüllt ist.

3. Die `Do...Loop While`-Variante

   ```
   Do
         ' ... Schleifenrumpf
   Loop While A > 10
   ```

   prüft die Bedingung erst nach dem ersten Durchlauf der Schleife und bricht ab, falls diese *erfüllt* ist.

4. Die `Do...Loop Until`-Variante

   ```
   Do
         ' ... Schleifenrumpf
   Loop Until A > 10
   ```

   prüft ebenfalls erst nach dem ersten Durchlauf ihre Bedingung, führt die Schleife jedoch so lange aus, bis die Bedingung *nicht mehr* erfüllt ist.

Analog zur `For...Next`-Schleife bietet auch die `Do...Loop`-Schleife einen Abbruchbefehl an, der dafür sorgt, dass die Schleife während eines Durchlaufs abgebrochen wird. Der betreffende Befehl lautet `Exit Do` und kann an jeder Stelle innerhalb der Schleife aufgeführt werden.

```
Do
      If A = 4 Then
```

```
            Exit Do
      End If

      ' ... Schleifenrumpf
While A > 10
```

Programmierbeispiel: Sortierung mit verschachtelten Schleifen

Die Anwendungsmöglichkeiten für Schleifen sind vielfältig. In vielen Fällen dienen Sie dazu, Listen zu durchsuchen oder Werte auszugeben. Und teilweise übernehmen sie komplexe mathematische Aufgaben. Das nachfolgende Beispiel zeigt einen Algorithmus, der mit Hilfe von zwei verschachtelten Schleifen eine Liste mit Namen sortiert.

```
Sub Sort
      Dim Entry(1 To 10) As String
      Dim Count As Integer
      Dim Count2 As Integer
      Dim Temp As String

      Entry(1) = "Patty"
      Entry(2) = "Kurt"
      Entry(3) = "Thomas"
      Entry(4) = "Michael"
      Entry(5) = "David"
      Entry(6) = "Cathy"
      Entry(7) = "Susie"
      Entry(8) = "Edward"
      Entry(9) = "Christine"
      Entry(10) = "Jerry"

      For Count = 1 To 10
            For Count2 = Count + 1 To 10
                  If Entry(Count) > Entry(Count2) Then
                        Temp = Entry(Count)
                        Entry(Count) = Entry(Count2)
                        Entry(Count2) = Temp
                  End If
            Next Count2
      Next Count

      For Count = 1 To 10
            Print Entry(Count)
      Next Count
End Sub
```

Er durchläuft das Array mit den zu sortierenden Werten mehrmals hintereinander und vertauscht diese paarweise so lange, bis sie schließlich aufsteigend sortiert sind. Wie Blasen wandern die Variablen so nach und nach an ihre richtige Position. Der Algorithmus wird daher auch als *Bubble Sort* bezeichnet.

Prozeduren und Funktionen

Funktionen und Prozeduren sind der Dreh- und Angelpunkt bei der Strukturierung eines Programmes. Sie liefern das Rüstzeug zur Gliederung eines komplexen Problems in verschiedene Teilaufgaben.

Prozeduren

Eine *Prozedur* führt eine Aktion aus, ohne einen expliziten Wert zu liefern. Ihre Syntax lautet

```
Sub Test

      ' ... hier steht der eigentliche Code der Prozedur

End Sub
```

Das Beispiel definiert eine Prozedur namens `Test`, die eine Liste mit Anweisungen umfasst und an jeder beliebigen Stelle des Programmcodes aufgerufen werden kann. Der Aufruf erfolgt durch Aufführung des Prozedurnamens an der betreffenden Programmstelle:

```
Test
```

Funktionen

Eine *Funktion* fasst wie eine Prozedur einen auszuführenden Programmblock als eine logische Einheit zusammen, liefert im Gegensatz zur Prozedur jedoch einen Rückgabewert.

```
Function Test

      ' ... hier steht der eigentliche Code der Funktion

      Test = 123
End Function
```

Die Zuordnung des Rückgabewertes erfolgt durch eine einfache Zuweisung. Die Zuweisung muss nicht unbedingt am Ende der Funktion stehen. Sie kann zu jedem beliebigen Zeitpunkt erfolgen.

Die oben stehende Funktion lässt sich wie folgt aufrufen:

```
Dim A

A = Test
```

Der Code definiert eine Variable `A` und weist ihr das Ergebnis der Funktion `Test` zu.

Innerhalb der Funktion kann der Rückgabewert mehrfach überschrieben werden. Wie bei einer klassischen Variablenzuweisung liefert die Funktion in diesem Falle den Wert zurück, der ihr zuletzt zugewiesen wurde.

```
Function Test

      Test = 12

      ' ...

      Test = 123

End Function
```

Im Beispiel lautet der Rückgabewert der Funktion `123`, da der zuvor zugewiesene Wert `12` mit diesem Wert überschrieben wird.

Unterbleibt eine Zuweisung, so liefert die Funktion einen `Null`-Wert zurück (Zahl 0 für numerische Werte, leere Zeichenfolge für Strings).

Der Rückgabewert einer Funktion kann einen beliebigen Typ besitzen. Die Deklaration des Typs erfolgt analog zu einer klassischen Variablen-Deklaration:

```
Function Test As Integer

      ' ... hier steht der eigentliche Code der Funktion

End Function
```

Wenn die Angabe eines expliziten Wertes unterbleibt, bekommt der Rückgabewert den Typ Variant.

Prozeduren und Funktionen vorzeitig abbrechen

Insbesondere im Rahmen einer Fehlerbehandlung ist es teilweise notwendig, eine Prozedur oder Funktion vorzeitig abzubrechen. StarOffice Basic stellt dafür die Anweisungen `Exit Sub` und `Exit Function` zur Verfügung. Sie beenden die Ausführung und sorgen für einen Rücksprung zu der Stelle, an der die Prozedur beziehungsweise die Funktion aufgerufen wurde.

Das folgende Beispiel zeigt eine Prozedur, die ihre Ausführung abbricht, falls die Variable `ErrorOccured` den Wert `True` besitzt.

```
Sub Test
      Dim ErrorOccured As Boolean

      ' ...

      If ErrorOccured Then
            Exit Sub
      End If

      ' ...

End Sub
```

Parameterübergabe

Funktionen und Prozeduren können einen oder mehrere Parameter entgegen nehmen. Zwingend benötigte Parameter stehen in Klammern hinter dem Funktions- oder Prozedurnamen. Das Beispiel

```
Sub Test (A As Integer, B As String)
End Sub
```

definiert eine Prozedur, die einen Integer-Wert `A` und eine Zeichenfolge `B` als Parameter erwartet.

Die Übergabe von Parametern erfolgt in StarOffice Basic normalerweise als *Referenz*. Änderungen der Variablen bleiben beim Verlassen der Prozedur oder Funktion erhalten. Was das bedeutet, veranschaulicht das folgende Beispiel:

```
Sub Test
      Dim A As Integer
      A = 10
```

```
      ChangeValue(A)
      ' Der Parameter A hat nun den Wert 20
End Sub
Sub ChangeValue(TheValue As Integer)
       TheValue = 20
End Sub
```

Der in der Funktion `Test` definierte Wert `A` wird als Parameter in die Funktion `ChangeValue` übergeben. Dort wird er unter dem Namen `TheValue` auf den Wert 20 gesetzt. Diese Änderung bleibt beim Verlassen der Prozedur erhalten. Der Parameter `A` hat damit nach Beendigung von `ChangeValue` den Wert 20.

Für den Fall, dass sich das Ändern eines Parameters nicht auf den übergeordneten Bereich auswirken soll, lässt sich dieser *als Wert* übergeben. Der Variablen-Deklaration im Funktionskopf wird dazu das Schlüsselwort `ByVal` vorgestellt.

Tauscht man im obigen Beispiel die Funktion `ChangeValue` durch die Funktion

```
Sub ChangeValue(ByVal TheValue As Integer)
      TheValue = 20
End Sub
```

aus, so bleibt die übergeordnete Variable `A` von der Änderung unberührt. Er hat nach dem Aufruf der Funktion `ChangeValue` weiterhin den Wert `10`.

Das Verhalten bei der Übergabe von Parametern an Prozeduren und Funktionen ist in StarOffice Basic und VBA weitgehend identisch. Standardmäßig erfolgt die Parameterübergabe*per Referenz*. Über das Schlüsselwort `ByVal` ist es möglich, die Parameter alternativ *per Wert* zu übergeben. VBA bietet darüber hinaus ein weiteres Schlüsselwort `ByRef`, mit dem sich eine Übergabe als Referenz erzwingen lässt. StarOffice Basic unterstützt dieses Schlüsselwort nicht, da das damit erzielbare Verhalten dem Standard in StarOffice Basic entspricht.

Funktionen und Prozeduren sind in StarOffice Basic prinzipiell `Public`. Die in VBA vorhandenen Schlüsselwörter `Public` und `Private` werden in StarOffice Basic nicht unterstützt.

Optionale Parameter

Funktionen und Prozeduren lassen sich nur dann aufrufen, wenn beim Aufruf alle erforderlichen Parameter übergeben werden.

Insbesondere bei Funktionen mit langen Parameterlisten erfordert dies häufig aufwändige Tipparbeit. StarOffice Basic bietet daher die Möglichkeit, Parameter als *optional* zu definieren. Fehlen die betreffenden Werte in einem Aufruf, so übergibt StarOffice Basic eine Leervariable. In dem Beispiel

```
Sub Test(A As Integer, Optional B As Integer)

End Sub
```

ist der Parameter `A` obligatorisch, der Parameter `B` hingegen optional.

Die Funktion `IsMissing` prüft, ob ein Parameter übergeben oder ausgelassen wurde.

```
Sub Test(A As Integer, Optional B As Integer)
      Dim B_Local As Integer

      ' Prüfung, ob Parameter B tatsächlich vorhanden
      If Not IsMissing (B) Then
            B_Local = B       ' Parameter B vorhanden
      Else
            B_Local = 0       ' Parameter B fehlt -> Vorgabewert 0
      End If

      ' ... Start der eigentlichen Funktion

End Sub
```

Das Beispiel testet zunächst ob der Parameter `B` übergeben wurde und überträgt diesen Wert gegebenenfalls in die interne Variable `B_Local`. Fehlt der betreffende Parameter, so wird anstelle des übergebenen Parameters ein Vorgabewert (hier der Wert `0`) in `B_ Local` abgelegt.

Die in VBA vorhandene Möglichkeit zur Definition von Vorgabewerten für optionale Parameter wird in StarOffice Basic nicht unterstützt.

Das in VBA vorhandene Schlüsselwort `ParamArray` wird von StarOffice Basic nicht unterstützt.

Rekursion

Eine rekursive Prozedur oder Funktion kann sich selbst aufrufen bis sie feststellt, dass eine Grundbedingung erfüllt ist. Wenn die Funktion mit der Grundbedingung aufgerufen wird, wird ein Ergebnis zurückgegeben.

Im folgenden Beispiel wird eine rekursive Funktion eingesetzt, um die Fakultäten der Zahlen `42`, `-42` und `3,14` zu berechnen:

```
Sub Main
   Msgbox CalculateFactorial(  42 )    ' zeigt 1,40500611775288E+51 an
   Msgbox CalculateFactorial( -42 )    ' zeigt "Invalid number for factorial!" an
   Msgbox CalculateFactorial( 3.14 )   ' zeigt "Invalid number for factorial!" an
End Sub

Function CalculateFactorial( Number )
   If Number < 0 Or Number <> Int( Number ) Then
       CalculateFactorial = "Invalid number for factorial!"
   ElseIf Number = 0 Then
       CalculateFactorial = 1
   Else
       ' This is the recursive call:
       CalculateFactorial = Number * CalculateFactorial( Number - 1 )
   Endif
End Function
```

Das Beispiel liefert die Fakultät der Zahl `42` durch rekursives Aufrufen der Funktion `CalculateFactorial` bis es die Grundbedingung `0! = 1` erreicht.

Die Zahl der Rekursionsebenen ist zur Zeit auf 500 begrenzt.

Fehlerbehandlung

Eine korrekte Behandlung von Fehlersituationen gehört mit zu den aufwendigsten Aufgaben bei der Programmierung. StarOffice Basic bietet eine ganze Reihe von Hilfsmitteln an, um diese zu vereinfachen.

Die Anweisung On Error

Kern einer jeden Fehlerbehandlung bildet die Anweisung `On Error`:

```
Sub Test
      On Error Goto ErrorHandler

      ' ... Aufgabe ausführen, bei der ein Fehler passieren kann

Exit Sub

ErrorHandler:

      ' ... individueller Code zur Fehlerbehandlung

End Sub
```

Die Zeile `On Error Goto ErrorHandler` legt fest, wie StarOffice Basic im Falle eines Fehlers verfahren soll. Das `Goto ErrorHandler` sorgt dafür, dass StarOffice Basic die aktuelle Programmausführung abbricht und mit der Zeile `ErrorHandler:` fort fährt. Dort erfolgt eine individuelle Fehlerbehandlung, in der sich beispielsweise geöffnete Dateien schließen und Warnmeldungen ausgeben lassen.

Die Anweisung Resume

Es besteht die Möglichkeit, innerhalb der Fehlerbehandlung einen Sprung-Befehl zu positionieren. Dieser kann aus einem Befehl `Resume Next` bestehen,

```
ErrorHandler:

      ' ... individueller Code zur Fehlerbehandlung

      Resume Next
```

der die Programmausführung mit dem Befehl fortführt, der auf die Fehlerstelle folgt.

Weiter ist es möglich, für die Fortsetzung des Programms eine explizite Sprungmarke anzugeben, zu der das Programm nach der Fehlerbehandlung verzweigen soll:

```
ErrorHandler:

      ' ... individueller Code zur Fehlerbehandlung
      Resume Proceed

Proceed:

      ' ... hier gehts nach dem Fehler weiter
```

Wenn die Programmausführung im Falle eines Fehlers ohne jegliche Fehlermeldung fortgesetzt werden soll, steht zudem eine Kurzform aus `On Error und Resume Next` zur Verfügung:

```
Sub Test
      On Error Resume Next

      ' ... Aufgabe ausführen, bei der ein Fehler passieren kann

End Sub
```

Der Befehl `On Error Resume Next` sollte aufgrund seiner universellen Wirkung jedoch nur mit Vorsicht verwendet werden. Mehr zu dem Thema finden Sie unter *Tipps für eine strukturierte Fehlerbehandlung*.

Abfragen von Fehlerinformationen

Neben der Information, dass ein Fehler aufgetreten ist, sind für eine sinnvolle Fehlerbehandlung Zusatzinformationen über das Wo und Warum des Fehlers interessant:

- Die Variable `Err` enthält die Nummer des aufgetretenen Fehlers.
- In der Variable `Error$` steht auch eine Beschreibung des Fehlers zur Verfügung.
- Die Variable `Erl` enthält die Zeilennummer eines Fehlers.

Der Aufruf

```
MsgBox "Error " & Err & ": " & Error$ & " (line : " & Erl & ")"
```

zeigt, wie sich die Fehlerinformationen in einem Meldungsfensters anzeigen lassen.

VBA fasst die Fehlermeldungen in einem statischen Objekt namens `Err` zusammen. An seiner Stelle stehen in StarOffice Basic die Variablen `Err`, `Error$` und `Erl` zur Verfügung.

Die Statusinformationen sind so lange gültig bis das Programm auf eine `Resume`- oder `On Error`-Anweisung trifft. Dann werden sie zurück gesetzt.

In VBA erfolgt die Rücksetzung des Fehlerstatus nach einem Laufzeitfehler über die Methode `Err.Clear` des `Err`-Objektes. In StarOffice Basic sind statt dessen die Anweisungen `On Error` oder `Resume` zu verwenden.

Tipps für eine strukturierte Fehlerbehandlung

Die vorstehend erläuterten Möglichkeiten der Fehlerbehandlung sind vielseitig. Sowohl der Definitionsbefehl `On Error` als auch der Rückgabebefehl `Resume` sind nichts weiter als eine Variante des `Goto`-Konstrukts.

Wer seinen Programmcode sauber strukturieren möchte, sollte sich der Gefahren dieser Sprachkonstrukte bewusste sein und auf einen unkontrollierten Einsatz der Sprungbefehle verzichten.

Auch bei der Verwendung von `On Error Resume Next` ist Vorsicht angebracht. Ein entsprechender Aufruf sorgt dafür, dass *alle* offenen Fehlermeldungen geschlossen werden.

In der Praxis empfiehlt es sich, die Fehlerbehandlung innerhalb eines Programmes stets nach ein und demselben Schema zu realisieren. Es hat sich bewährt, den eigentlichen Programmcode strikt von der Fehlerbehandlung zu trennen und auf Rücksprünge in den ursprünglichen Code komplett zu verzichten.

Eine entsprechende Prozedur könnte wie folgt aussehen:

```
Sub Example

      ' Fehlerbehandlungsroutine am Anfang der Funktion definieren
      On Error Goto ErrorHandler
```

```
      ' ... Hier steht der eigentliche Programmcode

      ' Fehler-Behandlung ausschalten
      On Error Goto 0

      ' Ende der regulären Programmausführung
Exit Sub

' Startpunkt der Fehlerbehandlung
ErrorHandler:

      ' Prüfen, ob Fehler erwartet wurde
      If Err = ExpectedErrorNo Then
            ' ... Bearbeitung des Fehlers
      Else
            ' ... Warnhinweis wegen unerwartetem Fehler
      End If

      On Error Goto 0                    ' Fehlerbehandlung ausschalten
End Sub
```

Die Beispiel-Prozedur beginnt mit der Definition einer Fehlerbehandlungsroutine. An sie schließt sich der eigentliche Programmcode an. Nach dem Ende des Programmcodes folgt ein Aufruf `On Error Goto 0`, der die Fehlerbehandlung abschaltet. Schließlich sorgt die Anweisung `Exit Sub` (nicht zu verwechseln mit `End Sub`) dafür, dass die Ausführung der Prozedur beendet wird.

Die Beispiel-Prozedur fragt zunächst ab, ob die Fehlernummer tatsächlich der vom Programmierer erwarteten entspricht (hier in einer gedachten Konstanten `ExpectedErrorNo` abgelegt) und behandelt den Fehler dann entsprechend. Beim Auftreten eines anderen Fehlers gibt das System einen Warnhinweis aus. Die Überprüfung der Fehlernummer ist wichtig, damit sich keine unerwarteten Fehler unter dem Deckmantel der Fehlerbehandlung verstecken können.

Zu guter Letzt folgt auch im Fehlerbehandlungsblock der Aufruf `On Error Goto 0`. Er setzt die Statusinformationen für den Fehler (den Fehlercode in der Systemvariablen `Err`) wieder zurück, damit ein später auftretender Fehler abermals eindeutig erkennbar ist.

Die Laufzeitbibliothek von StarOffice Basic

Die folgenden Abschnitte stellen die zentralen Funktionen der Laufzeitbibliothek vor.

Konvertierungsfunktionen

Häufig muss ein Variablentyp in einen anderen Variablentyp geändert werden.

Implizite und explizite Typ-Umwandlungen

Die einfachste Möglichkeit, den Typ einer Variablen zu ändern besteht in einer Zuweisung.

```
Dim A As String
Dim B As Integer

B = 101
A = B
```

Im Beispiel handelt es sich bei der Variable A um eine Zeichenfolge, bei B um eine Integer-Variable. StarOffice Basic stellt sicher, dass die Variable B bei der Zuweisung an A entsprechend konvertiert wird. Diese Konvertierung ist weit aufwendiger, als es auf den ersten Blick scheint: Der Integer B liegt in Form einer zwei Byte langen Zahl im Arbeitsspeicher. A hingegen ist eine Zeichenfolge. Hier legt der Computer für jedes Zeichen (jede Ziffer) einen ein oder zwei Byte langen Wert ab. Vor dem Kopieren des Inhaltes von B nach A muss B daher zunächst in das interne Format der Variablen A umgewandelt werden. Erst dann ist eine Zuweisung möglich.

Basic führt die Typumwandlung im Gegensatz zu den meisten anderen Programmiersprachen automatisch durch. Dies kann mitunter fatale Folgen haben. Die zunächst eindeutig scheinende Code-Sequenz

```
Dim A As String
Dim B As Integer
Dim C As Integer

B = 1
C = 1
A = B + C
```

entpuppt sich bei genauerem Hinsehen als Falle. Wie lautet das Ergebnis des Ausdruckes? Berechnet der Basic-Interpreter zunächst das Ergebnis der Addition und wandelt es dann in eine Zeichenfolge um, so ergibt sich als Ergebnis die Zeichenfolge 2. Wandelt der Basic-Interpreter hingegen zunächst die Ausgangswerte B und C in eine Zeichenfolge um und wendet auf diese den Plus-Operator an, so ergibt sich als Ergebnis die Zeichenfolge 11.

Ähnliches gilt bei der Verwendung von Variant-Variablen:

```
Dim A
Dim B
Dim C

B = 1
C = "1"
A = B + C
```

Da Variant-Variablen sowohl Zahlen als auch Zeichenfolgen enthalten können, bleibt unklar, ob die Variable A die Zahl 2 oder die Zeichenfolge 11 zugewiesen bekommt.

Die genannten Fehlerquellen durch implizite Typ-Konvertierung lassen sich nur durch Disziplin bei der Programmierung umgehen, indem beispielsweise auf die Verwendung des Datentypes Variant verzichtet wird (was an dieser Stelle nochmals angeraten wird).

Um weitere Zweifelsfälle durch implizite Typ-Konvertierungen zu vermeiden, bietet StarOffice Basic eine Reihe von Konvertierungsfunktionen an, mit denen der Programmierer festlegen kann, wann der Datentyp einer Operation wie umgewandelt wird:

- **`CStr(Var)`** – wandelt einen beliebigen Datentyp in einen String um.
- **`CInt(Var)`** – wandelt einen beliebigen Datentypen in einen Integer-Wert um.
- **`CLng(Var)`** – wandelt einen beliebigen Datentypen in einen Long-Wert um.
- **`CSng(Var)`** – wandelt einen beliebigen Datentypen in einen Single-Wert um.
- **`CDbl(Var)`** – wandelt einen beliebigen Datentypen in einen Double-Wert um.
- **`CBool(Var)`** – wandelt einen beliebigen Datentypen in einen Booleschen Wert um.
- **`CDate(Var)`** – wandelt einen beliebigen Datentypen in einen Date-Wert um.

Mit diesen Konvertierungsfunktionen können Sie festlegen, wie StarOffice Basic die Typ-Konvertierungen durchführt:

```
Dim A As String
Dim B As Integer
Dim C As Integer

B = 1
C = 1

A = CInt(B + C)         ' B und C werden erst addiert, dann umgewandelt (ergibt Zahl 2)
A = CStr(B) + Cstr(C)   ' B und C werden erst in einen String umgewandelt, dann
                        'zusammengefügt (ergibt Zeichenfolge "11")
```

Bei der ersten Addition des Beispiels addiert StarOffice Basic zunächst die Integer-Variablen und wandelt anschließend das Ergebnis in eine Zeichenkette um. `A` erhält die Zeichenfolge `2`. Im zweiten Fall hingegen werden die Integer-Variablen zunächst in zwei Zeichenfolgen konvertiert und dann über die Zuweisung miteinander verkettet. `A` erhält somit die Zeichenfolge `11`.

Die nummerischen Konvertierungsfunktionen `CSng` und `CDbl` akzeptieren auch Dezimalzahlen. Als Dezimaltrennzeichen muss dafür das in den entsprechenden Ländereinstellungen definierte Zeichen verwendet werden. Umgekehrt verwendet die Methoden `CStr` beim Formatieren von Zahlen, Datums- und Zeitangaben die aktuell ausgewählten Ländereinstellungen.

Im Unterschied dazu steht die Funktion `Val`. Sie wandelt eine Zeichenfolge in eine Zahl um, erwartet darin jedoch prinzipiell einen *Punkt* als Dezimaltrennzeichen.

```
Dim A As String
Dim B As Double

A = "2.22"
B = Val(A)        ' Wird unabhängig von den Ländereinstellungen korrekt umgewandelt
```

Prüfen des Inhaltes von Variablen

In einigen Fällen ist eine Umwandlung der Daten nicht möglich:

```
Dim A As String
Dim B As Date

A = "test"
B = A                   ' erzeugt Fehlermeldung
```

Das Zuweisen der Zeichenfolge `test` an eine Datumsvariable im Beispiel ergibt keinen Sinn, so dass der Basic-Interpreter einen Fehler meldet. Das gleiche gilt für den Versuch, einer Booleschen Variable eine Zeichenfolge zuzuweisen:

```
Dim A As String
Dim B As Boolean

A = "test"
B = A                   ' erzeugt Fehlermeldung
```

Auch hier meldet der Basic-Interpreter einen Fehler.

Solche Fehlermeldungen lassen sich vermeiden, indem das Programm vor einer Zuweisung prüft, ob der Inhalt der zuzuweisenden Variablen zum Typ der Zielvariable passt. Hierfür stellt StarOffice Basic folgende Prüffunktionen zur Verfügung:

- **`IsNumeric(Value)`** – prüft, ob es sich bei einem Wert um eine Zahl handelt.
- **`IsDate(Value)`** – prüft, ob es sich bei einem Wert um ein Datum handelt.
- **`IsArray(Value)`** – prüft, ob es sich bei einem Wert um ein Array handelt.

Insbesondere bei der Abfrage von Benutzereingaben leisten diese Funktionen wertvolle Dienste. So lässt sich damit beispielsweise prüfen, ob ein Anwender tatsächlich eine gültige Zahl beziehungsweise ein gültiges Datum eingegeben hat.

```
If IsNumeric(UserInput) Then
      ValidInput = UserInput
Else
      ValidInput = 0
      MsgBox "Fehlermeldung."
End If
```

Enthält die Variable `UserInput` im Beispiel einen gültigen nummerischen Wert, so wird dieser der Variable `ValidInput` zugewiesen. Ist `UserInput` hingegen keine Zahl, so bekommt `ValidInput` den Wert `0` und das System warnt den Anwender mit einem Meldungsfenster.

Während zur Überprüfung von Zahlen, Datumsangaben und Arrays in Basic die oben genannten Prüffunktionen existieren, sucht man für Boolesche Werte eine entsprechende Funktion vergeblich. Diese lässt sich jedoch einfach nachbilden:

```
Function IsBoolean(Value As Variant) As Boolean
      On Error Goto ErrorIsBoolean:
      Dim Dummy As Boolean

      Dummy = Value

      IsBoolean = True
      On Error Goto 0
Exit Sub

ErrorIsBoolean:
      IsBoolean = False
      On Error Goto 0
End Function
```

Die gezeigte Funktion `IsBoolean` definiert eine interne Hilfsvariable `Dummy` mit dem Typ Boolean und versucht, dieser den übergebenen Wert zuzuweisen. Ist die Zuweisung erfolgreich, so liefert die Funktion den Wert `True` zurück. Scheitert sie, so entsteht ein Laufzeitfehler, den die Prüffunktion abfängt, um einen Fehlerstatus zurück zu liefern.

Enthält eine Zeichenfolge in StarOffice Basic einen nicht-nummerischen Wert und wird diese einer Zahl zugewiesen, so erzeugt StarOffice Basic keine Fehlermeldung, sondern überträgt den Wert`0` in die Variable. Dieses Verhalten unterscheidet sich von VBA. Dort wird bei einer entsprechenden Zuweisung ein Fehler ausgelöst und die Programmausführung unterbrochen.

Zeichenfolgen (Strings)

Arbeiten mit Zeichensätzen

StarOffice Basic verwendet bei der Verwaltung von Strings den Unicode-Zeichensatz. Die Funktionen `Asc` und `Chr` erlauben es dabei, den zu einem Zeichen gehörenden Unicode-Wert zu ermitteln beziehungsweise für einen Unicode-Wert das entsprechende Zeichen herauszufinden. Die folgenden Ausdrücke weisen der Variablen `Code` verschiedene Unicode-Werte zu:

```
Code = Asc("A")         ' Lateinischer Buchstabe A (Unicode-Wert 65)
Code = Asc("€")         ' Euro-Zeichen (Unicode-Wert 8364)
Code = Asc("п")         ' Kyrillischer Buchstabe п (Unicode-Wert 1083)
```

Umgekehrt sorgt der Ausdruck

```
MyString = Chr(13)
```

dafür, dass die Zeichenfolge `MyString` mit dem Wert des Zeichens `13` (steht für einen Zeilenumbruch) initialisiert wird.

Der Befehl `Chr` wird in Basic-Sprachen häufig dazu verwendet, Steuerzeichen in einen String einzufügen. So sorgt die Zuweisung

```
MyString = Chr(9) + "Das ist ein Test" + Chr(13)
```

dafür, dass dem Text ein Tabulatorzeichen (Unicode-Wert `9`) voran und ein Zeilenumbruchszeichen (Unicode-Wert `13`) angehängt wird.

Zugriff auf Teile einer Zeichenfolge

StarOffice Basic bietet vier Funktionen an, die ein Auslesen von Teilen einer Zeichenfolge gestatten:

- **`Left(MyString, Length)`** – liefert die ersten `Length` Zeichen von `MyString`.
- **`Right(MyString, Length)`** – liefert die letzten `Length` Zeichen von `MyString`.
- **`Mid(MyString, Start, Length)`** – liefert die ersten `Length` Zeichen von `MyString` ab der Position Start.
- **`Len(MyString)`** – liefert die Anzahl der Zeichen von `MyString`.

Hier einige Beispielaufrufe für die genannten Funktionen:

```
Dim MyString As String
Dim MyResult As String
Dim MyLen As Integer

MyString = "Das ist ein kleiner Test"

MyResult = Left(MyString,5)         ' Liefert die Zeichenfolge "Das i" zurück
MyResult = Right(MyString, 5) ' Liefert die Zeichenfolge " Test" zurück
MyResult = Mid(MyString, 8, 5)' Liefert die Zeichenfolge "ein k" zurück
MyLength = Len(MyString)            ' Liefert den Wert 4 zurück
```

Suchen und Ersetzen

Für das Suchen einer Teilzeichenfolge innerhalb einer anderen Zeichenfolge bietet StarOffice Basic die Funktion `InStr`:

```
ResultString = InStr (SearchString, MyString)
```

Der Parameter `SearchString` gibt die zu suchende Zeichenfolge an, auf die `MyString` durchsucht werden soll. Als Rückgabewert liefert die Funktion eine Zahl, die die Position beinhaltet, an der `SearchString` das erste Mal innerhalb von `MyString` auftaucht. Möchte man weitere Treffer der Zeichenfolge aufspüren, so bietet die Funktion die Möglichkeit, eine optionale Startposition anzugeben, ab der StarOffice Basic mit der Suche beginnt. Die Syntax der Funktion lautet in diesem Fall:

```
ResultString = InStr(StartPosition, SearchString, MyString)
```

In den genannten Beispielen ignoriert `InStr` die Groß- und Kleinschreibung. Alternativ besteht die Möglichkeit, durch Anfügen eines weiteren Parameters (der Ziffer `0`) das Suchverhalten so zu verändern, dass `InStr` Abweichungen in der Groß- und Kleinschreibung mit berücksichtigt. Dieser Aufruf lautet:

```
ResultString = InStr(SearchString, MyString, 0)
```

Mit den vorstehenden Funktionen zur Bearbeitung von Zeichenfolgen ist es bereits möglich, Teile einer Zeichenfolge in einer anderen Zeichenfolge zu suchen und zu ersetzen:

```
Function Replace(Source As String, Search As String, NewPart As String)
  Dim Result As String
  Dim StartPos As Long
  Dim CurrentPos As Long

  Result = ""
  StartPos = 1
  CurrentPos = 1

  If Search = "" Then
        Result = Source
    Else
        Do While CurrentPos <> 0
            CurrentPos = InStr(StartPos, Source, Search)
            If CurrentPos <> 0 Then
                Result = Result + Mid(Source, StartPos, _
                        CurrentPos - StartPos)
                Result = Result + NewPart
                StartPos = CurrentPos + Len(Search)
            Else
                Result = Result + Mid(Source, StartPos, Len(Source))
            End If ' Position <> 0
        Loop
    End If
    Replace = Result
End Function
```

Die Funktion durchsucht die übergebene Zeichenfolge `Search` in einer Schleife mittels `InStr` im Ausgangsausdruck `Source`. Stößt sie auf den gewünschten Suchausdruck, nimmt sie den Teil vor dem Ausdruck und schreibt ihn in den Rückgabepuffer `Result`. Diesem fügt sie an Stelle des Suchausdrucks `Search` den neuen Teil `Part` hinzu. Finden sich keine Treffer mehr für den Such-

ausdruck, so ermittelt die Funktion den übrig gebliebenen Teil der Zeichenfolge und fügt diesen ebenfalls an den Rückgabepuffer an. Die so erzeugte Zeichenfolge gibt sie als Ergebnis des Ersetzungsvorgangs zurück.

Da das Ersetzen von Teilzeichenfolgen mit zu den am häufigsten benötigten Funktionen gehört, wurde in StarOffice Basic die `Mid`-Funktion gegenüber dem Standardfunktionsumfang von Basic so erweitert, dass sie diese Aufgabe direkt übernimmt. Der Aufruf

```
Dim MyString As String

MyString  = "This was my text"
Mid(MyString, 6, 3, "is")
```

ersetzt in der Zeichenfolge `MyString` ab der sechsten Position drei Zeichen durch die Zeichenfolge `is`.

Formatieren von Zeichenfolgen

Die Funktion `Format` formatiert Zahlen als Zeichenfolge. Die Funktion erwartet dazu die Angabe eines `Format`-Ausdruckes, der als Schablone für die Formatierung der Zahlen dient. Jeder Platzhalter innerhalb der Schablone sorgt für eine entsprechende Formatierung dieser Position im Ausgabewert. Die fünf wichtigsten Platzhalter innerhalb einer Schablone sind die Zeichen *Null* (0), *Doppelkreuz* (#), *Punkt* (.), *Komma* (,) und *Dollar* ($).

Die Ziffer *Null* innerhalb der Schablone sorgt dafür, dass an der betreffenden Stelle stets eine Ziffer ausgegeben wird. Ist keine Ziffer vorhanden, so wird 0 ausgegeben.

Ein *Punkt* steht für das in den Ländereinstellungen vom Betriebssystem definierten Dezimaltrennzeichen.

Das Beispiel zeigt, wie sich mit den Ziffern *Null* und *Punkt* die Nachkommastellen eines Ausdruckes festlegen lassen:

```
MyFormat = "0.00"

MyString = Format(-1579.8, MyFormat)' Liefert "-1579,80" zurück
MyString = Format(1579.8, MyFormat)       ' Liefert "1579,80" zurück
MyString = Format(0.4, MyFormat)          ' Liefert "0,40" zurück
MyString = Format(0.434, MyFormat)        ' Liefert "0,43" zurück
```

Analog ist es möglich, eine Zahl mit führenden Nullen aufzufüllen:

```
MyFormat = "0000.00"

MyString = Format(-1579.8, MyFormat)' Liefert "-1579,80" zurück
MyString = Format(1579.8, MyFormat)       ' Liefert "1579,80" zurück
MyString = Format(0.4, MyFormat)          ' Liefert "0000,40" zurück
MyString = Format(0.434, MyFormat)        ' Liefert "0000,43" zurück
```

Ein *Komma* steht für das im Betriebssystem eingestellte Tausender-Trennzeichen, ein *Doppelkreuz* für eine Ziffer, die allerdings nur ausgegeben wird, wenn sie auch tatsächlich vorhanden ist.

```
MyFormat = "#,##0.00"

MyString = Format(-1579.8, MyFormat)' Liefert "-1.579,80" zurück
MyString = Format(1579.8, MyFormat)       ' Liefert "1.579,80" zurück
MyString = Format(0.4, MyFormat)          ' Liefert "0,40" zurück
```

```
MyString = Format(0.434, MyFormat)       ' Liefert "0,43" zurück
```

An Stelle des Platzhalters *Dollar* gibt die Format-Funktion das über das Betriebssystem definierte Währungszeichen aus:

```
MyFormat = "#,##0.00 $"

MyString = Format(-1579.8, MyFormat)' Liefert "-1.579,80 €" zurück
MyString = Format(1579.8, MyFormat)       ' Liefert "1.579,80 €" zurück
MyString = Format(0.4, MyFormat)          ' Liefert "0,40 €" zurück
MyString = Format(0.434, MyFormat)        ' Liefert "0,43 €" zurück
```

Bei Systemen, auf denen eine andere Währung als Euro eingestellt ist, wird das jeweils gültige Währungszeichen ausgegeben.

Die in VBA vorhandenen Format-Anweisungen zur Formatierung von Datums- und Zeit-Angaben werden in StarOffice Basic nicht unterstützt.

Datum und Uhrzeit

StarOffice Basic bietet den Datentyp `Date`, der Datums- und Uhrzeitangaben binär abspeichert und so den Umgang mit diesen vereinfacht.

Angabe von Datums- und Uhrzeitangaben innerhalb des Programmcodes

Man könnte versuchen, einer Date-Variablen ein Datum durch Zuweisung einer einfachen Zeichenfolge zuzuweisen:

```
Dim MyDate As Date

MyDate = "1.1.2002"
```

Diese Zuweisung könnte tatsächlich funktionieren, da StarOffice Basic den als String definierten Datumswert automatisch in eine Datumsvariable konvertiert. Diese Art der Zuweisung kann jedoch Fehler verursachen. Denn die Darstellung von Datums- und Zeitwerten ist von Land zu Land unterschiedlich definiert.

Da StarOffice Basic bei der Konvertierung einer Zeichenfolge in einen Datumswert die Ländereinstellungen des Betriebssystems verwendet, arbeitet der oben stehende Ausdruck nur so lange korrekt, wie die Ländereinstellungen mit dem Zeichenfolgenausdruck übereinstimmen.

Um dieses Problem zu umgehen, sollte die Funktion `DateSerial` für die Zuordnung eines fixen Wertes zu einer Datumsvariable verwendet werden:

```
Dim MyVar As Date

MyDate = DateSerial (2001, 1, 1)
```

Die Funktion erwartet die Datumsangaben stets in der Reihenfolge Jahr, Monat, Tag und sorgt dafür, dass die Variable unabhängig von den Ländereinstellungen tatsächlich den richtigen Wert zugewiesen bekommt.

Für Zeitangaben existiert analog die Funktion `TimeSerial`.

```
Dim MyVar As Date

MyDate =  TimeSerial(11, 23, 45)
```

Ihre Parameter sind in der Reihenfolge Stunden, Minuten, Sekunden anzugeben.

Extrahieren von Datums- und Zeitangaben

Das Gegenstück zu den Funktionen DateSerial und TimeSerial bilden die Funktionen:

- **`Day(MyDate)`** – liefert den Tag des Monats von `MyDate`
- **`Month(MyDate)`** – liefert den Monat aus `MyDate`
- **`Year(MyDate)`** – liefert das Jahr von `MyDate`
- **`Weekday(MyDate)`** – Liefert die Nummer des Wochentags von `MyDate`
- **`Hour(MyTime)`** – Liefert die Stunde aus `MyTime`
- **`Minute(MyTime)`** – Liefert die Minuten aus `MyTime`
- **`Second(MyTime)`** – Liefert die Sekunden aus `MyTime`

Diese Funktionen extrahieren die genannten Datums- beziehungsweise Zeitanteile aus einer vorgegebenen `Date`-Variable. Das Beispiel

```
Dim MyDate As Date

' ... Initialisierung von MyDate

If Year(MyDate) = 2003 Then

      ' ... Angegebenes Datum liegt im Jahr 2003
End If
```

prüft, ob das in `MyDate` abgelegte Datum im Jahr 2003 liegt. Analog dazu das Beispiel

```
Dim MyTime As Date

' ... Initialisierung von MyTime

If Hour(MyTime) >= 12 And Hour(MyTime) < 14 Then

      ' ... Angegebene Zeit liegt zwischen 12 und 14 Uhr
End If
```

ob `MyTime` zwischen `12` und `14` Uhr liegt.

Die Funktion `Weekday` ermittelt die Nummer des Wochentages für das übergebene Datum:

```
Dim MyDate As Date
Dim MyWeekday As String

' ... MyDate initialisieren

Select Case WeekDay(MyDate)
case 1
      MyWeekday = "Sonntag"
case 2
```

```
      MyWeekday = "Montag"
case 3
      MyWeekday = "Dienstag"
case 4
      MyWeekday = "Mittwoch"
case 5
      MyWeekday = "Donnerstag"
case 6
      MyWeekday = "Freitag"
case 7
      MyWeekday = "Samstag"
End Select
```

Als erster Tag der Woche gilt dabei der Sonntag.

Systemdatum- und Zeit auslesen

Zum Auslesen der Systemzeit und des Systemdatums stehen in StarOffice Basic folgende Funktionen zur Verfügung:

- **`Date`** – liefert das aktuelle Datum
- **`Time`** – liefert die aktuelle Uhrzeit
- **`Now`** – liefert den aktuellen Zeitpunkt (Datum und Uhrzeit als Kombinationswert)

Dateien und Verzeichnisse

Der Umgang mit Dateien gehört mit zu den Basisaufgaben einer Anwendung. Das StarOffice API bietet Ihnen eine ganze Reihe von Objekten, mit denen Sie Office-Dokumente anlegen, öffnen und ändern können. Diese werden ausführlich in Kapitel 4 vorgestellt. Unabhängig davon ist es in einigen Fällen notwendig, direkt auf das Dateisystem zuzugreifen, Verzeichnisse zu durchsuchen oder Textdateien zu bearbeiten. Für diese Aufgaben stellt die Laufzeitbibliothek von StarOffice Basic einige grundlegende Funktionen zur Verfügung.

Mit StarOffice 7.0 stehen einige DOS-spezifische Datei- und Verzeichnisfunktionen nicht mehr oder nur noch eingeschränkt zur Verfügung. So fehlt beispielsweise eine Unterstützung der Funktionen `ChDir`, `ChDrive` und `CurDir`. In den Funktionen, die Dateiattribute als Parameter erwarten, sind einige DOS-spezifische Attribute weggefallen (etwa zur Differenzierung von versteckten Dateien und Systemdateien). Diese Änderung wurde notwendig, um eine möglichst hohe Plattformunabhängigkeit von StarOffice sicher zu stellen.

Dateien verwalten

Verzeichnisse durchsuchen

Das Durchsuchen von Verzeichnissen nach Dateien und Unterverzeichnissen übernimmt in StarOffice Basic die Funktion `Dir`. Beim ersten Aufruf muss `Dir` eine Zeichenfolge mit dem Pfad des zu durchsuchenden Verzeichnisses übergeben werden. Ein zweiter Parameter von `Dir` legt fest, wonach StarOffice Basic suchen soll (z.B. normale Dateien oder Verzeichnisse). StarOffice Basic liefert den Namen des ersten gefundenen Verzeichniseintrags zurück. Um den nächsten

Eintrag zu erhalten, ist die Funktion `Dir` ohne Parameter aufzurufen. Findet die Funktion `Dir` keine weiteren Einträge mehr, liefert sie einen Leerstring zurück.

Das nachfolgende Beispiel zeigt, wie sich mit der Funktion `Dir` sämtliche Dateien eines Verzeichnisses auslesen lassen. Die Prozedur legt die einzelnen Dateinamen in der Variablen `AllFiles` ab und gibt diese im Anschluss in einem Meldungsfenster aus.

```
Sub ShowFiles
      Dim NextFile As String
      Dim AllFiles As String

      AllFiles = ""
      NextFile = Dir("C:\", 0)

      While NextFile  <> ""
            AllFiles = AllFiles & Chr(13) &  NextFile
            NextFile  = Dir
      Wend

      MsgBox AllFiles
End Sub
```

Die `0` als zweiter Parameter der Funktion `Dir` sorgt dafür, dass `Dir` ausschließlich die Namen von Dateien zurück gibt, Verzeichnisse aber ignoriert. Hier kann man folgende Parameter angeben:

- `0` : normale Dateien
- `16` : Unterverzeichnisse

Das nachfolgende Beispiel entspricht weitgehend dem oben stehenden, übergibt der Funktion `Dir` jedoch den Wert 16 als Parameter, so dass diese anstelle der Dateinamen die Unterverzeichnisse eines Ordners zurück gibt.

```
Sub ShowDirs
      Dim NextDir As String
      Dim AllDirs As String
      AllDirs = ""
      NextDir = Dir("C:\", 16)
      While NextDir <> ""
            AllDirs = AllDirs & Chr(13) &  NextDir
            NextDir  = Dir
      Wend
      MsgBox AllDirs
End Sub
```

Mit dem Parameter `16` liefert die `Dir`-Funktion bei einem Aufruf in StarOffice Basic, im Gegensatz zu VBA, tatsächlich nur die Unterverzeichnisse eines Ordners. (In VBA liefert die Funktion zusätzlich die Namen der gewöhnlichen Dateien, so dass eine weitere Überprüfung notwendig ist, um tatsächlich nur die Verzeichnisse zu erhalten).

Die in VBA vorhandenen Möglichkeiten, Verzeichnisse gezielt nach Dateien mit den Attributen *versteckt*, *Systemdatei*, *archiviert* und *Volumenname* zu durchsuchen, existieren in StarOffice Basic nicht, da die betreffenden Dateisystemfunktionen nicht auf allen Betriebssystemen zur Verfügung stehen.

Die in `Dir` aufgeführten Pfadangaben dürfen sowohl in VBA als auch in StarOffice Basic die Platzhalter * und ? verwenden. In StarOffice Basic darf der Platzhalter * im Gegensatz zu VBA jedoch nur als letztes Zeichen des Dateinamens beziehungsweise der Dateierweiterung stehen.

Verzeichnisse anlegen und löschen

Für das Anlegen von Verzeichnissen bietet StarOffice Basic die Funktion `MkDir`:

```
MkDir ("C:\SubDir1")
```

Diese Funktion legt beliebige Verzeichnisse und Unterverzeichnisse an. Alle benötigten Verzeichnisse innerhalb einer Hierarchie werden bei Bedarf mit angelegt. Existiert beispielsweise lediglich das Verzeichnis `C:\SubDir1`, so legt ein Aufruf

```
MkDir ("C:\SubDir1\SubDir2\SubDir3\")
```

sowohl das Verzeichniss `C:\SubDir1\SubDir2` als auch `C:\SubDir1\SubDir2\SubDir3` an.

Das Gegenstück der Funktion `MkDir` bildet die Funktion `RmDir`:

```
RmDir ("C:\SubDir1\SubDir2\SubDir3\")
```

Sie löscht ein beliebiges Verzeichnis. Enthält das Verzeichnis Unterverzeichnisse oder Dateien, so werden diese *ebenfalls gelöscht.* Deshalb sollte man vorsichtig sein bei Aufrufen von `RmDir`. So leistungsfähig die Funktion ist, so gefährlich ist sie bei unsachgemäßer Anwendung.

In VBA beziehen sich die Funktionen `MkDir` und `RmDir` nur auf das aktuelle Verzeichnis. In StarOffice Basic hingegen ist es möglich, mit `MkDir` und `RmDir` mehrere Ebenen von Verzeichnissen anzulegen beziehungsweise zu löschen.

In VBA erzeugt `RmDir` eine Fehlermeldung, falls ein Verzeichnis eine Datei enthält. In StarOffice Basic wird das Verzeichnis *samt darin enthaltener Datei* gelöscht.

Dateien kopieren, umbenennen, löschen und auf Existenz prüfen

Der Aufruf

```
FileCopy(Source, Destination)
```

erstellt eine Kopie der Datei `Source` unter dem Namen `Destination`.

Mit Hilfe der Funktion

```
Name OldName As NewName
```

ist es möglich, die Datei `OldName` in `NewName` umzubenennen. Die etwas eigentümliche Syntax mit dem Schlüsselwort `As` und ohne Komma geht auf die Wurzeln von Basic zurück.

Der Aufruf

```
Kill(Filename)
```

löscht die Datei `Filename` (soll ein Verzeichnis inklusive Dateien gelöscht werden, ist die oben stehende Funktion `RmDir` zu verwenden).

Ob eine Datei existiert, lässt sich über die Funktion `FileExists` prüfen:

```
If FileExists(Filename) Then
      MsgBox "Datei existiert."
End If
```

Dateiattribute lesen und ändern

Bei der Arbeit mit Dateien ist es mitunter wichtig, die Dateiattribute, den Zeitpunkt der letzten Dateiänderung und die Länge der Datei zu ermitteln.

Der Aufruf

```
Dim Attr As Integer
Attr = GetAttr(Filename)
```

liefert einige Attribute über eine Datei zurück. Das Ergebnis wird als Bit-Maske ausgegeben, wobei folgende Werte möglich sind:

- 1 : schreibgeschützte Datei
- 16 : Name eines Verzeichnisses

Das Beispiel

```
Dim FileMask As Integer
Dim FileDescription As String

FileMask = GetAttr("test.txt")

If (FileMask AND 1) > 0 Then
      FileDescription = FileDescription & " schreibgeschützt "
End IF

If (FileMask AND 16) > 0 Then
      FileDescription = FileDescription & " Verzeichnis "
End IF

If FileDescription = "" Then
      FileDescription = " normal "
End IF

MsgBox FileDescription
```

ermittelt die Bit-Maske der Datei `test.txt` und prüft, ob diese schreibgeschützt und/oder ein Verzeichnis ist. Trifft keiner der Fälle zu, erhält `FileDescription` den Text "normal" zugewiesen.

Die in VBA vorhandenen Flags zum Abfragen der Dateiattribute *versteckt*, *Systemdatei*, *archiviert* und *Volumenname* werden in StarOffice Basic nicht unterstützt, da diese Windows-spezifisch sind und auf anderen Betriebssystemen nicht oder nur teilweise zur Verfügung stehen.

Die Funktion `SetAttr` gestattet es, die Attribute einer Datei zu ändern. So lässt sich eine Datei über den Aufruf

```
SetAttr("test.txt", 1)
```

mit einem Schreibschutz versehen. Ein bestehender Schreibschutz kann über den folgenden Aufruf gelöscht werden.

```
SetAttr("test.txt", 0)
```

Das Datum und die Uhrzeit der letzten Änderung einer Datei liefert die Funktion `FileDateTime` zurück. Das Datum wird dabei gemäß der auf dem System getroffenen Ländereinstellungen formatiert.

```
FileDateTime("test.txt")' Liefert Datum und Uhrzeit der letzten Dateiänderung zurück.
```

Die Funktion `FileLen` ermittelt die Länge einer Datei in Byte (als Long Integer-Wert).

```
FileLen("test.txt")     ' Liefert die Länge der Datei in Byte  zurück
```

Textdateien schreiben und lesen

StarOffice Basic bietet eine ganze Reihe von Methoden für das Lesen und Schreiben von Dateien an. Die nachfolgenden Erläuterungen beziehen sich auf die Arbeit mit Textdateien. (*nicht* Textdokumenten).

Schreiben von Textdateien

Vor dem Zugriff auf eine Textdatei muss diese zunächst geöffnet werden. Dazu wird ein freies *Datei-Handle* benötigt, dass bei allen späteren Dateizugriffen die Datei eindeutig identifiziert.

Für das Ermitteln eines freien Datei-Handles dient die Funktion `FreeFile`. Das von ihr zurück gelieferte Handle dient als Parameter der `Open`-Anweisung, die die Datei öffnet. Soll die Datei so geöffnet werden, dass sie sich als Textdatei beschreiben lässt, so lautet der `Open`-Aufruf:

```
Open Filename For Output As #FileNo
```

`Filename` steht dabei für eine Zeichenfolgenvariable mit dem Namen der Datei. `FileNo` für das über `FreeFile` ermittelte Handle.

Ist die Datei geöffnet, so lässt sie sich zeilenweise über die Anweisung `Print` beschreiben:

```
Print #FileNo, "Dies ist eine Testzeile."
```

`FileNo` steht auch hier für das Datei-Handle. Der zweite Parameter gibt den Text an, der als Zeile der Textdatei gespeichert werden soll.

Ist der Schreibvorgang abgeschlossen, so muss die Datei über einen Aufruf `Close` geschlossen werden:

```
Close #FileNo
```

Auch hier ist das Datei-Handle anzugeben.

Das nachfolgende Beispiel zeigt, wie eine Textdatei geöffnet, beschrieben und geschlossen wird:

```
Dim FileNo As Integer
```

```
Dim CurrentLine As String
Dim Filename As String

Filename = "c:\data.txt"                  ' Dateiname festlegen
FileNo = Freefile                               ' freies Datei-Handle ermitteln

Open Filename For Output As #FileNo             ' Datei öffnen (Schreibmodus)
Print #FileNo, "This is a line of text"         ' Zeile speichern
Print #FileNo, "This is another line of text"   ' Zeile speichern
Close #FileNo                                   ' Datei schließen
```

Lesen von Textdateien

Das Lesen von Textdateien erfolgt analog zum Schreiben der Datei. Die `Open`-Anweisung beim Öffnen enthält anstelle des Ausdrucks `For Output` den Ausdruck `For Input` und anstelle des `Print`-Befehls zum Schreiben der Daten ist die Anweisung `Line Input` zum Lesen der Daten zu verwenden.

Schließlich dient beim Auslesen einer Textdatei die Anweisung

```
eof(FileNo)
```

zur Prüfung, ob das Ende der Datei erreicht ist, oder ob weitere Zeilen vorhanden sind.

Das nachfolgende Beispiel verdeutlicht, wie eine Textdatei eingelesen werden kann.

```
Dim FileNo As Integer
Dim CurrentLine As String
Dim File As String
Dim Msg as String

' Dateiname festlegen
Filename = "c:\data.txt"

' freies Datei-Handle ermitteln
FileNo = Freefile

' Datei öffnen (Lesemodus)
Open Filename For Input As FileNo

' Prüfen, ob Dateiende erreicht ist
Do While not eof(FileNo)

      ' Zeile lesen
      Line Input #FileNo, CurrentLine
      If CurrentLine <>"" then
            Msg = Msg & CurrentLine & Chr(13)
      end if

Loop

' Datei schließen
Close #FileNo

Msgbox Msg
```

Die einzelnen Zeilen werden in einer `Do While`-Schleife ausgelesen, in der Variable `Msg` abgelegt und am Ende in Form eines Meldungsfensters ausgegeben.

Meldungs- und Eingabefenster

Für eine einfache Benutzerkommunikation stellt StarOffice Basic die Funktionen `MsgBox` und `InputBox` bereit.

Meldungen ausgeben

`MsgBox` gibt ein einfaches Hinweisfenster aus, das sich mit einer oder mehreren Schaltflächen versehen lässt. In der einfachsten Variante

```
MsgBox "Das ist ein Hinweis!"
```

enthält `MsgBox` lediglich den Text sowie die Schaltfläche Ok.

Das Erscheinungsbild des Hinweisfensters kann über einen Parameter geändert werden. Der Parameter bietet die Möglichkeit, zusätzliche Schaltflächen einzublenden, die vorbelegte Schaltfläche festzulegen und ein Hinweissymbol einzublenden. Die Werte zur Auswahl der Schaltflächen lauten:

- 0 – Schaltfläche OK
- 1 – Schaltflächen OK und Abbrechen
- 2 – Schaltflächen Abbrechen und Wiederholen
- 3 – Schaltflächen Ja, Nein und Abbrechen
- 4 – Schaltflächen Ja und Nein
- 5 – Schaltflächen Wiederholen und Abbrechen

Je nachdem ob die erste, zweite oder dritte Schaltfläche voreingestellt werden soll, ist einer der folgenden Werte zu addieren:

- 0 – erste Schaltfläche ist Vorgabewert
- 256 – zweite Schaltfläche ist Vorgabewert
- 512 – dritte Schaltfläche ist Vorgabewert

Schließlich stehen folgende Hinweissymbole zur Verfügung, die sich ebenfalls durch Addition des jeweiligen Werts einblenden lassen:

- 16 – Stop-Schild
- 32 – Fragezeichen
- 48 – Ausrufezeichen
- 64 – Hinweis-Icon

Der Aufruf

```
MsgBox "Möchten Sie fortfahren?",  292
```

gibt ein Hinweisfenster aus mit den Schaltflächen Ja und Nein (Wert 4), von denen die zweite Schaltfläche (Nein) als Vorgabewert eingestellt ist (Wert 256) und das zusätzlich ein Fragezeichen-Symbol enthält (Wert 32), 4+256+32=292.

Enthält ein Hinweisfenster mehrere Schaltflächen, so sollte sein Rückgabewert abgefragt werden um zu ermitteln, welche Schaltfläche gedrückt wurde. Dabei sind folgende Rückgabewerte möglich:

- 1 – Ok
- 2 – Abbrechen
- 4 – Wiederholen
- 5 – Ignorieren
- 6 – Ja
- 7 – Nein

Im obigen Beispiel könnte eine Prüfung des Rückgabewertes wie folgt aussehen:

```
If MsgBox ("Möchten Sie fortfahren?",  292) = 6 Then
      ' Schaltfläche Ja gedrückt
Else
      ' Schaltfläche Nein gedrückt
End IF
```

Neben dem Hinweistext und dem Parameter für die Gestaltung des Hinweisfensters gestattet `MsgBox` die Angabe eines dritten Parameters, der einen individuellen Text für den Fenstertitel festlegt:

```
MsgBox "Möchten sie fortfahren?",  292, "Fenstertitel"
```

Wird kein Fenstertitel angegeben, so lautet dieser `soffice`.

Eingabefenster zur Abfrage einfacher Zeichenfolgen

Die Funktion `InputBox` fragt einfache Zeichenfolgen vom Anwender ab. Sie stellt damit eine einfache Alternative für die Gestaltung aufwändiger Dialoge zur Verfügung. `InputBox` nimmt drei Standardparameter entgegen:

- einen Hinweistext,
- einen Fenstertitel,
- einen Vorgabewert, der innerhalb des Eingabereiches eingeblendet werden kann.

```
InputVal = InputBox("Bitte Wert eingeben:", "Test", "Vorgabewert")
```

Als Rückgabewert liefert `InputBox` die vom Anwender eingegebene Zeichenfolge.

Sonstige Funktionen

Beep

Die Funktion `Beep` gibt einen Hinweiston aus und eignet sich beispielsweise, um den Benutzer bei einer falschen Aktion über einen Hinweis zu warnen. `Beep` wird ohne jeden Parameter aufgerufen:

```
Beep  ' erzeugt Hinweiston
```

Shell

Über die Funktion `Shell` lassen sich externe Programme starten.

```
Shell(Pathname, Windowstyle, Param)
```

`Pathname` legt den Pfad sowie den Namen des aufzurufenden Programms fest. `Windowstyle` definiert, in welchem Fenster das Programm gestartet wird. Möglich sind die Werte:

- 0 – Das Programm erhält den Fokus und wird in einem versteckten Fenster gestartet.
- 1 – Das Programm erhält den Fokus und wird in einem Fenster in Normalgröße gestartet.
- 2 – Das Programm erhält den Fokus und wird in einem minimierten Fenster gestartet.
- 3 – Das Programm erhält den Fokus und wird in einem maximierten Fenster gestartet.
- 4 – Das Programm wird in einem Fenster in Normalgröße gestartet, ohne jedoch den Fokus zu erhalten.
- 6 – Das Programm wird in einem minimierten Fenster gestartet, der Fokus bleibt beim aktuellen Fenster.
- 10 – Das Programm wird im Vollbild-Modus gestartet.

Der dritte Parameter `Param` gestattet es, Kommandozeilenparameter an das zu startende Programm zu übergeben.

Wait

Die Funktion `Wait` unterbricht die Programmausführung für eine vorgegebene Zeit. Die Angabe der Wartezeit erfolgt in Millisekunden. Der Befehl

```
Wait 2000
```

beispielsweise sorgt für eine Unterbrechnung von 2 Sekunden (2000 Millisekunden).

Environ

Mittels der Funktion `Environ` lassen sich Umgebungsvariablen des Betriebssystems auslesen. Je nach System und Konfiguration sind dort unterschiedliche Daten abgelegt. Der Aufruf

```
Dim TempDir

TempDir=Environ ("TEMP")
```

ermittelt beispielsweise das temporäre Verzeichnis des Betriebssystems.

Einführung in das StarOffice API

Das StarOffice API ist eine universelle Programmierschnittstelle für den Zugriff auf StarOffice. Mit dem StarOffice API können Sie StarOffice-Dokumente anlegen, öffnen, verändern und ausdrucken. Es bietet die Möglichkeit, den Funktionsumfang von StarOffice durch eigene Makros zu erweitern und gestattet die Erstellung eigener Dialoge.

Das StarOffice API lässt sich nicht nur von StarOffice Basic aus nutzen, sondern auch von anderen Programmiersprachen wie Java und C++. Möglich macht das eine Technik namens *Universal Network Objects* (UNO), die eine Schnittstelle zu verschiedenen Programmiersprachen anbietet.

Dieses Kapitel beschreibt, wie sich StarOffice mit Hilfe von UNO in StarOffice Basic nutzen lässt. Es beschreibt die prinzipiellen Konzepte von UNO aus der Sicht eines StarOffice Basic-Programmierers. Details zum Umgang mit den verschiedenen Teilbereichen des StarOffice API finden Sie in den nachfolgenden Kapiteln.

Universal Network Objects (UNO)

StarOffice stellt eine Programmierschnittstelle in Form von Universal Network Objects (UNO) zur Verfügung. Es handelt sich dabei um eine objektorientierte Programmierschnittstelle, die StarOffice in verschiedene Objekte aufteilt, die ihrerseits programmgesteuerten Zugriff auf das Office-Paket gewähren.

Da StarOffice Basic an sich eine prozedurale Programmiersprache ist, musste es um einige Sprachkonstrukte erweitert werden, die eine Nutzung von UNO möglich machen.

Um ein Universal Network Object in StarOffice Basic zu verwenden, bedarf es einer Variablendeklaration für das zugehörige Objekt. Die Deklaration erfolgt über die `Dim`-Anweisung (vgl. Kapitel 2). Für die Deklaration einer Objektvariable ist die Typbezeichnung `Object` zu verwenden:

```
Dim Obj As Object
```

Der Aufruf definiert eine Objekt-Variable namens `Obj`.

Die erzeugte Objektvariable muss anschließend initialisiert werden, damit sie genutzt werden kann. Das ist beispielsweise über die Funktion `createUnoService` möglich:

```
Obj = createUnoService("com.sun.star.frame.Desktop")
```

Dieser Aufruf ordnet der Variable `Obj` einen Verweis auf ein neu erzeugtes Objekt zu. `com.sun.star.frame.Desktop` ist dabei so etwas wie der Typ des Objektes, der in der UNO-Terminologie jedoch nicht Typ sondern *Service* genannt wird. Nach der UNO-Philosophie

bezeichnet man `Obj` als *einen Verweis auf ein Objekt, das den Service* `com.sun.star.frame.Desktop` *unterstützt*. Der in StarOffice Basic verwendete Begriff Service entspricht somit mehr oder weniger den in anderen Programmiersprachen verwendeten Begriffen *Typ* und *Klasse*.

Es existiert allerdings ein wesentlicher Unterschied: Ein Universal Network Object kann mehrere Services gleichzeitig unterstützen. Einige UNO-Services unterstützen wiederum andere Services, so dass dem Programmierer mit einem Objekt automatisch eine ganze Reihe von Services zur Verfügung stehen. So ist es beispielsweise möglich, dass das oben stehende Objekt, das auf dem Service `com.sun.star.frame.Desktop` basiert, weitere Services zum Laden von Dokumenten und einen zum Beenden des Programms mit sich bringt.

Während der Aufbau eines Objektes in VBA über die Klasse festgelegt wird, zu der es gehört, wird der Aufbau in StarOffice Basic über die Services definiert, die es unterstützt. Ein VBA-Objekt ist immer genau einer einzigen Klasse zugeordnet. Ein StarOffice Basic-Objekt kann hingegen mehrere Services unterstützen.

Eigenschaften und Methoden

Ein Objekt stellt in StarOffice Basic eine Reihe von Eigenschaften und Methoden zur Verfügung, die über das Objekt aufgerufen werden können.

Eigenschaften

Eigenschaften (*Properties*) sind so etwas wie die Attribute eines Objektes, z. B. `Filename` und `Title` für ein `Document`- Objekt.

Das Setzen von Eigenschaften erfolgt über eine einfache Zuweisung:

```
Document.Title = "Programmierhandbuch StarOffice 7.0"
Document.Filename = "progman.sxv"
```

Eine Eigenschaft besitzt wie eine gewöhnliche Variable einen Typ, der festlegt, welche Werte sie aufnehmen kann. Die vorstehenden Eigenschaften `Filename` und `Title` haben den Typ String.

Echte Eigenschaften und nachgebildete Eigenschaften

Die meisten Eigenschaften eines Objektes in StarOffice Basic sind als solche in der UNO-Beschreibung des Services definiert. Neben diesen "echten" Eigenschaften gibt es in StarOffice Basic auch Eigenschaften, die auf UNO-Ebene aus zwei Methoden bestehen. Die eine dient dazu, den Wert der Eigenschaft abzufragen, die andere, ihn zu setzen (`get`- und `set`-Methoden). Die Eigenschaft wurde quasi aus zwei Methoden nachgebildet. So stellen Zeichenobjekte in UNO beispielsweise die Methoden `getPosition` und `setPosition` zur Verfügung, über die sich die zugehörigen Eckpunkte auslesen und ändern lassen. Der StarOffice Basic-Programmierer kann auf die Werte über die Eigenschaft `Position` zugreifen. Unabhängig davon stehen ihm zusätzlich die Original-Methoden zur Verfügung (im Beispiel `getPosition` und `setPosition`).

Methoden

Unter Methoden versteht man Funktionen, die sich direkt auf ein Objekt beziehen und über dieses aufgerufen werden. Das vorstehende Objekt `Document` könnte beispielsweise eine Methode `Save` anbieten, die sich wie folgt aufrufen ließe:

```
Document.Save()
```

Methoden können analog zu Funktionen Parameter und Rückgabewerte enthalten. Die Syntax derartiger Methodenaufrufe orientiert sich an der klassischer Funktionen. Der Aufruf

```
Ok = Document.Save(True)
```

gibt dem Objekt Document beim Aufruf der Methode Save den Parameter `True` mit. Nach Abschluss der Methode legt `Save` einen Rückgabewert in der Variablen `Ok` ab.

Module, Services und Schnittstellen

StarOffice stellt viele hundert Services zur Verfügung. Um den Überblick darüber zu vereinfachen, wurden diese in Modulen zusammen gefasst. Eine weitere funktionale Bedeutung haben die Module für den StarOffice Basic-Programmierer nicht. Einzig bei der Angabe eines Service-Namens ist der Modulname wichtig, da er mit im Namen aufgeführt werden muss. Der vollständige Name eines Services setzt sich zusammen aus dem Ausdruck `com.sun.star`, der angibt, dass es sich um einen StarOffice-Service handelt, und dem Modulnamen, beispielsweise `frame` und schließlich dem eigentlichen Servicenamen, etwa `Desktop`. Der vollständige Name würde in dem genannten Beispiel lauten:

```
com.sun.star.frame.Desktop
```

Neben den Begriffen Modul und Service führt UNO den Begriff der Schnittstelle (*Interface*) ein. Während dieser Begriff den Java-Programmierern vertraut vorkommen dürfte, ist er in der Basic-Welt unbekannt.

Ein Interface fasst mehrere Methoden zusammen. Streng genommen unterstützt ein Service in UNO keine Methoden sondern Schnittstellen, die wiederum verschiedene Methoden bereit stellen. Anders ausgedrückt: Die Methoden werden dem Service in Schnittstellen zusammengefasst zugeordnet. Diese Feinheit mag insbesondere den Java- oder C++-Programmierer interessieren. Denn dort wird die Schnittstelle für den Aufruf einer Methode benötigt. In StarOffice Basic ist dies irrelevant. Hier werden die Methoden direkt über das jeweilige Objekt aufgerufen.

Für das Verständnis des API ist es dennoch hilfreich, die Zuordnung der Methoden zu den verschiedenen Schnittstellen parat zu haben. Denn viele Schnittstellen werden in verschiedenen Services benutzt. Kennt man eine Schnittstelle, so kann man sein Wissen von einem Service auf einen anderen übertragen.

Einige zentrale Schnittstellen werden derart häufig verwendet, dass sie weiter hinten in diesem Kapitel losgelöst von den verschiedenen Services dargestellt werden.

Handwerkszeug zum Umgang mit UNO

Bleibt die Frage, welche Objekte – oder Services, um bei der UNO-Terminologie zu bleiben – welche Eigenschaften, Methoden und Schnittstellen unterstützen und wie sich diese ermitteln lassen. Neben diesem Handbuch, das sich im Folgenden praktisch ausschließlich der Beschreibung der verschiedenen UNO-Services widmet, existieren drei Möglichkeiten, mit deren Hilfe sich ein Programmierer eigenständig über *die Fähigkeiten* von Objekten sachkundig machen kann: Die Methode `supportsService`, die Debug-Methoden sowie der Developer's Guide und die API-Referenz.

Die Methode supportsService

Eine Reihe von UNO-Objekten unterstützt die Methode `supportsService`, mit der sich ermitteln lässt, ob ein Objekt einen bestimmten Service unterstützt oder nicht. So ermittelt der Aufruf

```
Ok = TextElement.supportsService("com.sun.star.text.Paragraph")
```

beispielsweise, ob das Objekt `TextElement` den Service `com.sun.star.text.Paragraph` unterstützt.

Die Debug-Eigenschaften

Jedes UNO-Objekt in StarOffice Basic *weiß*, welche Eigenschaften, Methoden und Schnittstellen es bereit hält. Es bietet Eigenschaften, die diese in Form einer Liste ausgeben. Die entsprechenden Eigenschaften lauten:

`DBG_properties` - liefert einen String mit allen Eigenschaften eines Objektes

`DBG_methods` - liefert einen String mit allen Methoden eines Objektes

`DBG_supportetInterfaces` - liefert einen String mit allen Interfaces, die ein Objekt unterstützt.

Der nachfolgende Programm-Code zeigt, wie sich `DBG_properties` und `DBG_methods` in der Praxis verwenden lassen. Er erzeugt zunächst den Service `com.sun.star.frame.Desktop` und gibt dann die unterstützten Eigenschaften und Methoden in Meldungsfenstern aus.

```
Dim Obj As Object
Obj = createUnoService("com.sun.star.frame.Desktop")

MsgBox Obj.DBG_Properties
MsgBox Obj.DBG_methods
```

Bei der Verwendung von `DBG_properties` ist zu beachten, dass die Funktion alle Eigenschaften zurück liefert, die ein bestimmter Service theoretisch unterstützen kann. Es ist jedoch nicht sicher gestellt, dass diese von dem betreffenden Objekt auch verwendet werden. Vor dem Auslesen einer Eigenschaft muss daher mit der Funktion `IsEmpty` geprüft werden, ob diese tatsächlich verfügbar ist.

API-Referenz

Weitere Informationen über die vorhandenen Services, ihre Schnittstellen, Methoden und Eigenschaften liefert die API-Referenz für das StarOffice API. Sie steht auf www.openoffice.org zur Verfügung:

```
http://api.openoffice.org/common/ref/com/sun/star/module-ix.html
```

Einige zentrale Schnittstellen im Überblick

Einige Schnittstellen von StarOffice finden sich in vielen Bereichen des StarOffice API. Sie definieren Sätze von Methoden für abstrakte Aufgabenstellungen, die sich auf verschiedene Probleme anwenden lassen. Hier finden Sie einen Überblick über die häufigsten Schnittstellen.

Der Ursprung der Objekte wird erst später in diesem Handbuch erklärt. An dieser Stelle geht es nur um einige abstrakte Aspekte der Objekte, für die das StarOffice API einige zentrale Schnittstellen zur Verfügung stellt.

Erzeugen kontextabhängiger Objekte

Das StarOffice API bietet zwei Möglichkeiten zur Erzeugung von Objekten. Eine besteht in der bereits zu Beginn dieses Kapitels genannten Funktion `createUnoService`. Sie erzeugt ein Objekt, das universell einsetzbar ist. Solche Objekte beziehungsweise Services werden auch als *kontextunabhängige Services* bezeichnet.

Neben den kontextunabhängigen Services gibt es *kontextabhängige Services*, deren Objekte nur im Zusammenhang mit einem anderen Objekt Sinn machen. So kann ein Zeichenobjekt für ein Spreadsheet-Dokument beispielsweise nur im Zusammenhang mit eben diesem Dokument existieren.

Schnittstelle com.sun.star.lang.XMultiServiceFactory

Die Erzeugung kontextabhängiger Objekte erfolgt in der Regel über eine Methode des Objektes, von dem es abhängig ist. Insbesondere in den Dokument-Objekten kommt dabei häufig die Methode `createInstance` zum Einsatz, die in der Schnittstelle `XMultiServiceFactory` definiert ist.

Das oben erwähnte Zeichenobjekt lässt sich beispielsweise wie folgt über ein Tabellendokument-Objekt erzeugen:

```
Dim RectangleShape As Object

RectangleShape = _
      Spreadsheet.createInstance("com.sun.star.drawing.RectangleShape")
```

Analog erfolgt die Erzeugung einer Absatzvorlage in einem Textdokument:

```
Dim Style as Object
Style = Textdocument.createInstance("com.sun.star.style.ParagraphStyle")
```

Benannter Zugriff auf untergeordnete Objekte

Die Schnittstellen `XNameAccess` und `XNameContainer` kommen in Objekten zum Einsatz, die untergeordnete Objekte enthalten, die über einen Klartextnamen angesprochen werden können.

Während `XNamedAccess` den Zugriff auf die einzelnen Objekte gestattet, übernimmt `XNameContainer` das Einfügen, Ändern und Löschen von Elementen.

Schnittstelle com.sun.star.container.XNameAccess

Ein Beispiel für den Einsatz von `XNameAccess` liefert das Sheet-Objekt eines Tabellendokuments. Es fasst sämtliche Blätter innerhalb des Tabellendokuments zusammen. Der Zugriff auf die einzelnen Blätter erfolgt über die Methode `getByName` von `XNameAccess`:

```
Dim Sheets As Object
Dim Sheet As Object

Sheets = Spreadsheet.Sheets
Sheet = Sheets.getByName("Sheet1")
```

Einen Überblick zu den Namen sämtlicher Elemente eines Objektes liefert die Methode `getElementNames`. Als Ergebnis liefert sie ein Datenfeld mit den Namen. Das nachfolgende Beispiel zeigt, wie sich damit alle Elementnamen eines Tabellendokuments ermitteln und in einer Schleife ausgeben lassen:

```
Dim Sheets As Object
Dim SheetNames
Dim I As Integer

Sheets = Spreadsheet.Sheets
SheetNames = Sheets.getElementNames

For I=LBound(SheetNames) To UBound(SheetNames)
      MsgBox SheetNames(I)
Next I
```

Die Methode `hasByName` der Schnittstelle `XNameAccess` verrät, ob ein untergeordnetes Objekt mit einem bestimmten Namen innerhalb des Basisobjektes vorhanden ist. So gibt das folgende Beispiel eine Meldung darüber aus, ob das Objekt `Spreadsheet` eine Seite mit dem Namen `Sheet1` enthält.

```
Dim Sheets As Object

Sheets = Spreadsheet.Sheets
If Sheets.HasByName("Sheet1") Then
      MsgBox "Sheet1 vorhanden"
Else
      MsgBox "Sheet1 nicht vorhanden"
End If
```

Schnittstelle com.sun.star.container.XNameContainer

Die Schnittstelle `XNameContainer` übernimmt das Einfügen, Löschen und Ändern von untergeordneten Elementen in einem Basisobjekt. Die zuständigen Funktionen lauten `insertByName`, `removeByName` und `replaceByName`.

Das folgende Beispiel verdeutlicht den Einsatz in der Praxis anhand eines Textdokumentes, das ein Objekt `StyleFamilies` enthält und über dieses wiederum die Absatzvorlagen (Paragraph-Styles) des Dokumentes zur Verfügung stellt.

```
Dim StyleFamilies As Objects
Dim ParagraphStyles As Objects
Dim NewStyle As Object

StyleFamilies = Textdoc.StyleFamilies
ParagraphStyles = StyleFamilies.getByName("ParagraphStyles")

ParagraphStyles.insertByName("NewStyle", NewStyle)
ParagraphStyles.replaceByName("ChangingStyle", NewStyle)
ParagraphStyles.removeByName("OldStyle")
```

Die Zeile `insertByName` fügt den Style `NewStyle` unter dem gleichnamigen Namen in das Objekt `ParagraphStyles` ein. Die Zeile `replaceByName` ändert das hinter `ChangingStyle` liegende Objekt in `NewStyle`. Der Aufruf `removeByName` entfernt schließlich das hinter `OldStyle` stehende Objekt aus `ParagraphStyles`.

Index-basierter Zugriff auf untergeordnete Objekte

Die Schnittstellen `XIndexAccess` und `XIndexContainer` kommen in Objekten zum Einsatz, die untergeordnete Objekte enthalten und über einen Index angesprochen werden können.

`XIndexAccess` bietet die Methoden für den Zugriff auf einzelne Objekte an. `XIndexContainer` liefert Methoden zum Einfügen und Entfernen von Elementen zurück.

Schnittstelle com.sun.star.container.XIndexAccess

`XIndexAccess` bietet zum Auslesen der untergeordneten Objekte die Methoden `getByIndex` sowie `getCount` an. `getByIndex` liefert ein Objekt mit einem bestimmten Index zurück. `getCount` verrät unterdessen, wie viele Objekte prinzipiell vorhanden sind.

```
Dim Sheets As Object
Dim Sheet As Object
Dim I As Integer

Sheets = Spreadsheet.Sheets

For I = 0 to Sheets.getCount() - 1
      Sheet = Sheets.getByIndex(I)
      ' Bearbeitung von Sheet
Next I
```

Das Beispiel zeigt eine Schleife, die sämtliche Elemente von `Sheets` nacheinander durchläuft und in der Objektvariable `Sheet` jeweils einen Verweis darauf ablegt. Beim Umgang mit den Indizes ist darauf zu achten, dass `getCount` die Anzahl der Elemente liefert, die Elemente in `getByIndex` allerdings von `0` beginnend durchnummeriert werden. Die Zählvariable der Schleife läuft daher von `0` bis `getCount()-1`.

Schnittstelle com.sun.star.container.XIndexContainer

Die Schnittstelle `XIndexContainer` stellt die Funktionen `insertByIndex` sowie `removeByIndex` zur Verfügung. Der Aufbau der Parameter erfolgt analog zu den entsprechenden Funktionen in `XNameContainer`.

Iterativer Zugriff auf untergeordnete Objekte

In einigen Fällen enthält ein Objekt zwar eine Liste von untergeordneten Objekten, diese sind jedoch weder über einen Namen noch über einen Index adressierbar. In diesen Situationen sind die Schnittstellen `XEnumeration` und `XenumerationAccess` zweckmäßig. Sie stellen einen Mechanismus zur Verfügung, über den sämtliche untergeordnete Elemente eines Objektes Schritt für Schritt durchwandert werden können, ohne dass eine Möglichkeit zur direkten Adressierung existieren muss.

Schnittstelle com.sun.star.container.XEnumeration und XEnumerationAccess

Das Basisobjekt muss die Schnittstelle `XEnumerationAccess` anbieten, die lediglich eine Methode `createEnumeration` bereit hält. Diese liefert ein Hilfsobjekt zurück, das wiederum die Schnittstelle `XEnumeration` mit den Methoden `hasMoreElements` und `nextElement` anbietet. Über diese haben Sie Zugriff auf die untergeordneten Objekte.

Der Einsatz dieses Konzeptes in der Praxis ist einfacher als es auf den ersten Blick scheint. Dies verdeutlicht ein Beispiel, das sämtliche Absätze eines Textes durchwandert:

```
Dim ParagraphEnumeration As Object
Dim Paragraph As Object

ParagraphEnumeration = Textdoc.Text.createEnumeration

While ParagraphEnumeration.hasMoreElements()
      Paragraph = ParagraphElements.nextElement()
Wend
```

Das Beispiel erzeugt zunächst ein Hilfsobjekt `ParagraphEnumeration`. Dieses liefert in einer Schleife nach und nach die einzelnen Absätze des Textes zurück. Die Schleife bricht ab, sobald die Methode `hasMoreElements` den Wert `False` zurückliefert und damit signalisiert, dass das Ende des Textes erreicht ist.

KAPITEL 5

Die Arbeit mit StarOffice-Dokumenten

Das StarOffice API wurde so aufgebaut, dass möglichst viele Teile universell für verschiedene Aufgabenstellungen eingesetzt werden können. Dazu zählen beispielsweise die Schnittstellen und Services zum Erzeugen, Öffnen, Speichern, Konvertieren und Drucken von Dokumenten und die Vorlagenverwaltung. Da diese Funktionsbereiche in allen Dokumentarten zur Verfügung stehen, werden sie in diesem Kapitel vorab erklärt.

Der StarDesktop

Beim Umgang mit Dokumenten kommen im wesentlichen zwei Services zum Einsatz:

- Der Service `com.sun.star.frame.Desktop`, der so etwas wie den Stammservice von StarOffice bildet. Er stellt die Funktionen für das Rahmenobjekt von StarOffice zur Verfügung, unter dem sich alle Dokumentfenster einreihen. Auch das Erzeugen, Öffnen und Importieren von Dokumenten läuft über diesen Service.
- Die Basisfunktionalität für die einzelnen Dokument-Objekte liefert der Service `com.sun.star.document.OfficeDocument`. Er bietet die Methoden zum Speichern, Exportieren und Drucken von Dokumenten.

Der Service `com.sun.star.frame.Desktop` steht mit dem Start von StarOffice automatisch zur Verfügung. Dazu erzeugt StarOffice ein Objekt, das über den globalen Namen `StarDesktop` erreichbar ist.

Die wichtigste Schnittstelle des `StarDesktop` lautet `com.sun.star.frame.XComponentLoader`. Sie umfasst im wesentlichen die Methode `loadComponentFromURL`, die für das Erzeugen, Importieren und Öffnen von Dokumenten zuständig ist.

> Der Name des Objektes `StarDesktop` geht auf StarOffice 5 zurück, bei dem alle Dokumentfenster in eine gemeinsame Anwendung eingebettet waren, die StarDesktop hieß. In der aktuellen Version von StarOffice gibt es keinen sichtbaren StarDesktop mehr. Der Name StarDesktop für das Rahmenobjekt von StarOffice wurde jedoch beibehalten, da er anschaulich klar macht, dass es sich um ein Basisobjekt der gesamten Anwendung handelt.
>
> Das Objekt `StarDesktop` tritt die Nachfolge des `Application`-Objektes von StarOffice 5 an, das bisher als Wurzelobjekt galt. Es ist im Gegensatz zu dem alten `Application`-Objekt jedoch primär für das Öffnen neuer Dokumente verantwortlich. Die im alten `Application`-Objekt angesiedelten Funktionen zur Steuerung der Bildschirmdarstellung von StarOffice (z.B. `FullScreen`, `FunctionBarVisible`, `Height`, `Width`, `Top`, `Visible`) sind weggefallen.

Während der Zugriff auf das aktive Dokument in Word über `Application.ActiveDocument` und in Excel über `Application.ActiveWorkbook` erfolgt, ist in StarOffice der `StarDesktop` für diese Aufgabe verantwortlich. Der Zugriff auf das aktive Dokument-Objekt erfolgt in StarOffice 7 über die Eigenschaft `StarDesktop.CurrentComponent`.

Grundlegendes zu Dokumenten in StarOffice

Bei der Arbeit mit StarOffice-Dokumenten ist es hilfreich, sich mit einigen grundlegenden Dingen der Dokumentverwaltung von StarOffice zu beschäftigen. Hierzu zählt die Art und Weise, wie Dateinamen für StarOffice-Dokumente aufgebaut sind sowie das Format, in dem die Dateien abgelegt werden.

Dateinamen in URL-Notation

Da StarOffice als plattformunabhängige Anwendung konzipiert wurde, verwendet es für Dateinamen die betriebssystemunabhängige URL-Notation gemäß dem Internet-Standard RFC 1738. Gewöhnliche Dateinamen beginnen dabei mit dem Prefix

```
file:///
```

gefolgt von dem lokalen Pfad. Enthält der Dateiname Unterverzeichnisse, so werden diese mit *einfachen* Schrägstrichen getrennt (also nicht mit dem unter Windows üblichen *rückwärtsgewandten* Schrägstrich). Der folgende Pfad verweist auf die Datei `test.sxw` im Verzeichnis `doc` auf dem Laufwerk `C:`.

```
file:///C:/doc/test.sxw
```

Zur Umwandlung von lokalen Dateinamen in eine URL bietet StarOffice die Funktion `ConvertToUrl` an. Für die Rückumwandlung einer URL in einen lokalen Dateinamen hält StarOffice die Funktion `ConvertFromUrl` bereit:

```
MsgBox ConvertToUrl("C:\doc\test.sxw")
            ' liefert file:///C:/doc/test.sxw

MsgBox ConvertFromUrl("file:///C:/doc/test.sxw")
            ' liefert (unter Windows) c:\doc\test.sxw
```

Das Beispiel konvertiert einen lokalen Dateinamen in eine URL und gibt diesen in einem Meldungsfenster aus. Danach konvertiert es eine URL in einen lokalen Dateinamen und gibt diesen ebenfalls aus.

Der zugrunde liegende Internet-Standard RFC 1738 gestattet die Verwendung der Zeichen `0-9`, `a-z` und `A-Z`. Alle anderen Zeichen werden in einer Escape-Kodierung in die URLs eingefügt. Dazu werden sie in ihren Hexadezimalwert im Zeichensatz ISO 8859-1 (ISO-Latin) umgewandelt und bekommen ein Prozentzeichen voran gestellt. Aus einem Leerzeichen in einem lokalen Dateinamen wird so beispielsweise ein `%20` in der URL.

XML-Dateiformat

Seit der Version 6.0 wird in StarOffice ein XML-basiertes Dateiformat verwendet. Durch den Einsatz von XML hat der Anwender die Möglichkeit, die Dateien auch mit anderen Programmen zu öffnen und zu bearbeiten.

Komprimierung der Dateien

Da XML auf gewöhnlichen Textdateien basiert, sind die sich ergebenden Dateien im Regelfall recht groß. StarOffice komprimiert die Dateien daher und speichert sie als ZIP-Datei ab.

Über eine Option hat der Anwender jedoch die Möglichkeit, die Original-XML-Dateien direkt abzuspeichern (siehe Abschnitt „Die Optionen der Methode `storeAsURL`“).

Dokumente erzeugen, öffnen und importieren

Das Öffnen, Importieren und Erzeugen von Dokumenten erfolgt über die Methode

```
StarDesktop.loadComponentFromURL(URL, Frame, _
                        SearchFlags, FileProperties)
```

Der erste Parameter von `loadComponentFromURL` gibt die URL der betreffenden Datei an.

Als zweiten Parameter erwartet `loadComponentFromURL` einen Namen für das Rahmenobjekt des Fensters, das StarOffice intern zu dessen Verwaltung erzeugt. Im Regelfall wird hier der vordefinierte Name `_blank` angegeben, der dafür sorgt, dass StarOffice ein neues Fenster erzeugt. Alternativ ist auch die Angabe `_hidden` möglich, die dafür sorgt, dass das betreffende Dokument zwar geladen wird, aber unsichtbar bleibt.

Mit den genannten Parametern ist es bereits möglich, ein StarOffice-Dokument zu öffnen. Denn die letzten beiden Parameter können mit Platzhaltern (Dummy-Werten) belegt werden:

```
Dim Doc As Object
Dim Url As String
Dim Dummy()

Url = "file:///C:/test.sxw"

Doc = StarDesktop.loadComponentFromURL(Url, "_blank", 0, Dummy())
```

Der genannte Aufruf öffnet die Datei text.sxw und zeigt sie in einem neuen Fenster an.

Auf diese Art lassen sich beliebig viele Dokumente in StarOffice Basic öffnen und anschließend über die jeweils zurück gelieferten Dokument-Objekte bearbeiten.

`StarDesktop.loadComponentFromURL` löst die Methoden `Documents.Add` und `Documents.Open` aus dem alten StarOffice API ab.

Inhalte des Dokumentfensters austauschen

Die genannten Werte `_blank` und `_hidden` für den Parameter `Frame` sorgen dafür, dass StarOffice für jeden Aufruf von `loadComponentFromURL` ein neues Fenster erzeugt. In einigen Situationen ist es jedoch sinnvoll, den Inhalt eines vorhandenen Fensters auszutauschen. In diesem Fall sollte das Rahmenobjekt des Fensters einen expliziten Namen erhalten (darf nicht mit Unterstrich beginnen). Des weiteren muss der Parameter `SearchFlags` so gesetzt sein, dass der betreffende Rahmen erzeugt wird, falls er noch nicht vorhanden ist. Die betreffende Konstante für `SearchFlags` lautet:

```
SearchFlags = com.sun.star.frame.FrameSearchFlag.CREATE + _
                com.sun.star.frame.FrameSearchFlag.ALL
```

Das nachfolgende Beispiel zeigt, wie sich mit Hilfe der Parameter `Frame` und `SearchFlags` der Inhalt eines geöffneten Fensters austauschen lässt:

```
Dim Doc As Object
Dim Dummy()
Dim Url As String
Dim SearchFlags As Long

SearchFlags = com.sun.star.frame.FrameSearchFlag.CREATE + _
                com.sun.star.frame.FrameSearchFlag.ALL

Url = "file:///C:/test.sxw"
Doc = StarDesktop.loadComponentFromURL(Url, "MyFrame", _
            SearchFlags, Dummy)

MsgBox "Drücken Sie Ok, um das zweite Dokument anzuzeigen."

Url = "file:///C:/test2.sxw"
Doc = StarDesktop.loadComponentFromURL(Url, "MyFrame", _
            SearchFlags, Dummy)
```

Das Beispiel öffnet zunächst die Datei `test.sxw` in einem neuen Fenster mit dem Rahmennamen `MyFrame`. Nach dem Bestätigen des Meldungsfensters tauscht es den Inhalt des Fensters durch die Datei `test2.sxw` aus.

Die Optionen der Methode loadComponentFromURL

Bleibt noch der vierte Parameter der Funktion `loadComponentFromURL`, für den bisher ein Platzhalter zum Einsatz kam. Bei ihm handelt es sich um ein `PropertyValue`-Datenfeld, das StarOffice verschiedene Optionen für das Öffnen beziehungsweise Erzeugen von Dokumenten mit gibt. Für jede Option muss das Datenfeld eine `PropertyValue`-Struktur bereit halten, in der der Name der Option als Zeichenfolge sowie der zugehörige Wert abgelegt werden.

`loadComponentFromURL` unterstützt folgende Optionen:

- **`AsTemplate (Boolean)`** – lädt eine Vorlage direkt, wenn `AsTemplate` auf den Wert `True` gesetzt ist. Ansonsten bewirkt das Laden einer Vorlage das Erzeugen eines neuen Dokumentes basierend auf der Vorlage.
- **`CharacterSet (String)`** – legt fest, welcher Zeichensatz einem Dokument zugrunde liegt.
- **`FilterName (String)`** – gibt einen speziellen Filter für die Funktion `loadComponentFromURL` vor. Die möglichen Filternamen sind in der Datei `\share\config\registry\instance\org\openoffice\office\TypeDetection.xml`definiert.
- **`FilterOptions (String)`** – legt Zusatzoptionen für Filter fest.
- **`JumpMark (String)`** – springt nach dem Öffnen eines Dokumentes zu der in `JumpMark` definierten Position.
- **`Password (String)`** – übergibt ein Passwort für das Öffnen einer geschützten Datei.
- **`ReadOnly (Boolean)`** – lädt ein Dokument mit Schreibschutz.

Das nachfolgende Beispiel zeigt, wie sich eine Komma-getrennte Textdatei in StarOffice Calc unter Verwendung der Option `FilterName` öffnen lässt.

```
Dim Doc As Object
Dim FileProperties(0) As New com.sun.star.beans.PropertyValue
Dim Url As String

Url = "file:///C:/csv.doc"

FileProperties(0).Name = "FilterName"
FileProperties(0).Value ="scalc: Text - txt - csv (StarOffice Calc)"

Doc = StarDesktop.loadComponentFromURL(Url, "_blank", 0, FileProperties())
```

Das Datenfeld `FileProperties` umfasst genau einen Wert, da es eine Option aufnehmen soll. Das Attribut `Filtername` legt hierbei fest, dass StarOffice einen StarOffice Calc-Textfilter für das Öffnen der Datei verwenden soll.

Neue Dokumente erzeugen

Wie bereits im vorherigen Abschnitt erwähnt, erzeugt StarOffice automatisch ein neues Dokument, falls es sich bei dem in der URL angegebenen Dokument um eine Vorlage handelt.

Wird lediglich ein leeres Dokument ohne jegliche Anpassungen benötigt, so lässt sich alternativ eine `private:factory`-URL angeben:

```
Dim Dummy()
Dim Url As String
Dim Doc As Object

Url = "private:factory/swriter"
Doc = StarDesktop.loadComponentFromURL(Url, "_blank", 0, Dummy())
```

Der Aufruf erzeugt ein leeres StarOffice Writer-Dokument.

Dokument-Objekte

Die in den vorstehenden Absätzen eingeführte Funktion `loadComponentFromURL` liefert ein Dokument-Objekt zurück. Dieses unterstützt den Service `com.sun.star.document.OfficeDocument`, der wiederum zwei zentrale Schnittstellen anbietet:

- die Schnittstelle `com.sun.star.frame.XStorable`, die für das Speichern von Dokumenten zuständig ist und
- die Schnittstelle `com.sun.star.view.XPrintable`, die Methoden für das Ausdrucken von Dokumenten bereit hält.

Der Funktionsumfang der Dokumentobjekte ist bei der Umstellung auf StarOffice 7 über weite Strecken gleich geblieben. So stellen die Dokumentobjekte beispielsweise weiterhin Methoden für das Speichern und Drucken von Dokumenten zur Verfügung. Die Namen und Parameter der Methoden haben sich jedoch geändert.

Dokumente speichern und exportieren

Das Speichern von StarOffice-Dokumenten erfolgt direkt über das Dokument-Objekt. Hierzu steht die Methode `store` der Schnittstelle `com.sun.star.frame.XStorable` zur Verfügung:

```
Doc.store()
```

Dieser Aufruf funktioniert nur, wenn dem Dokument bereits ein Speicherplatz zugewiesen wurde. Bei neuen Dokumenten ist dies nicht der Fall. Hier kommt daher die Methode `storeAsURL` zum Einsatz, die ebenfalls in `com.sun.star.frame.XStorable` definiert ist und über die sich der Standort des Dokumentes festlegen lässt:

```
Dim URL As String
Dim Dummy()

Url = "file:///C:/test3.sxw"

Doc.storeAsURL(URL, Dummy())
```

Neben den genannten Methoden stellt `com.sun.star.frame.XStorable` einige Hilfsmethoden zur Verfügung, die beim Speichern von Dokumenten nützlich sein können. Dies sind:

- **`hasLocation()`** – gibt an, ob dem Dokument bereits eine URL zugewiesen wurde.
- **`isReadonly()`** - gibt an, ob ein Dokument schreibgeschützt ist.
- **`isModified()`** - gibt an, ob ein Dokument nach dem letzten Speichervorgang geändert wurde

Mit diesen Prüfmöglichkeiten lässt sich der oben stehende Code zum Speichern eines Dokumentes ausbauen, so dass die Speicherung nur dann erfolgt, wenn das Objekt tatsächlich geändert wurde und der Dateiname nur dann abgefragt wird, wenn er auch wirklich benötigt wird:

```
If (Doc.isModified) Then
      If (Doc.hasLocation And (Not Doc.isReadOnly)) Then
            Doc.store()
      Else
            Doc.storeAsURL(URL, Dummy())
      End If
End If
```

Das Beispiel prüft zunächst, ob das betreffende Dokument seit dem letzten Speichervorgang geändert wurde. Nur falls dies der Fall ist, wird der Speichervorgang fortgesetzt. Wurde dem Dokument bereits eine URL zugeordnet und besitzt es keinen Schreibschutz, so wird es unter der vorhandenen URL gespeichert. Besitzt es keine URL beziehungsweise wurde es mit Schreibschutz geöffnet, so wird es unter einer neuen URL gespeichert.

Die Optionen der Methode storeAsURL

Wie bei der Methode `loadComponentFromURL` lassen sich auch bei der Methode `storeAsURL` einige Optionen in Form eines `PropertyValue`-Datenfeld angeben. Diese bestimmen das Vorgehen von StarOffice bei der Speicherung eines Dokumentes. Für `storeAsURL` stehen unter anderem folgende Optionen zur Verfügung:

- **`CharacterSet (String)`** – legt fest, welcher Zeichensatz einem Dokument zugrunde liegt.
- **`FilterName (String)`** – gibt einen speziellen Filter für die Funktion `loadComponentFromURL` vor. Die möglichen Filternamen sind in der Datei `\share\config\registry\instance\org\openoffice\office\TypeDetection.xml` definiert.
- **`FilterOptions (String)`** – legt Zusatzoptionen für Filter fest.
- **`Overwrite (Boolean)`** – erlaubt das Überschreiben einer bereits existierenden Datei ohne Rückfrage.
- **`Password (String)`** – übergibt das Passwort für eine geschützte Datei.
- **`Unpacked (Boolean)`** – speichert das Dokument unkomprimiert in Unterverzeichnissen ab.

Das nachfolgende Beispiel zeigt, wie die Option `Overwrite` zusammen mit `storeAsURL` verwendet werden kann:

```
Dim Doc As Object
Dim FileProperties(0) As New com.sun.star.beans.PropertyValue
Dim Url As String

' ... Doc initialisieren

Url = "file:///c:/test3.sxw"

FileProperties(0).Name = "Overwrite"
FileProperties(0).Value = True

Doc.storeAsURL(sUrl, mFileProperties())
```

Das Beispiel speichert `Doc` auch dann unter dem angegebenen Dateinamen ab, wenn unter dem Namen bereits eine Datei existiert.

Dokumente drucken

Ähnlich wie das Speichern, erfolgt auch das Ausdrucken von Dokumenten direkt über das Dokumentobjekt. Dazu dient die Methode `Print` der Schnittstelle `com.sun.star.view.XPrintable`. In seiner einfachsten Form lautet der Aufruf von `print`:

```
Dim Dummy()

Doc.print(Dummy())
```

Bei dem Parameter `Dummy` handelt es sich wie im Falle der Methode `loadComponentFromURL` um ein `PropertyValue`-Datenfeld, über das StarOffice einige Optionen für das Ausdrucken spezfieren kann.

Die Optionen der Methode print

Die Methode `print` erwartet ein `PropertyValue`-Datenfeld als Parameter, das die Einstellungen des Druck-Dialoges von StarOffice wiederspiegelt:

- **`CopyCount (Integer)`** – gibt die Anzahl der auszudruckenden Kopien an.
- **`FileName (String)`** – druckt das Dokument in die genannte Datei.
- **`Collate (Boolean)`** – druckt doppelseitig.
- **`Sort (Boolean)`** – sortiert die Seiten beim Ausdruck mehrerer Exemplare (`CopyCount` > 1).
- **`Pages (String)`** – enthält die Auflistung der auszudruckenden Seiten (Syntax wie über den Druck-Dialog)

Das nachfolgende Beispiel zeigt, wie sich über die Option `Pages` mehrere Seiten (Nummer 1 bis 3, 7 und 9) eines Dokumentes ausdrucken lassen:

```
Dim Doc As Object
Dim PrintProperties(0) As New com.sun.star.beans.PropertyValue

PrintProperties(0).Name="Pages"
PrintProperties(0).Value="1-3; 7; 9"

Doc.print(PrintProperties())
```

Druckerauswahl und -einstellungen

Zur Auswahl des für einen Ausdruck zu verwendenden Druckers stellt die Schnittstelle `com.sun.star.view.XPrintable` eine Eigenschaft `Printer` zur Verfügung, die ein `PropertyValue`-Datenfeld mit folgenden Einstellungen entgegen nimmt:

- **`Name (String)`** – Name des Druckers.
- **`PaperOrientation (Enum)`** – Seitenausrichtung (Wert `com.sun.star.view.PaperOrientation.PORTRAIT` für Hochformat, `com.sun.star.view.PaperOrientation.LANDSCAPE` für Querformat).
- **`PaperFormat (Enum)`** – Papierformat (z.B. com.sun.star.view.PaperFormat.A4 für DIN A4 oder `com.sun.star.view.PaperFormat.Letter` für US-Letter).
- **`PaperSize (Size)`** – Papiergröße in hundertstel Millimeter.

Das nachfolgende Beispiel zeigt, wie sich mit Hilfe der `Printer`-Eigenschaft ein Drucker wechseln und die Papiergröße einstellen lässt.

```
Dim Doc As Object
Dim PrinterProperties(1) As New com.sun.star.beans.PropertyValue
Dim PaperSize As New com.sun.star.awt.Size

PaperSize.Width = 20000       ' entspricht 20 cm
PaperSize.Height = 20000' entspricht 20 cm

PrinterProperties (0).Name="Name"
PrinterProperties (0).Value="My HP Laserjet"

PrinterProperties (1).Name="PaperSize"
PrinterProperties (1).Value=PaperSize

Doc.Printer = PrinterProperties()
```

Das Beispiel definiert ein Objekt namens `PaperSize` mit dem Typ `com.sun.star.awt.Size`. Dies wird zur Angabe der Papiergröße benötigt. Des weiteren erstellt es ein Datenfeld für zwei `PropertyValue`-Einträge namens `PrinterProperties`. Dieses Datenfeld wird anschließend mit den zu setzenden Werten initialisiert und der Eigenschaft `Printer` zugewiesen (Aus UNO-Sicht ist Printer keine echte, sondern eine *nachgebildete* Eigenschaft).

Vorlagen

Vorlagen sind benamte Listen mit Formatierungsattributen. Sie ziehen sich durch alle Anwendungen von StarOffice und tragen zu einer deutlichen Vereinfachung von Formatierungen bei. Ändert der Anwender eines der Vorlagen-Attribute, so passt StarOffice automatisch alle davon abhängigen Dokumentteile an. Dadurch ist es beispielsweise möglich, die Schriftart sämtlicher Überschriften der Ebene eins durch eine zentrale Änderung im Dokument anzupassen. Abhängig von den jeweiligen Dokumenttypen kennt StarOffice eine ganze Reihe verschiedener Vorlagenarten.

StarOffice Writer unterstützt

- Zeichenvorlagen,
- Absatzvorlagen,
- Rahmenvorlagen,
- Seitenvorlagen
- Nummerierungsvorlagen

StarOffice Calc unterstützt

- Zellvorlagen
- Seitenvorlagen

StarOffice Impress unterstützt

- Zeichenelement-Vorlagen
- Präsentationsvorlagen

Die verschiedenen Vorlagenarten heißen in der StarOffice-Terminologie `StyleFamilies` gemäß des zugrunde liegenden Services `com.sun.star.style.StyleFamily`. Der Zugriff auf die `StyleFamilies` erfolgt über das Dokument-Objekt:

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim CellStyles As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
CellStyles = StyleFamilies.getByName("CellStyles")
```

Das Beispiel ermittelt über die Eigenschaft `StyleFamilies` eines Spreadsheet-Dokumentes eine Liste mit sämtlichen vorhandenen Zellvorlagen.

Der Zugriff auf die Einzelvorlagen ist unmittelbar über einen Index möglich:

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim CellStyles As Object
Dim CellStyle As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
CellStyles = StyleFamilies.getByName("CellStyles")

For I = 0 To CellStyles.Count - 1
      CellStyle = CellStyles(I)
      MsgBox CellStyle.Name
Next I
```

Die gegenüber dem vorigen Beispiel hinzu gekommene Schleife gibt die Namen sämtlicher Zellvorlagen nacheinander in einem Meldungsfenster aus.

Details zu den verschiedenen Formatierungsmöglichkeiten

Jede Vorlagenart stellt eine ganze Reihe individueller Formatierungsattribute zur Verfügung. Da die Attribute nicht nur auf Vorlagen, sondern auch direkt auf die Dokumentinhalte angewendet werden können, werden sie bei den verschiedenen Dokumentarten erklärt. Hier ein Überblick zu den wichtigsten Formatierungsattributen und den Stellen, an denen sie erklärt werden:

- Zeichenattribute, Kapitel 6, Textdokumente,
  Service `com.sun.star.style.CharacterProperties`
- Absatzattribute, Kapitel 6, Textdokumente,
  Service `com.sun.star.text.Paragraph`
- Zellenattribute, Kapitel 7, Tabellendokumente,
  Service `com.sun.star.table.CellProperties`
- Seitenattribute, Kapitel 7, Tabellendokumente,
  Service `com.sun.star.style.PageStyle`
- Zeichenelementattribute, Kapitel 7, Tabellendokumente,
  verschiedene Services

Die Formatattribute sind im Regelfall keineswegs auf die Anwendungen beschränkt, in denen sie erklärt sind, sondern viel mehr universell einsetzbar. So lassen sich die meisten der in Kapitel 7 beschriebenen Seitenattribute beispielsweise nicht nur in StarOffice Calc einsetzen, sondern auch in StarOffice Writer.

Weitergehende Informationen für den Umgang mit Vorlagen finden Sie im Abschnitt *Vorgabewerte für Zeichen und Absatzattribute* in Kapitel 6, Textdokumente.

KAPITEL 6

Textdokumente

Neben der reinen Zeichenfolge enthalten Textdokumente Formatierungsinformationen. Diese können an jeder beliebigen Stelle des Textes auftreten. Weiter verkompliziert wird der Aufbau durch Tabellen. Diese bestehen nicht etwa aus eindimensionalen Zeichenfolgen, sondern aus zweidimensionalen Feldern. Schließlich bieten moderne Textverarbeitungsprogramme die Möglichkeit, Zeichenobjekte, Textrahmen und andere Objekte innerhalb eines Textes zu platzieren. Diese können außerhalb des Textflusses stehen und frei auf einer Seite positioniert werden.

StarOffice stellt leistungsfähige Schnittstellen für den Umgang mit Textdokumenten zur Verfügung. Im ersten Abschnitt geht es um die Anatomie von Textdokumenten. Hier dreht sich alles darum, wie man mit Hilfe eines StarOffice Basic-Programms Schritt für Schritt durch ein StarOffice Dokument iterieren kann. Im Mittelpunkt stehen dabei Absätze, Absatzteile und deren Formatierung.

Im zweiten Abschnitt steht die effiziente Arbeit mit Textdokumenten im Vordergrund. Hierzu bietet StarOffice über die im ersten Abschnitt genannten Objekte hinaus einige Hilfsobjekte an (etwa das `TextCursor`-Objekt).

Der dritte Abschnitt geht über die Arbeit mit Texten hinaus. Er dreht sich um Tabellen, Textrahmen, Textfelder, Lesezeichen, Inhaltsverzeichnisse und einiges mehr.

Informationen zum Erzeugen, Öffnen, Speichern und Drucken von Dokumenten sind in Kapitel 5 beschrieben, da sie nicht nur für Textdokumente, sondern auch für andere Dokumentarten herangezogen werden können.

Der Aufbau von Textdokumenten

Ein Textdokument kann im Wesentlichen vier Arten von Informationen enthalten:

- den eigentlichen Text
- Vorlagen für die Formatierung von Zeichen, Absätzen, Seiten und ähnlichem
- nichttextuelle Elemente wie Tabellen, Grafiken und Zeichenobjekte
- globale Einstellungen für das Textdokument

In diesem Abschnitt geht es insbesondere um den Text und die zugehörigen Formatierungsmöglichkeiten.

Das API von StarOffice 7 für StarOffice Writer unterscheidet sich konzeptionell von dem der Vorgängerversion. Im alten API stand die Arbeit mit dem `Selection`-Objekt im Vordergrund, das sich stark an der Idee der Benutzerschnittstelle für Endanwender orientierte und bei dem die mausgesteuerte Arbeit mit Markierungen im Mittelpunkt stand.

Das API von StarOffice 7 löst diese Verbindung zwischen Benutzerschnittstelle und Programmierschnittstelle auf. Der Programmierer hat so parallelen Zugriff auf alle Teile einer Anwendung und kann mit Objekten für verschiedene Teilbereiche eines Dokumentes gleichzeitig arbeiten. Das alte `Selection`-Objekt steht nicht mehr zur Verfügung.

Absätze und Absatzteile

Der Kern eines Textdokumentes besteht aus einer Folge von Absätzen. Sie sind weder benannt noch indiziert, so dass es keine Möglichkeit zum *direkten* Zugriff auf einzelne Absätze gibt. Die Absätze lassen sich jedoch mit Hilfe des in Kapitel 4 beschriebenen `Enumeration`-Objektes sequenziell durchlaufen. Schritt für Schritt liefert es in einer Schleife einen Absatz nach dem anderen zurück, so dass die Absätze bearbeitet werden können.

Bei der Arbeit mit dem `Enumeration`-Objekt ist jedoch ein Sonderfall zu beachten: Es liefert nicht nur Absätze zurück, sondern auch Tabellen (streng genommen handelt es sich in StarOffice Writer bei einer Tabelle um einen speziellen Absatztyp). Daher muss vor einem Zugriff auf das zurück gelieferte Objekt geprüft werden, ob es den Service `com.sun.star.text.Paragraph` für Absätze oder den Service `com.sun.star.text.TextTable` für Tabellen unterstützt.

Das folgende Beispiel durchläuft den Inhalt eines Textdokumentes in einer Schleife und teilt jeweils in einer Meldung mit, ob es sich um einen Absatz oder eine Tabelle handelt.

```
Dim Doc As Object
Dim Enum As Object
Dim TextElement As Object

' Dokument-Objekt erzeugen
Doc = StarDesktop.CurrentComponent

' Enumeration-Objekt erzeugen
Enum = Doc.Text.createEnumeration

' Schleife über alle Text-Elemente
While Enum.hasMoreElements
      TextElement = Enum.nextElement

      If TextElement.supportsService("com.sun.star.text.TextTable") Then
            MsgBox "Der aktuelle Block enthält eine Tabelle."
      End If

      If TextElement.supportsService("com.sun.star.text.Paragraph") Then
            MsgBox "Der aktuelle Block enthält einen Absatz."
      End If
Wend
```

Das Beispiel erzeugt ein Dokument-Objekt `Doc`, das auf das aktuelle Dokument von StarOffice verweist. Mit Hilfe von `Doc` wird anschließend ein `Enumeration`-Objekt erstellt, das die einzelnen Text-Teile (Absätze und Tabellen) nacheinander durchläuft und in dem Objekt `TextElement` ablegt. Über die Methode `supportsService` prüft das Beispiel, ob es sich bei `TextElement` um einen Absatz oder eine Tabelle handelt und gibt dann eine entsprechende Meldung aus.

Absätze

Der Service `com.sun.star.text.Paragraph` gestattet Zugriff auf den Inhalt eines Absatzes. Der innerhalb des Absatzes vorhandene Text kann über die Eigenschaft `String` ausgelesen und geändert werden:

```
Dim Doc As Object
Dim Enum As Object
Dim TextElement As Object

Doc = StarDesktop.CurrentComponent
Enum = Doc.Text.createEnumeration

While Enum.hasMoreElements
      TextElement = Enum.nextElement

      If TextElement.supportsService("com.sun.star.text.Paragraph") Then
            TextElement.String = Replace(TextElement.String, "you", "U")
            TextElement.String = Replace(TextElement.String, "too", "2")
            TextElement.String = Replace(TextElement.String, "for", "4")
      End If
Wend
```

Das Beispiel öffnet das aktuelle Textdokument und durchwandert dieses unter Zuhilfenahme des `Enumeration`-Objektes. In sämtlichen Absätzen greift es über die Eigenschaft `TextElement.String` auf die jeweiligen Absätze zu und ersetzt die Zeichenfolgen `you`, `too` und `for` durch die Zeichen `U`, `2` und `4`. Die für das Ersetzen verwendete Funktion `Replace` gehört nicht zum Standardsprachumfang von StarOffice Basic. Es handelt sich hierbei um die in Kapitel 3 im Abschnitt *Suchen und Ersetzen* beschriebene Beispielfunktion.

Das hier beschriebene Verfahren für den Zugriff auf die Absätze eines Textes ist inhaltlich vergleichbar mit der `Paragraphs`-Auflistung von VBA, die in den dortigen `Range`- und `Document`-Objekten zur Verfügung stehen. Während in VBA der Zugriff auf die Absätze über deren Nummer erfolgt (z.B. über den Aufruf `Paragraph(1)`), ist in StarOffice Basic das oben beschriebene `Enumeration`-Objekt zu verwenden.

Für die in VBA vorhandenen Auflistungen `Characters`, `Sentences` und `Words` existiert kein direktes Gegenstück in StarOffice Basic. Man kann jedoch auf einen `TextCursor` ausweichen, der eine Navigation auf der Ebene von Zeichen, Sätzen und Wörtern gestattet (siehe Abschnitt *Der TextCursor*).

Absatzteile

Das oben stehende Beispiel kann zwar den Text wie gewünscht ändern, dabei jedoch teilweise die Formatierung zerstören.

Dies liegt daran, dass sich ein Absatz wiederum aus einzelnen Teilobjekten zusammen setzt. Jedes dieser Teilobjekte enthält seine eigenen Formatierungsinformationen. Enthält ein Absatz beispielsweise in der Mitte ein fett gedrucktes Wort, so wird er in StarOffice durch drei Absatzteile repräsentiert: Den Teil vor dem Fettdruck, das fettgedruckte Wort und den Teil nach dem Fettdruck, der wieder normal dargestellt wird.

Wird der Text des Absatzes nun über die `String`-Eigenschaft des Absatzes geändert, so löscht StarOffice zunächst die alten Absatzteile und fügt danach einen neuen Absatzteil ein. Dabei gehen die Formatierungen der vorherigen Teile zwangsläufig verloren.

Um diesen Effekt zu vermeiden ist es möglich anstatt auf den gesamten Absatz auf die zugehörigen Absatzteile zuzugreifen. Dazu bieten Absätze ein eigenes `Enumeration`-Objekt an. Das folgende Beispiel zeigt eine doppelte Schleife, die über sämtliche Absätze und die darin enthaltenen Absatzteile eines Textdokumentes wandert und die Ersetzungsvorgänge aus dem vorstehenden Beispiel anwendet:

```
Dim Doc As Object
Dim Enum1 As Object
Dim Enum2 As Object
Dim TextElement As Object
Dim TextPortion As Object

Doc = StarDesktop.CurrentComponent
Enum1 = Doc.Text.createEnumeration

' Schleife über alle Absätze
While Enum1.hasMoreElements
      TextElement = Enum1.nextElement
      If TextElement.supportsService("com.sun.star.text.Paragraph") Then
            Enum2 = TextElement.createEnumeration

            ' Schleife über alle Teilabsätze
            While Enum2.hasMoreElements
                  TextPortion = Enum2.nextElement
                  MsgBox "'" & TextPortion.String & "'"
                  TextPortion.String = Replace(TextPortion.String, "you", "U")
                  TextPortion.String = Replace(TextPortion.String, "too", "2")
                  TextPortion.String = Replace(TextPortion.String, "for", "4")
            Wend

      End If
Wend
```

Das Beispiel durchläuft ein Textdokument in einer doppelten Schleife. Die äußere Schleife bezieht sich auf die Absätze des Textes. Die innere Schleife bearbeitet die darin enthaltenen Absatzteile. In jedem dieser Absatzteile ändert der Beispielcode den Inhalt über die `String`-Eigenschaft der Zeichenfolge analog zu dem vorstehenden Beispiel für Absätze. Da jedoch die Absatzteile direkt bearbeitet werden, bleiben ihre Formatierungsinformationen beim Ersetzen der Zeichenfolge erhalten.

Formatierungen

Es gibt verschiedene Arten einen Text zu formatieren. Der einfachste Weg besteht darin, der betreffenden Textsequenz die gewünschten Formatattribute direkt zuzuweisen. Dies wird als *direkte Formatierung* bezeichnet. Solche Formatierungen werden insbesondere bei kurzen Dokumenten verwendet, da sie ohne große Umwege per Mausklick zugewiesen werden können. So

kann man beispielsweise über eine direkte Formatierung ein spezielles Wort innerhalb eines Textes als Fettdruck hervorheben oder eine Zeile zentrieren.

Neben der direkten Formatierung ist eine Formatierung über die Verwendung von Vorlagen möglich. Diese wird als *indirekte Formatierung* bezeichnet. Hierbei weist der Anwender der betreffenden Textstelle anstelle der direkten Formatattribute eine Vorlage zu. Soll das Layout des Textes nachträglich geändert werden, so braucht der Anwender lediglich die Vorlage anzupassen. StarOffice ändert daraufhin die Darstellung sämtlicher Textstellen, bei denen die Vorlage eingesetzt wird.

In VBA sind die Formatierungsattribute eines Objektes im Regelfall auf eine Reihe von Unterobjekten verteilt (z.B. `Range.Font`, `Range.Borders`, `Range.Shading`, `Range.ParagraphFormat`). Der Zugriff auf die Attribute erfolgt über kaskadierte Ausdrücke (z.B. `Range.Font.AllCaps`). In StarOffice Basic stehen die Formatierungsattribute hingegen direkt an den jeweiligen Objekten (`TextCursor`, `Paragraph` usw.) zur Verfügung. Einen Überblick zu den verfügbaren Zeichen- und Absatzattributen von StarOffice finden Sie in den folgenden beiden Unterkapiteln.

Im alten StarOffice API erfolgte die Formatierung eines Textes im wesentlichen über das `Selection`-Objekt und dessen untergeordnete Objekte (z.B. `Selection.Font`, `Selection.Paragraph` und `Selection.Border`). Im neuen API stehen die Formatierungsattribute an allen jeweils zuständigen Objekten (`Paragraph`, `TextCursor` usw.) zur Verfügung und können darauf direkt angewendet werden. Eine Auflistung der verfügbaren Zeichen- und Absatzattribute finden Sie in den folgenden Absätzen.

Zeichenattribute

Als Zeichenattribute werden all jene Formatattribute bezeichnet, die sich auf einzelne Zeichen beziehen. Dazu zählen z.B. der Fettdruck und die Schriftart. Objekte, die das Setzen von Zeichenattributen gestatten, müssen den Service `com.sun.star.style.CharacterProperties` unterstützen. StarOffice kennt eine ganze Reihe von Services, die diesen Service unterstützen. Dazu zählen die vorstehenden Services `com.sun.star.text.Paragraph` für Absätze sowie `com.sun.star.text.TextPortion` für Absatzteile.

Der Service `com.sun.star.style.CharacterProperties` stellt keinerlei Schnittstellen zur Verfügung, bietet dafür allerdings eine Reihe von Eigenschaften an, über die sich die Zeichenattribute festlegen und auslesen lassen. Eine komplette Liste sämtlicher Zeichenattribute findet sich in der Referenz des StarOffice API. Hier ein Auszug der wichtigsten Attribute:

- **`CharFontName (String)`** – Name der ausgewählten Schriftart .
- **`CharColor (Long)`** – Farbwert des Textes.
- **`CharHeight (Float)`** – Höhe der Zeichen in Punkt (pt).
- **`CharUnderline (Constant group)`** – Art des Unterstriches (Konstanten gemäß `com.sun.star.awt.FontUnderline`).
- **`CharWeight (Constant group)`** – Schriftstärke (Konstanten gemäß `com.sun.star.awt.FontWeight`).
- **`CharBackColor (Long)`** – Farbwert des Schrifthintergrundes.

- **CharKeepTogether (Boolean) –** Unterdrückung des automatischen Zeilenumbruches.
- **CharStyleName (String) –** Name der Zeichenvorlage.

Absatzattribute

Als Absatzattribute gelten jene Formatierungsinformationen, die sich nicht auf einzelne Zeichen, sondern auf den gesamten Absatz beziehen, wie der Abstand des Absatzes zum Seitenrand sowie der Zeilenabstand. Verfügbar sind die Absatzattribute über den Service `com.sun.star.style.ParagraphProperties`.

Auch die Absatzattribute stehen in verschiedenen Objekten zur Verfügung. Sämtliche Objekte, die den Service `com.sun.star.text.Paragraph` unterstützen, bieten auch eine Unterstützung für die Absatzattribute in `com.sun.star.style.ParagraphProperties` an.

Eine vollständige Auflistung der Absatzattribute findet sich in der Referenz des StarOffice API. Die gebräuchlichsten Absatzattribute lauten:

- **ParaAdjust (enum) –** vertikale Textausrichtung (Konstanten gemäß `com.sun.star.style.ParagraphAdjust`).
- **ParaLineSpacing (struct) –** Zeilenabstand (Struktur gemäß `com.sun.star.style.LineSpacing`).
- **ParaBackColor (Long) –** Hintergrundfarbe.
- **ParaLeftMargin (Long) –** linker Abstand in 100stel Millimeter.
- **ParaRightMargin (Long) –** rechter Abstand in 100stel Millimeter.
- **ParaTopMargin (Long) –** oberer Abstand in 100stel Millimeter.
- **ParaBottomMargin (Long) –** unterer Abstand in 100stel Millimeter.
- **ParaTabStops (Array of struct) –** Art und Position der Tabulatoren (Array mit Strukturen des Typs `com.sun.star.style.TabStop`).
- **ParaStyleName (String) –** Name der Absatzvorlage.

Beispiel: Einfacher HTML-Export

Das folgende Beispiel demonstriert den Umgang mit Formatierungsinformationen. Es iteriert durch ein Textdokument und erzeugt eine einfache HTML-Datei. Jeder Absatz wird dazu in ein eigenes HTML-Element `<P>` gefasst. Fett dargestellte Absatzteile werden beim Export des weiteren über ein HTML-Element `<B>` gekennzeichnet.

```
Dim FileNo As Integer, Filename As String, CurLine As String

Dim Doc As Object
Dim Enum1 As Object, Enum2 As Object
Dim TextElement As Object, TextPortion As Object

Filename = "c:\text.html"
FileNo = Freefile
Open Filename For Output As #FileNo
Print #FileNo, "<HTML><BODY>"
```

```
Doc = StarDesktop.CurrentComponent
Enum1 = Doc.Text.createEnumeration

' Schleife über alle Absätze
While Enum1.hasMoreElements
      TextElement = Enum1.nextElement
      If TextElement.supportsService("com.sun.star.text.Paragraph") Then
            Enum2 = TextElement.createEnumeration
            CurLine = "<P>"

            ' Schleife über alle Absatz-Teile
            While Enum2.hasMoreElements
                  TextPortion = Enum2.nextElement
                  If TextPortion.CharWeight = com.sun.star.awt.FontWeight.BOLD THEN
                        CurLine = CurLine & "<B>" & TextPortion.String & "</B>"
                  Else
                        CurLine = CurLine & TextPortion.String
                  End If
            Wend

            ' Ausgabe der Zeile
            CurLine = CurLine & "</P>"
            Print #FileNo, CurLine

      End If
Wend

' HTML-Fuß schreiben
Print #FileNo, "</BODY></HTML>"
Close #FileNo
```

Die Basisstruktur des Beispieles orientiert sich an den weiter oben bereits aufgeführten Beispielen zum Durchlaufen der Absatzteile eines Textes. Hinzu gekommen sind die Funktionen zum Schreiben der HTML-Datei sowie ein Prüfcode, der die Schriftstärke der jeweiligen Textabschnitte prüft und fett dargestellte Absatzteile mit einem entsprechenden HTML-Tag versieht.

Vorgabewerte für Zeichen und Absatzattribute

Direkte Formatierungen haben stets Vorrang gegenüber *indirekten* Formatierungen, oder anders ausgedrückt: Formatierungen über Vorlagen haben eine niedrigere Priorität als direkte Formatierungen in einem Text.

Ob ein Teil eines Dokumentes direkt oder indirekt formatiert wurde, ist zumindest für einen Endanwender gar nicht so leicht feststellbar. Die Symbolleiste von StarOffice verrät zwar die gängigen Textattribute wie Schriftart, -stärke und -größe auf den ersten Blick. Unklar bleibt dabei allerdings, ob die betreffenden Einstellungen auf eine Vorlage oder eine direkte Formatierung im Text zurück gehen.

StarOffice Basic stellt eine Schnittstelle zur Verfügung, mit der geprüft werden kann, wie ein spezielles Formatierungsattribut zustande kommt. Hierzu steht die Methode `getPropertyState` zur Verfügung. Sie erwartet als Parameter den Namen des zu überprüfenden Attributs und liefert eine Konstante zurück, die Auskunft über die Ursache der Formatierung gibt. Möglich sind folgende Antworten, die in der Aufzählung `com.sun.star.beans.PropertyState` definiert sind:

- **`com.sun.star.beans.PropertyState.DIRECT_VALUE`–** das Attribut wird direkt im Text definiert (direkte Formatierung),

- **com.sun.star.beans.PropertyState.DEFAULT_VALUE–** das Attribut wird über eine Vorlage definiert (indirekte Formatierung)
- **com.sun.star.beans.PropertyState.AMBIGUOUS_VALUE–** das Attribut ist unklar. Dieser Zustand tritt beispielsweise beim Abfragen der Eigenschaft Fettdruck eines Absatzes auf, der sowohl fett dargestellte Worte als auch normal dargestellte Worte umfasst.

Das nachfolgende Beispiel zeigt auf, wie sich Formatattribute in StarOffice bearbeiten lassen. Es durchsucht einen Text nach Absatzteilen, die über eine direkte Formatierung als Fettdruck dargestellt wurden. Stößt es auf einen entsprechenden Absatzteil, so löscht es die direkte Formatierung über die Methode `setPropertyToDefault` und weist dem betreffenden Absatzteil eine Zeichenvorlage `MyBold` zu.

```
Dim Doc As Object
Dim Enum1 As Object
Dim Enum2 As Object
Dim TextElement As Object
Dim TextPortion As Object

Doc = StarDesktop.CurrentComponent
Enum1 = Doc.Text.createEnumeration

' Schleife über alle Absätze
While Enum1.hasMoreElements
      TextElement = Enum1.nextElement
      If TextElement.supportsService("com.sun.star.text.Paragraph") Then
            Enum2 = TextElement.createEnumeration

            ' Schleife über alle Absatz-Teile
            While Enum2.hasMoreElements
                  TextPortion = Enum2.nextElement
                  If TextPortion.CharWeight = _
                     com.sun.star.awt.FontWeight.BOLD AND _
                     TextPortion.getPropertyState("CharWeight") = _
                     com.sun.star.beans.PropertyState.DIRECT_VALUE Then

                        TextPortion.setPropertyToDefault("CharWeight")
                        TextPortion.CharStyleName = "MyBold"

                  End If
            Wend

      End If
Wend
```

Textdokumente effizient bearbeiten

Im vorigen Abschnitt wurden bereits eine ganze Reihe von Möglichkeiten zur Bearbeitung von Textdokumenten aufgezeigt. Im Zentrum standen dabei die Services `com.sun.star.text.TextPortion` und `com.sun.star.text.Paragraph`, die einen Zugriff auf Absatzteile sowie Absätze ermöglichen. Sie eignen sich für Anwendungen, in denen der Inhalt eines Textes von vorne bis hinten in einer Schleife bearbeitet werden soll. Für viele Probleme reicht dies jedoch nicht aus. So gibt es Aufgabenstellungen, in denen rückwärts innerhalb eines Textdokumentes navigiert werden muss oder in denen sich die Navigationssprünge besser an Sätzen beziehungsweise Wörtern statt an abstrakten `TextPortions` orientieren sollten. Für all diese Aufgaben bietet StarOffice den Service `com.sun.star.text.TextCursor` an.

Der TextCursor

Ein `TextCursor` in der StarOffice API ist vergleichbar mit dem sichtbaren Cursor in einem StarOffice-Dokument. Er markiert eine bestimmte Stelle innerhalb eines Textdokumentes und lässt sich per Befehl in verschiedene Richtungen navigieren. Dennoch sollte man die in StarOffice Basic verfügbaren `TextCursor`-Objekte nicht mit dem sichtbaren Cursor verwechseln. Es handelt sich um zwei verschiedene Dinge.

> Achtung! Unterschiedliche Terminologie zu VBA: Das `Range`-Objekt von VBA ist vom Funktionsumfang her mit dem `TextCursor`-Objekt von StarOffice vergleichbar und nicht – wie der Name eventuell vermuten lässt – mit dem `Range`-Objekt von StarOffice.
>
> Das `TextCursor`-Objekt von StarOffice liefert beispielsweise Methoden zum Navigieren und Ändern von Text, die in VBA im `Range`-Objekt untergebracht sind (z.B. `MoveStart`, `MoveEnd`, `InsertBefore`, `InsertAfter`). Die passenden Gegenstücke des `TextCursor`-Objektes von StarOffice finden Sie in den folgenden Abschnitten beschrieben.

Navigieren innerhalb eines Textes

Die `TextCursor`-Objekte in StarOffice Basic agieren unabhängig von dem sichtbaren Cursor in einem Textdokument. Eine programmgesteuerte Positionsänderung eines `TextCursor`-Objektes hat keinerlei Auswirkungen auf den sichtbaren Cursor. Es ist sogar möglich, für ein und dasselbe Dokument mehrere `TextCursor`-Objekte zu öffnen, die unabhängig voneinander in verschiedenen Positionen eingesetzt werden.

Ein `TextCursor`-Objekt wird erstellt über den Aufruf `createTextCursor`:

```
Dim Doc As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent
Cursor = TextDocument.Text.createTextCursor()
```

Das so erzeugte `Cursor`-Objekt unterstützt den Service `com.sun.star.text.TextCursor`, der wiederum eine ganze Reihe von Methoden für das Navigieren innerhalb von Textdokumenten zur Verfügung stellt. Das nachfolgende Beispiel bewegt den `TextCursor` zunächst 10 Zeichen nach links und dann drei Zeichen nach rechts:

```
Cursor.goLeft(10, False)
Cursor.goRight(3, False)
```

Ein `TextCursor` kann einen kompletten, zusammenhängenden Bereich markieren. Eine solche Markierung ist vergleichbar mit der Markierung einer Textstelle durch die Maus. Der Parameter `False` in den oben stehenden Funktionsaufrufen gibt an, ob der mit der Cursor-Bewegung überquerte Bereich markiert werden soll oder nicht. So wandert der `TextCursor` in folgendem Beispiel

```
Cursor.goLeft(10, False)
Cursor.goRight(3, True)
```

zunächst zehn Zeichen nach rechts ohne diese zu markieren und schreitet danach drei Zeichen zurück, wobei er diese markiert. Der vom `TextCursor` markierte Bereich beginnt somit nach dem siebten Zeichen im Text und endet nach dem zehnten Zeichen.

Hier die zentralen Methoden, die der Service `com.sun.star.text.TextCursor` zur Navigation zur Verfügung stellt:

- **`goLeft (Count, Expand)`** – springt `Count` Zeichen nach links.
- **`goRight (Count, Expand)`** – springt `Count` Zeichen nach rechts.
- **`gotoStart (Expand)`** – springt zum Anfang des Textdokumentes.
- **`gotoEnd (Expand)`** – springt zum Ende des Textdokumentes.
- **`gotoRange (TextRange, Expand)`** – springt zu dem angegebenen `TextRange`-Objekt.
- **`gotoStartOfWord (Expand)`** – springt zum Anfang der aktuellen Wortes.
- **`gotoEndOfWord (Expand)`** – springt zum Ende des aktuellen Wortes.
- **`gotoNextWord (Expand)`** – springt zum Anfang des nächsten Wortes.
- **`gotoPreviousWord (Expand)`** – springt zum Anfang des vorstehenden Wortes.
- **`isStartOfWord ()`** - liefert `True`, falls der `TextCursor` am Anfang eines Wortes steht.
- **`isEndOfWord ()`** - liefert `True`, falls der `TextCursor` am Ende eines Wortes steht.
- **`gotoStartOfSentence (Expand)`** – springt zum Anfang des aktuellen Satzes.
- **`gotoEndOfSentence (Expand)`** – springt zum Ende des aktuellen Satzes.
- **`gotoNextSentence (Expand)`** – springt zum Anfang des nächsten Satzes.
- **`gotoPreviousSentence (Expand)`** – springt zum Anfang des vorstehenden Satzes.
- **`isStartOfSentence ()`** - liefert `True`, falls der `TextCursor` am Anfang eines Satzes steht.
- **`isEndOfSentence ()`** - liefert `True`, falls der `TextCursor` am Ende eines Satzes steht.
- **`gotoStartOfParagraph (Expand)`** – springt zum Anfang des aktuellen Absatzes.
- **`gotoEndOfParagraph (Expand)`** – springt zum Ende des aktuellen Absatzes.
- **`gotoNextParagraph (Expand)`** – springt zum Anfang des nächsten Absatzes.
- **`gotoPreviousParagraph (Expand)`** – springt zum Anfang des vorstehenden Absatzes.

- **isStartOfParagraph ()** – liefert `True`, falls der `TextCursor` am Anfang eines Absatzes steht.
- **isEndOfParagraph ()** – liefert `True`, falls der `TextCursor` am Ende eines Absatzes steht.

Die Aufteilung in Sätze erfolgt auf der Grundlage von Satzzeichen. Dadurch werden zum Beispiel auch Abkürzungspunkte als Satzendezeichen interpretiert.

Bei dem Parameter `Expand` handelt es sich um einen Booleschen Wert, der angibt, ob der beim Navigieren überstrichene Bereich markiert werden soll oder nicht. Sämtliche Navigations-Methoden geben darüber hinaus übrigens einen Parameter aus, der angibt, ob die Navigation erfolgreich war, oder ob die Aktion mangels Text abgebrochen wurde.

Bleiben noch einige Methoden zur Bearbeitung der von einem `TextCursor` markierten Bereiche zu erwähnen, die der Service `com.sun.star.text.TextCursor` ebenfalls unterstützt:

- **collapseToStart ()** – setzt die Markierung zurück und positioniert den `TextCursor` am Beginn des bisher markierten Bereiches.
- **collapseToEnd ()** – setzt die Markierung zurück und positioniert den `TextCursor` am Ende des bisher markierten Bereiches.
- **isCollapsed ()** – gibt `True` aus, falls der `TextCursor` aktuell keine Markierung umfasst.

Text via TextCursor formatieren

Der Service `com.sun.star.text.TextCursor` unterstützt sämtliche Zeichen- und Absatzattribute, die weiter vorne in diesem Kapitel bereits vorgestellt wurden.

Das nachfolgende Beispiel verdeutlicht, wie sich diese zusammen mit einem `TextCursor` verwenden lassen. Es durchwandert ein komplettes Dokument und formatiert das erste Wort eines jeden Satzes in Fettdruck.

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor

Do

      Cursor.gotoEndOfWord(True)
      Cursor.CharWeight = com.sun.star.awt.FontWeight.BOLD
      Proceed = Cursor.gotoNextSentence(False)
      Cursor.gotoNextWord(False)

Loop While Proceed
```

Das Beispiel erzeugt zunächst ein Dokument-Objekt für den gerade geöffneten Text. Danach iteriert es Satz für Satz durch den gesamten Text, markiert jeweils das erste Wort und formatiert dieses fett.

Textinhalte auslesen und ändern

Enthält ein `TextCursor` eine Markierung, so ist dieser Text über die Eigenschaft `String` des `TextCursor`-Objektes verfügbar. Das folgende Beispiel verwendet die Eigenschaft `String`, um jeweils das erste Wort eines Satzes in einem Meldungsfenster auszugeben:

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor

Do

      Cursor.gotoEndOfWord(True)
      MsgBox Cursor.String
      Proceed = Cursor.gotoNextSentence(False)
      Cursor.gotoNextWord(False)

Loop While Proceed
```

Analog ist es möglich, das erste Wort eines jeden Satzes über die Eigenschaft `String` zu ändern:

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor

Do

      Cursor.gotoEndOfWord(True)
      Cursor.String = "Ups"
      Proceed = Cursor.gotoNextSentence(False)
      Cursor.gotoNextWord(False)

Loop While Proceed
```

Enthält der `TextCursor` eine Markierung, ersetzt eine Zuweisung an die Eigenschaft `String` diese durch den neuen Text. Fehlt die Markierung, so wird der Text an der aktuellen `TextCursor`-Position eingefügt.

Steuerzeichen einfügen

In einigen Situationen ist es notwendig, nicht den eigentlichen Text eines Dokumentes zu modifizieren, sondern seine Struktur. Hierfür bietet StarOffice Steuerzeichen an, die in den Text eingefügt werden und Einfluss auf dessen Aufbau nehmen. Die Steuerzeichen sind in der Konstantengruppe `com.sun.star.text.ControlCharacter` definiert. Folgende Steuerzeichen sind in StarOffice verfügbar:

- **`PARAGRAPH_BREAK`** – Absatzumbruch
- **`LINE_BREAK`** – Zeilenumbruch innerhalb eines Absatzes.
- **`SOFT_HYPHEN`** – mögliche Stelle für eine Silbentrennung.
- **`HARD_HYPHEN`** – obligatorische Stelle für eine Silbentrennung.
- **`HARD_SPACE`** – Geschütztes Leerzeichen, dass auch im Blocksatz nicht gespreizt oder gepresst wird.

Zum Einfügen der Steuerzeichen bedarf es neben dem Cursor auch des zugehörigen Textdokument-Objektes. Das folgende Beispiel fügt einen Absatz nach dem 20. Zeichen eines Textes ein:

```
Dim Doc As Object
Dim Cursor As Object
Dim Proceed As Boolean

Doc = StarDesktop.CurrentComponent

Cursor = Doc.Text.createTextCursor
Cursor.goRight(20, False)

Doc.Text.insertControlCharacter(Cursor, _
      com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, False)
```

Der Parameter `False` im Aufruf der Methode `insertControlCharacter` sorgt dafür, dass der aktuell vom `TextCursor` markierten Bereich durch die Einfügeoperation bestehen bleibt. Wird hier der Parameter `True` übergeben, so ersetzt `insertControlCharacter` den aktuellen Text.

Suchen von Textteilen

In vielen Fällen gilt es, einen Text auf einen bestimmten Begriff hin zu durchsuchen und die betreffende Stelle zu bearbeiten. Hierzu bieten alle StarOffice-Dokumente eine spezielle Schnittstelle an, die stets nach dem gleichen Prinzip arbeitet: Vor einem Suchvorgang muss zunächst ein so genannter `SearchDescriptor` erstellt werden, der festlegt, nach was StarOffice ein Dokument durchsuchen soll. Ein `SearchDescriptor` ist ein Objekt, das den Service `com.sun.star.util.SearchDescriptor` unterstützt und sich über die Methode `createSearchDescriptor` eines Dokumentes erzeugen lässt:

```
Dim SearchDesc As Object
SearchDesc = Doc.createSearchDescriptor
```

Nachdem der `SearchDescriptor` erzeugt ist, nimmt er den zu suchende Text entgegen:

```
SearchDesc.searchString="any text"
```

Von seiner Funktion her ist der `SearchDescriptor` am ehesten mit dem Such-Dialog von StarOffice zu vergleichen. Ähnlich wie im Suchfenster kann man auch im `SearchDescriptor`-Objekt die notwendigen Einstellungen für eine Suche treffen.

Folgende Eigenschaften werden von dem Service `com.sun.star.util.SearchDescriptor` zur Verfügung gestellt:

- **`SearchBackwards (Boolean)`** - durchsucht den Text rückwärts statt vorwärts.
- **`SearchCaseSensitive (Boolean)`** - beachtet bei der Suche Groß- und Kleinschreibung.
- **`SearchRegularExpression (Boolean)`** - behandelt den Suchausdruck als regulären Ausdruck.
- **`SearchStyles (Boolean)`** - sucht im Text nach der beschriebenen Absatzvorlage.
- **`SearchWords (Boolean)`** - sucht nur komplette Worte.

Auch die Ähnlichkeitssuche von StarOffice steht in StarOffice Basic zur Verfügung. Hierbei sucht StarOffice nach einem Begriff, der zwar ähnlich, aber nicht gleich dem Suchbegriff ist. Die Anzahl der hinzu gekommenen, gelöschten und geänderten Zeichen lässt sich dabei individuell definieren. Hier die zugehörigen Eigenschaften des Service `com.sun.star.util.SearchDescriptor`:

- **`SearchSimilarity (Boolean)`** - führt eine Ähnlichkeitssuche durch.
- **`SearchSimilarityAdd (Short)`** - Anzahl der Zeichen, die für eine Ähnlichkeitssuche hinzugefügt werden dürfen.
- **`SearchSimilarityExchange (Short)`** - Anzahl der Zeichen, die im Rahmen einer Ähnlichkeitssuche ausgetauscht werden dürfen.
- **`SearchSimilarityRemove (Short)`** - Anzahl der Zeichen, die im Rahmen einer Ähnlichkeitssuche entfernt werden dürfen.
- **`SearchSimilarityRelax (Boolean)`** - beachtet alle Abweichungsregeln für den Suchausdruck gleichzeitig.

Ist der `SearchDescriptor` wie gewünscht vorbereitet, kann er auf das Textdokument angewendet werden. Hierzu stellen die StarOffice-Dokumente die Methoden `findFirst` sowie `findNext` zur Verfügung:

```
Found = Doc.findFirst (SearchDesc)

Do While Found
      ' Suchergebnis bearbeiten
      Found = Doc.findNext( Found.End, Search)
Loop
```

Das Beispiel ermittelt in einer Schleife sämtliche Treffer und liefert ein `TextRange`-Objekt zurück, das auf die gefundene Textstelle verweist.

Beispiel: Ähnlichkeitssuche

Dieses Beispiel zeigt, wie man einen Text nach dem Wort "Umsatz" durchsucht und die gefundenen Stellen fett formatieren lässt. Um neben dem Wort Umsatz auch die Pluralform "Umsätze" und Deklinationen wie "Umsatzes" zu finden, kommt eine Ähnlichkeitssuche zum Einsatz. Die gefundenen Begriffe dürfen gegenüber dem Suchbegriff bis zu zwei Buchstaben mehr oder weniger enthalten und zusätzlich um zwei Buchstaben vom Original abweichen:

```
Dim SearchDesc As Object
Dim Doc As Object

Doc = StarDesktop.CurrentComponent

SearchDesc = Doc.createSearchDescriptor
SearchDesc.SearchString="Umsatz"
SearchDesc.SearchSimilarity = True
SearchDesc.SearchSimilarityAdd = 2
SearchDesc.SearchSimilarityExchange = 2
SearchDesc.SearchSimilarityRemove = 2
SearchDesc.SearchSimilarityRelax = False

Found = Doc.findFirst (SearchDesc)

Do While Found
      Found.CharWeight = com.sun.star.awt.FontWeight.BOLD
      Found = Doc.findNext( Found.End, Search)
Loop
```

Die Grundidee beim Suchen und Ersetzen in StarOffice ist vergleichbar mit der in VBA. Beide Schnittstellen bieten dem Programmierer ein Objekt an, über das er die Eigenschaften für das Suchen und Ersetzen definieren kann. Dieses Objekt wird anschließend auf den gewünschten Textbereich angewendet um die Aktion auszuführen. Während das zuständige Hilfsobjekt in VBA über die `Find`-Eigenschaft des `Range`-Objektes erreichbar ist, wird es in StarOffice Basic jedoch über den Aufruf `createSearchDescriptor` beziehungsweise `createReplaceDescriptor` des Dokument-Objektes erzeugt. Auch die verfügbaren Suchattribute und Methoden unterscheiden sich.

Wie im alten API von StarOffice erfolgt auch im neuen API das Suchen und Ersetzen von Text über das Dokument-Objekt. Während es speziell zur Definition der Suchoptionen bisher ein Objekt namens `SearchSettings` gab, erfolgt die Suche im neuen Objekt nun über ein Objekt `SearchDescriptor` beziehungsweise `ReplaceDescriptor` für das automatische Ersetzen von Text. Diese Objekte umfassen nicht nur die Optionen, sondern auch den eigentlichen Suchtext und gegebenenfalls die zugehörige Textersetzung. Die Deskriptor-Objekte werden über das Dokumentobjekt erzeugt, entsprechend den jeweiligen Wünschen ausgefüllt und im Anschluss daran dem Dokumentobjekt als Parameter der Suchmethoden zurück gegeben.

Ersetzen von Textteilen

Analog zur Suchfunktion steht auch die Ersetzungsfunktion von StarOffice in StarOffice Basic zur Verfügung. Die Handhabung der beiden Funktionen ist identisch. Auch für einen Ersetzungsvorgang bedarf es zunächst eines speziellen Objektes, dass die Rahmenbedingungen für den Ersetzungsvorgang aufnimmt. Es wird als `ReplaceDescriptor` bezeichnet und unterstützt den Service `com.sun.star.util.ReplaceDescriptor`. Sämtliche der im vorstehenden Absatz beschriebenen Eigenschaften des `SearchDescriptor` werden auch von `ReplaceDescriptor` unterstützt. So ist es beispielsweise möglich, bei einem Ersetzungsvorgang die Beachtung von Groß- und Kleinschreibung ein- und auszuschalten und Ähnlichkeitssuchen durchzuführen.

Das nachfolgende Beispiel demonstriert den Einsatz eines `ReplaceDescriptors` für eine Suche innerhalb eines StarOffice-Dokumentes.

```
Dim I As Long
Dim Doc As Object
Dim Replace As Object
Dim BritishWords(5) As String
Dim USWords(5) As String

BritishWords() = Array("colour", "neighbour", "centre", "behaviour", _
      "metre", "through")

USWords() = Array("color", "neighbor", "center", "behavior", _
      "meter", "thru")

Doc = StarDesktop.CurrentComponent
Replace = Doc.createReplaceDescriptor

For O = 0 To 5
      Replace.SearchString = BritishWords(I)
      Replace.ReplaceString = USWords(I)
      Doc.replaceAll(Replace)
Next n
```

Die Begriffe zum Suchen und Ersetzen werden über die Eigenschaften `SearchString` beziehungsweise `ReplaceString` des `ReplaceDescriptors` gesetzt. Schließlich folgt der eigentliche Ersetzungsvorgang über die Methode `replaceAll` des Dokument-Objektes, die sämtliche Vorkommen des Suchbegriffs ersetzt.

Beispiel: Suchen und Ersetzen von Text mit regulären Ausdrücken

Besonders leistungsfähig ist die Ersetzungsfunktion von StarOffice im Zusammenhang mit regulären Ausdrücken. Diese bieten eine Möglichkeit, anstelle eines festen Wertes einen variablen Suchausdruck mit Platzhaltern und Sonderzeichen zu definieren.

Die von StarOffice unterstützten regulären Ausdrücke sind in der Online-Hilfe von StarOffice ausführlich beschrieben. Einige Beispiele seien hier exemplarisch genannt:

- Ein Punkt innerhalb eines Suchausdruckes steht für ein beliebiges Zeichen. So steht der Suchausdruck `sh.rt` sowohl für `shirt` als auch für `short`.
- Das Zeichen `^` markiert den Beginn eines Absatzes. Über den Suchausdruck `^Peter` lassen sich so alle Vorkommen des Namens `Peter` finden, die am Anfang eines Absatzes stehen.

- Das Zeichen $ markiert ein Absatzende. Über den Suchausdruck `Peter$` lassen sich so alle Vorkommen des Namens `Peter` finden, die am Ende eines Absatzes stehen.
- Ein * bestimmt, dass das vorstehende Zeichen beliebig oft wiederholt vorkommen darf. Er ist mit dem Punkt als Platzhalter für ein beliebiges Zeichen kombinierbar. Der Ausdruck `temper.*e` steht beispielsweise für die englischen Begriffe `temperance` und `temperature`.

Das folgende Beispiel zeigt, wie sich mit Hilfe des regulären Ausdrucks `^$` sämtliche Leerzeilen aus einem Textdokument entfernen lassen:

```
Dim Doc As Object
Dim Replace As Object
Dim I As Long

Doc = StarDesktop.CurrentComponent
Replace = Doc.createReplaceDescriptor

Replace.SearchRegularExpression = True
Replace.SearchString = "^$"
Replace.ReplaceString = ""

Doc.replaceAll(Replace)
```

Textdokumente: Mehr als nur Text

Bisher ging es in diesem Kapitel ausschließlich um Textabsätze und deren Teile. Aber Textdokumente können auch andere Objekte enthalten. Dazu zählen Tabellen, Zeichnungen, Textfelder und Verzeichnisse. Alle diese Objekte können an jeder beliebigen Position innerhalb eines Textes verankert werden.

Aufgrund dieser Gemeinsamkeiten unterstützen alle diese Objekte in StarOffice einen gemeinsamen Basis-Service namens `com.sun.star.text.TextContent`. Er stellt folgende Eigenschaften zur Verfügung:

- **`AnchorType (Enum)`** – bestimmt den Verankerungstyp eines `TextContent`-Objekts (Vorgabewerte gemäß Aufzählung `com.sun.star.text.TextContentAnchorType`)
- **`AnchorTypes (sequence of Enum)`** – Aufzählung aller `AnchorTypes`, die ein spezielles `TextContent`-Objekt unterstützt.
- **`TextWrap (Enum)`** – bestimmt den Umlauftyp um ein `TextContent`-Objekt (Vorgabewerte gemäß Aufzählung `com.sun.star.text.WrapTextMode`).

Auch einige Methoden teilen sich die `TextContent`-Objekte – insbesondere die zum Erzeugen, Einfügen und Löschen von Objekten.

- Das **Erzeugen** eines neuen `TextContent`-Objektes erfolgt über die Methode `createInstance` des Dokument-Objektes.
- Das **Einfügen** eines Objektes erfolgt über die Methode `insertTextContent` des Text-Objektes.
- Das **Löschen** von `TextContent`-Objekten erfolgt über die Methode `removeTextContent`.

In den nachfolgenden Abschnitten finden sich eine Reihe von Beispielen, die diese Methoden verwenden.

Tabellen

Das nachfolgende Beispiel erzeugt eine Tabelle unter Zuhilfenahme der weiter oben beschriebenen Methode `createInstance`.

```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

Table = Doc.createInstance("com.sun.star.text.TextTable")
Table.initialize(5, 4)

Doc.Text.insertTextContent(Cursor, Table, False)
```

Nach der Erzeugung wird die Tabelle über einen Aufruf `initialize` auf die gewünschte Anzahl der Zeilen und Spalten eingestellt und dann über `insertTextContent` in das Textdokument eingefügt.

Wie im Beispiel zu sehen, erwartet die Methode `insertTextContent` neben dem einzufügenden `Content`-Objekt noch zwei weitere Parameter:

- ein `Cursor`-Objekt, dass die Einfügeposition bestimmt
- eine Boolesche Variable, die angibt, ob das `Content`-Objekt die aktuelle Selektion des Cursors ersetzen (Wert `True`) oder vor dieser in den Text eingefügt werden soll (`False`)

Beim Erzeugen und Einfügen von Tabellen in ein Textdokument kommen in StarOffice Basic ähnliche Objekte zum Einsatz wie in VBA: Das Dokumentobjekt und ein `TextCursor`-Objekt in StarOffice Basic bzw. das `Range`-Objekt als VBA-Gegenstück. Während das Erstellen und Einfügen der Tabelle in VBA die Methode `Document.Tables.Add` übernimmt, wird die Tabelle in StarOffice Basic gemäß dem vorstehenden Beispiel über `createInstance` erzeugt, initialisiert und via `insertTextContent` in das Dokument eingefügt.

Die in ein Textdokument eingefügten Tabellen können über eine einfache Schleife ermittelt werden. Dazu dient die Methode `getTextTables()` des Textdokument-Objektes:

```
Dim Doc As Object
Dim TextTables As Object
Dim Table As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
TextTables = Doc.getTextTables()

For I = 0 to TextTables.count - 1
      Table = TextTables(I)

      ' Bearbeitung von Table
Next I
```

Texttabellen sind in StarOffice 7 über die Liste `TextTables` des Dokument-Objektes verfügbar. Sie löst damit die bisherige Tabellen-Auflistung im `Selection`-Objekt ab. Das vorstehende Beispiel zeigt, wie sich eine Texttabelle ermitteln lässt. Im nachfolgenden Abschnitt werden die Zugriffsmöglichkeiten auf

Texttabellen beschrieben.

Tabellen bearbeiten

Eine Tabelle setzt sich zusammen aus einzelnen Zeilen. Diese wiederum enthalten die verschiedenen Zellen. Streng genommen gibt es in StarOffice keine Tabellenspalten. Diese ergeben sich implizit durch die gleiche Anordnung der untereinander stehenden Zeilen. Um den Zugriff auf die Tabellen zu vereinfachen, bietet StarOffice dennoch einige Methoden an, die spaltenweise arbeiten. Sie sind nutzbar, sofern dieTabelle ohne Zellzusammenfassungen auskommt.

Wenden wir uns zunächst den Eigenschaften der Tabelle selbst zu. Sie sind in dem Service `com.sun.star.text.TextTable` definiert. Hier ein Auszug der wichtigsten Eigenschaften des Tabellen-Objektes:

- **`BackColor (Long)`** – Hintergrundfarbe der Tabelle.
- **`BottomMargin (Long)`** – unterer Abstand in 100stel Millimeter.
- **`LeftMargin (Long)`** – linker Abstand in 100stel Millimeter.
- **`RightMargin (Long)`** – rechter Abstand in 100stel Millimeter.
- **`TopMargin (Long)`** – oberer Abstand in 100stel Millimeter.
- **`RepeatHeadline (Boolean)`** – Tabellenüberschrift wird auf jeder Seite wiederholt.
- **`Width (Long)`** – absolute Breite der Tabelle in 100stel Millimeter.

Zeilen

Eine Tabelle setzt sich aus einer Liste mit Zeilen zusammen. Das folgende Beispiel zeigt, wie die Zeilen einer Tabelle ausgelesen und formatiert werden können.

```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object
Dim Rows As Object
Dim Row As Object
Dim I As Integer
Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

Table = Doc.createInstance("com.sun.star.text.TextTable")
Table.initialize(5, 4)

Doc.Text.insertTextContent(Cursor, Table, False)
Rows = Table.getRows
For I = 0 To Rows.getCount() - 1
      Row = Rows.getByIndex(I)
      Row.BackColor = &HFF00FF
Next
```

Das Beispiel erstellt zunächst über einen Aufruf `Table.getRows` eine Liste mit allen Zeilen. Die Methoden `getCount` und `getByIndex` gestatten eine Weiterverarbeitung der Liste und gehören zu der Schnittstelle `com.sun.star.table.XTableRows`. Die Methode `getByIndex` liefert ein Zeilenobjekt zurück, das den Service `com.sun.star.text.TextTableRow` unterstützt.

Hier die zentralen Methoden der Schnittstelle `com.sun.star.table.XTableRows`:

- **`getByIndex(Integer)`** – liefert ein Zeilenobjekt für den angegebenen Index.
- **`getCount()`** – liefert die Anzahl der Zeilenobjekte.
- **`insertByIndex(Index, Count)`** – fügt `Count` Zeilen ab der Position `Index` in die Tabelle ein.
- **`removeByIndex(Index, Count)`** – löscht `Count` Zeilen ab der Position `Index` aus der Tabelle.

Während die Methoden `getByIndex` und `getCount` in allen Tabellen zur Verfügung stehen, können die Methoden `insertByIndex` und `removeByIndex` nur in Tabellen genutzt werden, die keine zusammengefassten Zellen enthalten.

Der Service `com.sun.star.text.TextTableRow` bietet unter anderem folgende Eigenschaften an:

- **`BackColor (Long)`** – Hintergrundfarbe der Zeile.
- **`Height (Long)`** – Höhe der Zeile in 100stel Millimeter.
- **`IsAutoHeight (Boolean)`** – Tabellenhöhe passt sich dynamisch dem Inhalt an.
- **`VertOrient (const)`** – vertikale Ausrichtung des Textrahmens - Angaben zur vertikalen Ausrichtung des Textes innerhalb der Tabelle (Werte gemäß `com.sun.star.text.VertOrientation`)

Spalten

Der Zugriff auf Spalten erfolgt analog dem Zeilenzugriff über die Methoden `getByIndex`, `getCount`, `insertByIndex` und `removeByIndex` am `Column`-Objekt, das über `getColumns` erreicht wird . Sie können jedoch nur in Tabellen angewendet werden, die keine zusammengefassten Tabellenzellen enthalten. Eine spaltenweise Formatierung von Zellen über StarOffice Basic ist nicht möglich. Hierbei müssen die Tabellenzellen einzeln formatiert werden.

Zellen

Jede Zelle eines StarOffice-Dokumentes hat einen eindeutigen Namen. Steht der Cursor von StarOffice in einer Zelle, so findet sich der jeweilige Name in der Statusleiste. Die Zelle oben links hat in der Regel den Namen `A1`, die Zeile unten rechts den Name `Xn`, wobei `X` für den Buchstaben der höchsten Spalte und `n` für die Nummer der letzten Zeile steht. Die Zellenobjekte sind über die Methode `getCellByName()` des Tabellenobjektes verfügbar. Das nachfolgende Beispiel zeigt eine Schleife, die nacheinander alle Zellen einer Tabelle durchwandert und die jeweilige Zeilen- und Spaltennummer in die Zellen einträgt.

```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object
Dim Rows As Object
Dim RowIndex As Integer
Dim Cols As Object
Dim ColIndex As Integer
Dim CellName As String
Dim Cell As Object
```

```
Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

Table = Doc.createInstance("com.sun.star.text.TextTable")
Table.initialize(5, 4)

Doc.Text.insertTextContent(Cursor, Table, False)

Rows = Table.getRows
Cols = Table.getColumns

For RowIndex = 1 To Rows.getCount()
      For ColIndex = 1 To Cols.getCount()
            CellName = Chr(64 + ColIndex) & RowIndex
            Cell = Table.getCellByName(CellName)
            Cell.String = "Zeile: " & CStr(RowIndex) + ", Spalte: " & CStr(ColIndex)
      Next
Next
```

Eine Tabellenzelle ist mit einem gewöhnlichen Text vergleichbar. Sie unterstützt die Schnittstelle `createTextCursor` zur Erzeugung eines zugehörigen `TextCursor`-Objektes.

```
CellCursor = Cell.createTextCursor()
```

Damit stehen automatisch sämtliche Formatierungsmöglichkeiten für einzelne Zeichen und Absätze zur Verfügung.

Das folgende Beispiel verdeutlicht die Möglichkeiten, die sich dadurch eröffnen. Es durchsucht sämtliche Tabellen eines Textdokumentes und formatiert alle Zellen mit nummerischen Werten über das entsprechende Absatzattribut rechtsbündig.

```
Dim Doc As Object
Dim TextTables As Object
Dim Table As Object
Dim CellNames
Dim Cell As Object
Dim CellCursor As Object
Dim I As Integer
Dim J As Integer

Doc = StarDesktop.CurrentComponent
TextTables = Doc.getTextTables()

For I = 0 to TextTables.count - 1
      Table = TextTables(I)
      CellNames = Table.getCellNames()

      For J = 0 to UBound(CellNames)
            Cell = Table.getCellByName(CellNames(J))
            If IsNumeric(Cell.String) Then
                  CellCursor = Cell.createTextCursor()
                  CellCursor.paraAdjust = com.sun.star.style.ParagraphAdjust.RIGHT
            End If
      Next
Next
```

Das Beispiel erzeugt eine `TextTables`-Liste mit allen Tabellen eines Textes, die in einer Schleife durchwandert wird. Für jede dieser Tabellen erzeugt StarOffice dann wiederum eine Liste der zugehörigen Zellnamen. Diese werden wiederum in einer Schleife durchwandert. Enthält eine Zelle einen nummerischen Wert, so passt das Beispiel die Formatierung entsprechend an. Hierzu

erzeugt es zunächst ein `TextCursor`-Objekt, das auf den Inhalt der Tabellenzelle verweist und passt daraufhin die Absatzattribute der Tabellenzelle an.

Textrahmen

Textrahmen gelten wie Tabellen und Grafiken als `TextContent`-Objekte. Sie bestehen zwar im wesentlichen aus gewöhnlichem Text, sind jedoch frei auf einer Seite positionierbar und befinden sich außerhalb des Textflusses.

Wie bei allen `TextContent`-Objekten wird auch bei einem Textrahmen zwischen der eigentlichen Erzeugung und dem Einfügen in das Dokument unterschieden.

```
Dim Doc As Object
Dim TextTables As Object
Dim Cursor As Object
Dim Frame As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()
Frame = Doc.createInstance("com.sun.star.text.TextFrame")

Doc.Text.insertTextContent(Cursor, Frame, False)
```

Die Erzeugung erfolgt über die Methode `createInstance` des Dokument-Objektes. Der so erzeugte Textrahmen kann anschließend über die Methode `insertTextContent` des `Text`-Objektes in das Dokument eingefügt werden. Hierbei ist der Name des zuständigen Service `com.sun.star.text.TextFrame` anzugeben.

Die Einfügeposition des Textrahmens wird durch ein `Cursor`-Objekt bestimmt, das beim Einfügen mit aufgeführt wird.

> Textrahmen bilden das StarOffice-Gegenstück zu den Positionsrahmen von Word. Während hierzu in VBA die Methode `Document.Frames.Add` verwendet wird, erfolgt die Erstellung in StarOffice Basic über oben stehendes Verfahren mit Hilfe eines TextCursors sowie der Methode`createInstance` des Dokument-Objektes.

Textrahmen-Objekte stellen eine ganze Reihe von Eigenschaften zur Verfügung, die Position und Verhalten des Rahmens beeinflussen können. Der Großteil dieser Eigenschaften ist in dem Service `com.sun.star.text.BaseFrameProperties` definiert, der von jedem TextFrame-Service unterstützt wird. Die zentralen Eigenschaften lauten:

- **`BackColor (Long)`** – Hintergrundfarbe des Textrahmens.
- **`BottomMargin (Long)`** – unterer Abstand in 100stel Millimeter.
- **`LeftMargin (Long)`** – linker Abstand in 100stel Millimeter.
- **`RightMargin (Long)`** – rechter Abstand in 100stel Millimeter.
- **`TopMargin (Long)`** – oberer Abstand in 100stel Millimeter.
- **`Height (Long)`** – Höhe des Textrahmens in 100stel Millimeter.
- **`Width (Long)`** – Breite des Textrahmens in 100stel Millimeter.

- **`HoriOrient (const)`** – horizontale Ausrichtung des Textrahmens (gemäß `com.sun.star.text.HoriOrientation`).
- **`VertOrient (const)`** – vertikale Ausrichtung des Textrahmens (gemäß `com.sun.star.text.VertOrientation`).

Das nachfolgende Beispiel erstellt einen Textrahmen unter Verwendung der vorstehenden Eigenschaften.

```
Dim Doc As Object
Dim TextTables As Object
Dim Cursor As Object
Dim Frame As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()
Cursor.gotoNextWord(False)
Frame = Doc.createInstance("com.sun.star.text.TextFrame")

Frame.Width = 3000
Frame.Height = 1000
Frame.AnchorType = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
Frame.TopMargin = 0
Frame.BottomMargin = 0
Frame.LeftMargin = 0
Frame.RightMargin = 0
Frame.BorderDistance = 0
Frame.HoriOrient = com.sun.star.text.HoriOrientation.NONE
Frame.VertOrient = com.sun.star.text.VertOrientation.LINE_TOP

Doc.Text.insertTextContent(Cursor, Frame, False)
```

Das Beispiel erstellt einen `TextCursor` als Einfügemarke für den Textrahmen. Dieser wird zwischen dem ersten und dem zweiten Wort des Textes positioniert. Es folgt die Erstellung des Textrahmens über `Doc.createInstance`. Nun werden die Eigenschaften des Textrahmenobjektes auf die gewünschten Ausgangswerte gesetzt.

Zu beachten ist hierbei das Zusammenspiel der Eigenschaften `AnchorType` (aus dem Service `TextContent`) und `VertOrient` (aus dem Service `BaseFrameProperties`). `AnchorType` erhält den Wert `AS_CHARACTER`. Der Textrahmen wird damit unmittelbar in den Textfluss eingefügt und verhält sich wie ein Zeichen. So kann er beispielsweise bei einem Zeilenumbruch in die nächste Zeile geschoben werden. Der Wert `LINE_TOP` der Eingenschaft `VertOrient` stellt sicher, dass die Oberkante des Textrahmens auf gleicher Höhe steht wie die Oberkante der Zeichen.

Nach abgeschlossener Initialisierung wird der Textrahmen schließlich über einen Aufruf von `insertTextContent` in das Textdokument eingefügt.

Zum Bearbeiten des Inhalts eines Textrahmens kommt der bereits mehrfach erwähnte `TextCursor` zum Einsatz, der auch für Textrahmen verfügbar ist.

```
Dim Doc As Object
Dim TextTables As Object
Dim Cursor As Object
Dim Frame As Object
Dim FrameCursor As Object

Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()
Frame = Doc.createInstance("com.sun.star.text.TextFrame")

Frame.Width = 3000
Frame.Height = 1000

Doc.Text.insertTextContent(Cursor, Frame, False)

FrameCursor = Frame.createTextCursor()
FrameCursor.charWeight = com.sun.star.awt.FontWeight.BOLD
FrameCursor.paraAdjust = com.sun.star.style.ParagraphAdjust.CENTER
FrameCursor.String = "Das ist ein kleiner Test!"
```

Das Beispiel erstellt einen Textrahmen, fügt diesen in das aktuelle Dokument ein und öffnet einen `TextCursor` für den Textrahmen. Über diesen Cursor wird die Schrift des Rahmens fett gesetzt und der Absatz mittig ausgerichtet. Schließlich bekommt der Textrahmen den String "Das ist ein kleiner Test!" zugewiesen.

Textfelder

Textfelder zählen zu den `TextContent`-Objekten, da sie über den reinen Text hinaus eine zusätzliche Logik zur Verfügung stellen. Textfelder lassen sich über die gleichen Methoden in ein Textdokument einfügen wie andere `TextContent`-Objekte:

```
Dim Doc As Object
Dim DateTimeField As Object
Dim Cursor As Object
Doc = StarDesktop.CurrentComponent
Cursor = Doc.Text.createTextCursor()

DateTimeField = Doc.createInstance("com.sun.star.text.TextField.DateTime")
DateTimeField.IsFixed = False
DateTimeField.IsDate = True
Doc.Text.insertTextContent(Cursor, DateTimeField, False)
```

Das Beispiel fügt am Anfang des aktuellen Textdokumentes ein Textfeld mit dem aktuellen Datum ein. Die auf `True` gesetzte Eigenschaft `IsDate` führt dazu, dass das Datum und nicht etwa die Uhrzeit angezeigt wird. Der Wert `False` für `IsFixed` sorgt dafür, dass die Datumsangabe beim Öffnen des Dokumentes automatisch aktualisiert wird.

Während der Typ eines Feldes in VBA durch einen Parameter der Methode `Document.Fields.Add` angegeben wird, erfolgt die Festlegung in StarOffice Basic durch den Namen des Services, der für den betreffenden Feldtyp veranwortlich ist.

Der Zugriff auf Textfelder erfolgte bisher über eine ganze Reihe von Methoden, die StarOffice im alten `Selection`-Objekt zur Verfügung gestellt hat (etwa `InsertField`, `DeleteUserField`, `SetCurField`).

Mit StarOffice 7 erfolgt die Verwaltung der Felder nach einem objektorientierten Konzept. Für die Erstellung eines Textfeldes ist zunächst ein Textfeld des gewünschten Typs zu erstellen und über die benötigten Eigenschaften zu initialisieren. Anschließend wird das Textfeld über die Methode `insertTextContent` in das Dokument eingfügt. Ein entsprechender Quelltext findet sich im vorstehenden Beispiel. Die wichtigsten Feldtypen und ihre Eigenschaften sind in den folgenden Abschnitten beschrieben.

Neben dem Einfügen von Textfeldern kann es wichtig sein, ein Dokument nach vorhandenen Feldern zu durchsuchen. Das folgende Beispiel zeigt, wie sämtliche Textfelder eines Textdokumentes in einer Schleife durchlaufen und auf ihren jeweiligen Typ hin überprüft werden können.

```
Dim Doc As Object
Dim TextFieldEnum As Object
Dim TextField As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent

TextFieldEnum = Doc.getTextFields.createEnumeration

While TextFieldEnum.hasMoreElements()

      TextField = TextFieldEnum.nextElement()

      If TextField.supportsService("com.sun.star.text.TextField.DateTime") Then
            MsgBox "Datum/Uhrzeit"
      ElseIf TextField.supportsService("com.sun.star.text.TextField.Annotation") Then
            MsgBox "Notiz"
      Else
            MsgBox "unbekannt"
      End If

Wend
```

Ansatzpunkt für die Ermittlung der vorhandenen Textfelder bildet die Liste `TextFields` des Dokument-Objektes. Auf dieser Grundlage erstellt das Beispiel ein `Enumeration`-Objekt, über das sich wiederum alle Textfelder in einer Schleife abfragen lassen. Die gefundenen Textfelder werden über die Methode `supportsService` auf den unterstützten Service hin überprüft. Handelt es sich um ein Datums/Uhrzeitfeld oder eine Notiz, so wird die betreffende Feldart in einem Hinweisfenster ausgegeben. Stößt das Beispiel hingegen auf ein anderes Feld, so gibt es den Hinweis "unbekannt" aus.

Nachfolgend findet sich eine Aufstellung der wichtigsten Textfelder sowie der zugehörigen Eigenschaften. Eine komplette Liste aller Textfelder liefert die API-Referenz im Modul `com.sun.star.text.TextField`. (Beim Aufführen der Service-Namen eines Textfeldes ist in StarOffice Basic die Großkleinschreibung wie im vorstehenden Beispiel zu verwenden.)

Anzahl der Seiten, Wörter und Zeichen

Die Anzahl der Seiten, Wörter beziehungsweise Zeichen eines Textes liefern die Textfelder

- `com.sun.star.text.TextField.PageCount`

- com.sun.star.text.TextField.WordCount
- com.sun.star.text.TextField.CharacterCount

zurück. Sie unterstützen folgende Eigenschaft:

- **NumberingType (const)** - Zahlenformat (Vorgaben gemäß Konstanten aus com.sun.star.style.NumberingType).

Aktuelle Seite

Die Nummer der aktuellen Seite lässt sich über das Textfeld com.sun.star.text.TextField.PageNumber in ein Dokument einfügen. Folgende Eigenschaften können dabei angegeben werden:

- **NumberingType (const)** - Zahlenformat (Vorgaben gemäß Konstanten aus com.sun.star.style.NumberingType).
- **Offset (short)** - Offset, der der Seitenzahl hinzugezählt werden soll (auch negative Angaben möglich).

Das folgende Beispiel zeigt wie man eine Seitenzahl in die Fußzeile eines Dokumentes einfügt.

```
Dim Doc As Object
Dim DateTimeField As Object
Dim PageStyles As Object
Dim StdPage As Object
Dim FooterCursor As Object
Dim PageNumber As Object

Doc = StarDesktop.CurrentComponent

PageNumber = Doc.createInstance("com.sun.star.text.TextField.PageNumber")
PageNumber.NumberingType = com.sun.star.style.NumberingType.ARABIC

PageStyles = Doc.StyleFamilies.getByName("PageStyles")

StdPage = PageStyles("Default")
StdPage.FooterIsOn = True

FooterCursor = StdPage.FooterTextLeft.Text.createTextCursor()
StdPage.FooterTextLeft.Text.insertTextContent(FooterCursor, PageNumber, False)
```

Das Beispiel erzeugt zunächst ein Textfeld, das den Service com.sun.star.text.TextField.PageNumber unterstützt. Da die Kopf- und Fußzeilen als Teil der Seitenvorlagen von StarOffice definiert sind, wird dies zunächst über die Liste aller PageStyles ermittelt.

Um sicher zu gehen, dass die Fußzeile tatsächlich sichtbar ist, wird die EigenschaftFooterIsOn auf True gesetzt. Anschließend wird das Textfeld über das zugehörige Text-Objekt der linken Fußzeile in das Dokument eingefügt.

Notizen

Notiz-Felder (com.sun.star.text.TextField.Annotation) sind durch ein kleines gelbes Symbol im Text sichtbar. Ein Klick auf das Symbol öffnet ein Textfeld, das eine Anmerkung zur aktuellen Textstelle aufnehmen kann. Ein Notiz-Feld hat folgende Eigenschaften.

- **`Author (String)`** - Name des Autors.
- **`Content (String)`** - Anmerkungstext.
- **`Date (Date)`** - Erstellungsdatum der Notiz.

Datum / Uhrzeit

Ein Datums-/Uhrzeit-Feld (`com.sun.star.text.TextField.DateTime`) stellt das aktuelle Datum beziehungsweise die aktuelle Uhrzeit dar. Es unterstützt u.a. folgende Eigenschaften:

- **`IsFixed (Boolean)`** – falls `True`, bleiben die Angaben vom Zeitpunkt des Einfügens unverändert erhalten, falls `false`, werden diese bei jedem Öffnen des Dokumentes aktualisiert.
- **`IsDate (Boolean)`** – falls `True` zeigt das Feld das aktuelle Datum an, ansonsten die aktuelle Uhrzeit.
- **`DateTimeValue (struct)`** – Aktueller Inhalt des Feldes (Struktur `com.sun.star.util.DateTime`)
- **`NumberFormat (const)`** – Format zur Darstellung der Uhrzeit beziehungsweise des Datums.

Kapitelname / -nummer

Der Name des aktuellen Kapitels ist über ein Textfeld des Typs `com.sun.star.text.TextField.Chapter` verfügbar. Die Darstellung kann über zwei Eigenschaften angepasst werden.

- **`ChapterFormat (const)`** – bestimmt, ob der Kapitelname oder die Kapitelnummer dargestellt wird (gemäß `com.sun.star.text.ChapterFormat`)
- **`Level (Integer)`** – bestimmt die Kapitelebene, deren Name beziehungsweise Nummer ausgegeben werden soll. Der Wert 0 steht für die höchste verfügbare Ebene.

Lesezeichen (Bookmarks)

Lesezeichen (Service `com.sun.star.text.Bookmark`) gehören zu den `TextContent`-Objekten. Das Erzeugen und Einfügen von Lesezeichen folgt diesem bereits mehrfach beschriebenen Konzept:

```
Dim Doc As Object
Dim Bookmark As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent

Cursor = Doc.Text.createTextCursor()

Bookmark = Doc.createInstance("com.sun.star.text.Bookmark")
Bookmark.Name = "Mein Lesezeichen"
Doc.Text.insertTextContent(Cursor, Bookmark, True)
```

Das Beispiel erzeugt einen `Cursor`, der die Einfügeposition des Bookmarks markiert und dann das eigentliche Lesezeichen-Objekt (`Bookmark`). Das Lesezeichen bekommt anschließend einen Namen zugewiesen und wird via `insertTextContent` an der Position des Cursors (hier der Anfang des Dokumentes) in das Dokument eingefügt.

Der Zugriff auf die Lesezeichen eines Textes erfolgt über eine Liste namens `Bookmarks` des Textdokumentes. Die Lesezeichen sind wahlweise über ihre Nummer oder über ihren Namen erreichbar.

Das folgende Beispiel zeigt, wie sich ein Lesezeichen innerhalb eines Textes auffinden und ein Text an seiner Position einfügen lässt.

```
Dim Doc As Object
Dim Bookmark As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent

Bookmark = Doc.Bookmarks.getByName("Mein Lesezeichen")

Cursor = Doc.Text.createTextCursorByRange(Bookmark.Anchor)
Cursor.String = "Hier steht das Lesezeichen"
```

Im Beispiel dient die Methode `getByName` dazu, das gewünschte Lesezeichen anhand seines Namens zu ermitteln. Danach erzeugt der Aufruf `createTextCursorByRange` einen `Cursor`, der auf die Anker-Position des Lesezeichens positioniert wird. Dort fügt der Cursor dann den gewünschten Text ein.

KAPITEL 7

Tabellendokumente

StarOffice stellt eine umfangreiche Schnittstelle für die programmgesteuerte Erstellung und Bearbeitung von Spreadsheets zur Verfügung. In diesem Kapitel geht es darum, wie sich die betreffenden Services, Methoden und Eigenschaften von Tabellendokumenten nutzen lassen.

Im ersten Abschnitt geht es um den prinzipiellen Aufbau von Tabellen-Dokumenten. Hier erfährt der Leser, wie er auf die Inhalte der einzelnen Zellen zugreifen und diese bearbeiten kann.

Der zweite Abschnitt konzentriert sich auf die effiziente Bearbeitung von Spreadsheets. Im Vordergrund stehen dabei der Service zum Bearbeiten von Zellbereichen und die Möglichkeiten zum Suchen und Ersetzen von Zellinhalten.

Die Unterschiede bei der Arbeit mit Tabellendokumenten zwischen dem alten und dem neuen API von StarOffice fallen geringer aus als in anderen Bereichen. Dies liegt im Wesentlichen daran, dass das alte API über das `Range`-Objekt bereits eine Möglichkeit bot, beliebige Tabellenbereiche zu adressieren. Diese Möglichkeit besteht im neuen API weiterhin und wurde sogar noch ausgebaut.

Der Aufbau von Tabellendokumenten

Die Dokument-Objekte eines Spreadsheets basieren auf dem Service `com.sun.star.sheet.SpreadsheetDocument`. Jedes dieser Dokumente kann mehrere Spreadsheets enthalten. Da der Begriff Spreadsheet sowohl für die betreffenden Dokumente als auch für die einzelnen, darin enthaltenen Seiten verwendet wird, kommt es mitunter zu Begriffsverwirrungen. In diesem Handbuch ist daher von einem *Tabellendokument* oder *Spreadsheet-Dokument* die Rede, wenn es um das gesamte Dokument geht. Ein *Spreadsheet* (oder auch kurz *Sheet*) hingegen bezeichnet eine einzelne Tabelle innerhalb des Dokuments.

Die Terminologie für Tabellendokumente und ihren Inhalt unterscheidet sich zwischen VBA und StarOffice Basic. Während das Dokumentobjekt in VBA *Workbook* und seine einzelnen Seiten *Worksheets* heißen, lauten ihre Bezeichnungen in StarOffice Basic *SpreadsheetDocument* und *Sheet*.

Spreadsheets

Die einzelnen Spreadsheets eines Spreadsheet-Dokuments sind über die Liste `Sheets` verfügbar.

Der Zugriff auf die `Sheets` erfolgt wahlweise über ihre Nummer oder Ihren Namen. Die folgenden Beispiele zeigen, wie man auf ein einzelnes Sheet einerseits über den Namen und andererseits über seine Nummer zugreifen kann.

Beispiel 1: Zugriff über die Nummer (Nummerierung beginnt mit 0)

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet  = Doc. Sheets (0)
```

Beispiel 2: Zugriff über den Namen

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
```

Im Beispiel 1 wird das Sheet über seine Nummer angesprochen (Zählung beginnt hier mit 0). Im zweiten Beispiel erfolgt der Zugriff über den Namen und die Methode `getByName`.

Das über `getByName` gewonnene `Sheet`-Objekt unterstützt den Service `com.sun.star.sheet.Spreadsheet`. Neben mehreren Schnittstellen zum Bearbeiten des Inhaltes stellt er folgende Eigenschaften zur Verfügung:

- **`IsVisible (Boolean)`** – das Spreadsheet ist sichtbar.
- **`PageStyle (String)`** – Name der Seitenvorlage des Spreadsheets.

Sheets erzeugen, löschen und umbenennen

Auch das Erzeugen, Löschen und Umbenennen einzelner Spreadsheets erfolgt über die `Sheets`-Liste des `Spreadsheet`-Dokumentes. Das folgende Beispiel prüft über die Methode `hasByName`, ob ein Spreadsheet mit dem Namen *MySheet* existiert. Falls ja, ermittelt es eine entsprechende Objektreferenz über die Methode `getByName` und legt diese in der Variable in `Sheet` ab. Existiert das betreffende Sheet noch nicht, so wird es über den Aufruf `createInstance` erzeugt und über die Methode `insertByName` in das Spreadsheet-Dokument eingefügt.

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

If Doc.Sheets.hasByName("MySheet") Then
      Sheet = Doc.Sheets.getByName("MySheet")
Else
      Sheet = Doc.createInstance("com.sun.star.sheet.Spreadsheet")
      Doc.Sheets.insertByName("MySheet", Sheet)
End If
```

Die betreffenden Methoden (`getByName`, `insertByName`...) stammen aus der in Kapitel 4 vorgestellten Schnittstelle `com.sun.star.container.XNameContainer`.

Zeilen und Spalten

Jedes Spreadsheet enthält eine Liste mit seinen Zeilen und Spalten. Sie sind über die Eigenschaften `Rows` beziehungsweise `Columns` des Spreadsheet-Objektes verfügbar und unterstützen

die Services `com.sun.star.table.TableColumns` beziehungsweise `com.sun.star.table.TableRows`.

Das folgende Beispiel erzeugt zwei Objekte, die auf die erste Zeile und die erste Spalte eines Spreadsheets verweisen und legt diese in den Objektvariablen `FirstCol` und `FirstRow` ab.

```
Dim Doc As Object
Dim Sheet As Object
Dim FirstRow As Object
Dim FirstCol As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

FirstCol = Sheet.Columns(0)
FirstRow = Sheet.Rows(0)
```

Die Spaltenobjekte unterstützen den Service `com.sun.star.table.TableColumn`, der folgende Eigenschaften bereit hält:

- **`Width (long)`** – Breite der Spalte in 100stel Millimeter.
- **`OptimalWidth (Boolean)`** – setzt die Spalte auf ihre optimale Breite.
- **`IsVisible (Boolean)`** – zeigt die Spalte an.
- **`IsStartOfNewPage (Boolean)`** – erzeugt beim Ausdrucken vor der Spalte einen Seitenumbruch.

Die Zeilenobjekte basieren auf dem Service `com.sun.star.table.RowColumn` mit den Eigenschaften:

- **`Height (long)`** – Höhe der Zeile in 100stel Millimeter.
- **`OptimalHeight (Boolean)`** – setzt die Zeile auf ihre optimale Höhe.
- **`IsVisible (Boolean)`** – zeigt die Zeile an.
- **`IsStartOfNewPage (Boolean)`** – erzeugt beim Ausdrucken vor der Zeile einen Seitenumbruch.

Die Funktionen zur automatischen Anpassung der Spaltenbreite unterscheidet sich leicht von der der Zeilenhöhe. Die Spaltenbreite wird ausschließlich beim Setzen der Eigenschaft `OptimalWidth` auf `True` angepasst. Verändern sich einzelne Zellen, so hat dies keine Auswirkungen auf die Spaltenbreite. Funktional handelt es sich also weniger um eine Eigenschaft als eine Methode.

Erhält die Eigenschaft `OptimalHeight` einer Zeile den Wert `True`, so ändert sich die Zeilenhöhe automatisch, sofern eine Zelle die gesetzte Optimalhöhe überschreitet. Die automatische Anpassung bleibt so lange gültig bis die Zeile über die Eigenschaft `Height` eine absolute Höhe erhält.

Das folgende Beispiel demonstriert den Umgang mit den oben beschriebenen Eigenschaften. Es schaltet die automatische Ermittlung der Höhe für die ersten fünf Zeilen ein und blendet die zweite Spalte aus.

```
Dim Doc As Object
Dim Sheet As Object
Dim Row As Object
```

```
Dim Col As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

For I = 0 To 4
      Row = Sheet.Rows(I)
      Row.OptimalHeight = True
Next I

Col = Sheet.Columns(1)
Col.IsVisible = False
```

Die in VBA vorhandenen Auflistungen `Rows` und `Columns` sind wie in StarOffice Basic über einen Index adressierbar. In StarOffice Basic hat die erste Spalte jedoch den Index 0 und nicht wie in VBA den Index 1.

Zeilen und Spalten einfügen und löschen

Die Objekte `Rows` und `Columns` des Spreadsheets unterstützen nicht nur den Zugriff auf existierende Zeilen beziehungsweise Spalten. Sie gestatten auch deren Einfügen und Löschen.

```
Dim Doc As Object
Dim Sheet As Object
Dim NewColumn As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Sheet.Columns.insertByIndex(3, 1)
Sheet.Columns.removeByIndex(5, 1)
```

Das Beispiel fügt über die Methode `insertByIndex` eine neue Spalte ein, die an vierter Position in der Tabelle erscheint (Index 3, da Nummerierung mit 0 beginnt). Der zweite Parameter von `insertByIndex` gibt die Anzahl der einzufügenden Spalten an (im Beispiel eine).

Der Aufruf `removeByIndex` löscht die sechste Spalte (Index 5, da Nummerierung mit 0 beginnt). Auch hier gibt der zweite Parameter die Anzahl der betroffenen Spalten an.

Die Methoden zum Einfügen und Löschen von Zeilen über das `Rows`-Objekt arbeiten analog zu den dargestellten Methoden zum Bearbeiten der Spalten über das `Columns`-Objekt.

Zellen

Ein Spreadsheet setzt sich zusammen aus einer zweidimensionalen Liste mit Zellen. Jede Zelle ist über ihre X- und Y-Position eindeutig bestimmt. Die linke obere Zelle hat die Position 0/0.

Das nachfolgende Beispiel erzeugt ein Objekt, das auf die Zelle links oben verweist und fügt einen Text in die Zelle ein:

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Cell = Sheet.getCellByPosition(0, 0)
Cell.String = "Test"
```

Neben den nummerischen Koordinaten haben die Zellen auch einen Namen. Die linke obere Zelle eines Spreadsheets heißt `A1`. Der Buchstabe `A` steht dabei für die Spalte, die Ziffer `1` für die Zeile. Es ist wichtig, den *Namen* und die *Position* einer Zelle nicht zu verwechseln, weil die Zählung der Zeilen für den Namen mit 1 beginnt, die für die Position jedoch mit 0.

Eine Tabellenzelle kann in StarOffice wahlweise Text, Zahlen, Formeln oder gar nichts enthalten. Maßgeblich für den Typ ist nicht der Inhalt, der dort abgelegt wurde, sondern die Objekt-Eigenschaft, die für das Eintragen verwendet wurde. Zahlen lassen sich über die Eigenschaft `Value` einfügen und auslesen, Text über die Eigenschaft `String` und Formeln über die Eignenschaft `Formula`.

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Cell = Sheet.getCellByPosition(0, 0)
Cell.Value = 100

Cell = Sheet.getCellByPosition(0, 1)
Cell.String = "Test"

Cell = Sheet.getCellByPosition(0, 2)
Cell.Formula = "=A1"
```

Das Beispiel fügt in die Felder A1 bis A3 je eine Zahl, einen Text und eine Formel ein.

Die Eigenschaften `Value`, `String` und `Formula` ersetzen die Methode `PutCell` zum Setzen der Werte einer Tabellenzelle.

StarOffice betrachtet den über die Eigenschaft `String` eingetragenen Inhalt einer Zelle als Text. Das gilt auch dann, wenn es sich um eine Zahl handelt. Dies sieht man auch daran, dass die Ausgabe einer solchen Zahl linksbündig und nicht – wie bei "echten" Zahlen üblich – rechtsbündig erfolgt. Aber auch beim Einsatz von Formeln ist auf den Unterschied zwischen Text und Zahlen zu achten:

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

Cell = Sheet.getCellByPosition(0, 0)
Cell.Value = 100

Cell = Sheet.getCellByPosition(0, 1)
Cell.String = 1000

Cell = Sheet.getCellByPosition(0, 2)
Cell.Formula = "=A1+A2"

MsgBox Cell.Value
```

Obwohl die Zelle A1 den Wert 100 und die Zelle A2 den Wert 1000 enthält, liefert die Formel A1+A2 den Wert 100 zurück. Dies liegt daran, dass A2 nicht als Zahl, sondern als Zeichenfolge in die Zelle A2 eingetragen wurde und Zeichenfolgen bei einer Addition unberücksichtigt bleiben.

Wenn unklar ist, ob eine Zelle eine Zahl oder eine Zeichenfolge enthält, so hilft die Eigenschaft `Type` weiter:

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 1000

Select Case Cell.Type
Case com.sun.star.table.CellContentType.EMPTY
      MsgBox "Content: Empty"
Case com.sun.star.table.CellContentType.VALUE
      MsgBox "Content: Value"
Case com.sun.star.table.CellContentType.TEXT
      MsgBox "Content: Text"
Case com.sun.star.table.CellContentType.FORMULA
      MsgBox "Content: Formula"
End Select
```

`Cell.Type` liefert einen Wert der Aufzählung `com.sun.star.table.CellContentType`, der verrät, um was für einen Wert es sich handelt. Die möglichen Werte lauten:

- **`EMPTY`** – kein Wert
- **`VALUE`** – Zahl
- **`TEXT`** – Zeichenfolgen
- **`FORMULA`** – Formel

Zellen einfügen, löschen, kopieren und verschieben

Neben der Möglichkeit zum direkten Ändern der Zellinhalte bietet StarOffice Calc eine Schnittstelle zum Einfügen, Löschen, Kopieren und Verschieben von Zellen an. Die über das Spreads-

heet-Objekt erreichbare Schnittstelle (`com.sun.star.sheet.XRangeMovement`) stellt dazu vier Methoden zur Verfügung.

Das Einfügen von Zellen übernimmt die Methode `insertCell`.

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRangeAddress As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

CellRangeAddress.Sheet = 0
CellRangeAddress.StartColumn = 1
CellRangeAddress.StartRow = 1
CellRangeAddress.EndColumn = 2
CellRangeAddress.EndRow = 2

Sheet.insertCells(CellRangeAddress, com.sun.star.sheet.CellInsertMode.DOWN)
```

Das Beispiel fügt im ersten Spreadheet (trägt die Nummer 0) in der zweiten Spalte und Zeile (tragen jeweils die Nummer 1) einen zwei Zeilen hohen und zwei Spalten breiten Bereich ein. Um den dafür notwendigen Platz zu schaffen, werden darunter liegenden Werte entsprechend nach unten geschoben.

Die Definition des einzufügenden Bereiches erfolgt über die Struktur `com.sun.star.table.CellRangeAddress`, die folgende Werte umfasst:

- **`Sheet (short)`** – Nummer des Spreadsheets (Nummerierung beginnt mit 0).
- **`StartColumn (long)`** – Spalte, an der der Bereich eingefügt werden soll (Nummerierung beginnt mit 0).
- **`StartRow (long)`** – Zeile, an der der Bereich eingefügt werden soll (Nummerierung beginnt mit 0).
- **`EndColumn (long)`** – Spalte, bis zu der der einzufügende Bereich reicht (Nummerierung beginnt mit 0).
- **`EndRow (long)`** – Zeile, bis zu der der einzufügende Bereich reicht (Nummerierung beginnt mit 0).

Die ausgefüllte `CellRangeAddress`-Struktur muss der Methode `insertCells` als erster Parameter übergeben werden. Der zweite Parameter von `insertCells` enthält einen Wert der Aufzählung `com.sun.star.sheet.CellInsertMode` und legt fest, was mit den Werten passieren soll, die sich an der Einfügeposition befinden. `CellInsertMode` kennt folgende Werte:

- **`NONE`** – die aktuellen Werte bleiben an ihrem Platz.
- **`DOWN`** – die Zellen an und unterhalb der Einfügeposition werden nach unten verschoben.
- **`RIGHT`** – die Zellen an und rechts neben der Einfügeposition werden nach rechts verschoben.
- **`ROWS`** – die Zeilen ab der Einfügeposition werden nach unten verschoben.
- **`COLUMNS`** – die Spalten ab der Einfügeposition werden nach rechts verschoben.

Das Gegenstück zu der Methode `insertCells` stellt die Methode `removeRange` dar. Sie erwartet ebenfalls eine `CellRangeAddress`-Struktur und löscht den darin definierten Bereich aus der Tabelle.

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRangeAddress As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

CellRangeAddress.Sheet = 0
CellRangeAddress.StartColumn = 1
CellRangeAddress.StartRow = 1
CellRangeAddress.EndColumn = 2
CellRangeAddress.EndRow = 2

Sheet.removeRange(CellRangeAddress, com.sun.star.sheet.CellDeleteMode.UP)
```

Das vorstehende Beispiel entfernt die Zellen des Bereiches B2:C3 und verschiebt die darunter liegenden Zellen zwei Zeilen nach oben. Die Art der Verschiebung wird über einen Wert der Aufzählung `com.sun.star.sheet.CellDeleteMode` definiert, die folgende Werte bereit hält:

- **`NONE`** – die aktuellen Werte bleiben an ihrem Platz.
- **`UP`** – die Zellen an und unterhalb der Einfügeposition werden nach oben verschoben.
- **`LEFT`** – die Zellen an und rechts neben der Einfügeposition werden nach links verschoben.
- **`ROWS`** – die Zeilen ab der Einfügeposition werden nach oben verschoben.
- **`COLUMNS`** – die Spalten ab der Einfügeposition werden nach links verschoben.

Neben den Methoden `insertCells` und `removeRange` bietet die Schnittstelle `XRangeMovement` zwei weitere Methoden an, die es erlauben, Zellen zu verschieben (`moveRange`) beziehungsweise zu kopieren (`copyRange`). Das folgende Beispiel kopiert den Bereich B2:C3 an die Position A6:

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRangeAddress As New com.sun.star.table.CellRangeAddress
Dim CellAddress As New com.sun.star.table.CellAddress

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

CellRangeAddress.Sheet = 0
CellRangeAddress.StartColumn = 1
CellRangeAddress.StartRow = 1
CellRangeAddress.EndColumn = 2
CellRangeAddress.EndRow = 2

CellAddress.Sheet = 0
CellAddress.Column = 0
CellAddress.Row = 5

Sheet.moveRange(CellAddress, CellRangeAddress)
```

Neben der bereits bekannten Struktur `CellRangeAdress`, die den zu verschiebenden Bereich angibt, erwartet `moveRange` eine Struktur `com.sun.star.table.CellAddress`, die den Ursprung des Verschiebungsziels definiert. `CellAddress` hält folgende Felder bereit:

- **`Sheet (short)`** – Nummer des Spreadsheets (Nummerierung beginnt mit 0).
- **`Column (long)`** – Nummer der adressierten Spalte (Nummerierung beginnt mit 0).
- **`Row (long)`** – Nummer der adressierten Zeile (Nummerierung beginnt mit 0).

Die Werte in der Zielregion der Verschiebung werden durch die Methode `moveRange` prinzipiell überschrieben. Ein Parameter für eine automatische Verschiebung wie bei `insertCells` und `removeRange` steht nicht zur Verfügung.

Analog zur Methode `moveRange` arbeitet die Methode `copyRange`. Als einzigen Unterschied verschiebt sie nicht etwa den adressierten Bereich, sondern kopiert ihn.

Die StarOffice Basic-Methoden `insertCell`, `removeRange` und `copyRange` sind vom Funktionsumfang her vergleichbar mit den VBA-Methoden `Range.Insert`, `Range.Delete` und `Range.Copy`. Während sie in VBA auf das betreffende `Range`-Objekt angewendet werden, basieren sie in StarOffice Basic auf dem zugehörigen `Sheet`-Objekt.

Formatierungen

Ein Spreadsheet-Dokument stellt Eigenschaften und Methoden zum Formatieren von Zellen auszudruckenden Seiten zur Verfügung.

Zellattribute

Die Möglichkeiten zur Formatierung von Tabellenzellen sind vielfältig. So lässt sich beispielsweise die Schriftart und -größe eines Textes ändern. Jede Zelle unterstützt dazu die Services `com.sun.star.style.CharacterProperties` sowie `com.sun.star.style.ParagraphProperties`. Die wesentlichen Eigenschaften dieser Schnittstellen sind bereits im Kapitel 6 (*Textdokumente*) beschrieben worden.

Da die genannten Schnittstellen einige Besonderheiten von Spreadsheets außer acht lassen, stellt StarOffice mit `com.sun.star.table.CellProperties` einen weitere Service zur Verfügung, der spezielle Eigenschaften für die Formatierung von Tabellenzellen bereit hält. Die zentralen Eigenschaften dieses Services werden in den folgenden Abschnitten vorgestellt.

Alle genannten Eigenschaften lassen sich sowohl auf einzelne Zellen, als auch auf größere Zellbereiche (etwa Zeilen und Spalten) anwenden.

> Die `CellProperties` des StarOffice API sind vergleichbar mit dem `Interior`-Objekt aus VBA, das ebenfalls zellenspezifische Attribute definiert.

Hintergrundfarbe und Schatten

Der Service `com.sun.star.table.CellProperties` stellt folgende Eigenschaften bereit, die sich auf den Umgang mit Hintergrundfarben und Schatten beziehen:

- **`CellBackColor (Long)`** - Hintergrundfarbe der Tabellenzelle.
- **`IsCellBackgroundTransparent (Boolean)`** - setzt die Hintergrundfarbe transparent.
- **`ShadowFormat (struct)`** - Angaben zum Schatten der Tabellenzelle (Struktur gemäß `com.sun.star.table.ShadowFormat`).

Die Struktur `com.sun.star.table.ShadowFormat` mit den Detailangaben für den Tabellenschatten hat folgenden Aufbau:

- **`Location (enum)`** - Position des Schattens (Wert aus der Struktur `com.sun.star.table.ShadowLocation`).
- **`ShadowWidth (Short)`** - Größe des Schattens in 100stel Millimeter.
- **`IsTransparent (Boolean)`** - setzt den Schatten transparent.
- **`Color (Long)`** - Farbwert des Schattens.

Das folgende Beispiel schreibt die Zahl 1000 in die Zelle B2, ändert die Hintergrundfarbe über die Eigenschaft `CellBackColor` auf rot und legt einen hellgrauen Schatten für die Tabellenzelle an, der um 1 mm nach rechts unten verschoben ist.

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object
Dim ShadowFormat As New com.sun.star.table.ShadowFormat

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 1000

Cell.CellBackColor = RGB(255, 0, 0)

ShadowFormat.Location = com.sun.star.table.ShadowLocation.BOTTOM_RIGHT
ShadowFormat.ShadowWidth = 100
ShadowFormat.Color = RGB(160, 160, 160)

Cell.ShadowFormat = ShadowFormat
```

Ausrichtung

StarOffice stellt verschiedene Funktionen zur Verfügung, über die sich die Ausrichtung eines Textes innerhalb der Tabellenzelle ändern lässt.

Zur Definition der horizontalen und vertikalen Ausrichtung eines Textes stehen folgende Eigenschaften zur Verfügung:

- **`HoriJustify (enum)`** - Horizontale Ausrichtung des Textes (Wert aus `com.sun.star.table.CellHoriJustify`).
- **`VertJustify (enum)`** - Vertikale Ausrichtung des Textes (Wert aus `com.sun.star.table.CellVertJustify`).
- **`Orientation (enum)`** - Laufrichtung des Textes (Wert gemäß `com.sun.star.table.CellOrientation`).
- **`IsTextWrapped (Boolean)`** - erlaubt einen automatischen Umbruch innerhalb der Zelle.
- **`RotateAngle (Long)`** - Drehwinkel des Textes in 100stel Grad.

Das folgende Beispiel zeigt, wie sich der Inhalt einer Tabellenzelle "gestapelt" in der linken oberen Ecke der Zelle darstellen lässt. Die einzelnen Zeichen werden dazu ohne jegliche Drehung untereinander ausgegeben.

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 1000

Cell.HoriJustify = com.sun.star.table.CellHoriJustify.LEFT
Cell.VertJustify = com.sun.star.table.CellVertJustify.TOP
Cell.Orientation = com.sun.star.table.CellOrientation.STACKED
```

Zahlen-, Datums- und Textformat

StarOffice bietet eine ganze Reihe von vordefinierten Datums- und Uhrzeitformaten. Jedes dieser Formate hat eine interne Nummer. Ist die Nummer bekannt, so lässt sich das betreffende Format einer Tabellenzelle über die Eigenschaft `NumberFormat` zuweisen. Da die Nummer eine interne Größe ist, ist es jedoch unmöglich, ohne weitere Hilfsmittel ein passendes Format auszuwählen. StarOffice stellt daher ein Objekt zur Verfügung, das Zugriff auf die existierenden Zahlenformate gewährt und eine Möglichkeit bietet, neue Zahlenformate anzulegen. Der Zugriff auf dieses Objekt erfolgt über den Aufruf

```
NumberFormats = Doc.NumberFormats
```

Seine Methoden `queryKey` und `addNew` gestatten die Suche nach vorhandenen Formaten und bieten die Möglichkeit, neue Formate hinzuzufügen.

Die Angabe eines Formats erfolgt über einen Format-String, der ähnlich dem der Format-Funktion von StarOffice Basic aufgebaut ist. Es existiert jedoch ein wesentlicher Unterschied: Während der Format-Befehl immer englische Abkürzungen und Dezimal- beziehungsweise Tausender-Trennzeichen erwartet, müssen beim Aufbau eines Format-Befehls für das NumberFormats-Objekt die landesspezifischen Abkürzungen verwendet werden.

Das folgende Beispiel formatiert die Tabellenzelle B2 so, dass ihr Inhalt mit drei Nachkommastellen und einem Tausender-Trennzeichen dargestellt wird. Dabei erfolgt die Darstellung in Englisch unter Berücksichtigung der amerikanischen Landeseinstellungen.

```
Dim Doc As Object
Dim Sheet As Object
Dim Cell As Object
Dim NumberFormats As Object
Dim NumberFormatString As String
Dim NumberFormatId As Long
Dim LocalSettings As New com.sun.star.lang.Locale

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
Cell = Sheet.getCellByPosition(1,1)

Cell.Value = 23400.3523565

LocalSettings.Language = "en"
LocalSettings.Country = "us"

NumberFormats = Doc.NumberFormats
NumberFormatString = "#,##0.000"

NumberFormatId = NumberFormats.queryKey(NumberFormatString, LocalSettings, True)
If NumberFormatId = -1 Then
      NumberFormatId = NumberFormats.addNew(NumberFormatString, LocalSettings)
End If

MsgBox NumberFormatId
Cell.NumberFormat = NumberFormatId
```

Einen Überblick zu den verschiedenen Formatierungsmöglichkeiten liefert die Benutzerschnittstelle von StarOffice im Dialog zur Auswahl der Formatierungsinformationen.

Seitenattribute

Unter Seitenattributen versteht man sämtliche Formatierungsangaben, die sich auf die Positionierung des Dokumentinhaltes auf einer Seite beziehen sowie auf optische Elemente, die sich Seite für Seite wiederholen. Hierzu zählen unter anderem

- das zugrunde liegende Papierformat,
- der Seitenrand und
- die Kopf- und Fußzeilen.

Das Vorgehen bei der Definition von Seitenattributen unterscheidet sich von dem anderer Formatierungen. Während der Anwender bei Zellen, Absätzen und Zeichenelementen die Wahl hat, ob er seine Formatierungen direkt an einem bestimmten Dokumentteil anwendet oder indirekt über den Einsatz von Vorlagen festlegt, können Seitenattribute prinzipiell nur über die zugehörige Seitenvorlage definiert werden. Beim Einfügen einer Kopf- oder Fußzeile muss daher beispielsweise nicht das eigentliche Dokument, sondern die verwendete Seitenvorlage um eine Kopf- beziehungsweise Fußzeile erweitert werden.

Nachfolgend findet sich eine Beschreibung der zentralen Attribute zur Formatierung von Dokumentseiten. Ein großer Teil der beschriebenen Vorlagen steht nicht nur innerhalb von Tabellendokumenten zur Verfügung, sondern auch innerhalb von Textdokumenten. Die für beide Dokumentarten gültigen Seitenattribute sind in dem Service `com.sun.star.style.PageProperties` definiert. Diejenigen Attribute, die sich ausschließlich auf Tabellendokumente beziehen, finden sich im Service `com.sun.star.sheet.TablePageStyle`.

Die Seitenattribute (Seitenränder, Rahmen etc.) für ein Microsoft Office-Dokument werden auf der Ebene der `Worksheet`-Objekte (Excel) beziehungsweise `Document`-Objekte (Word) über ein `PageSetup`-Objekt definiert. In StarOffice erfolgt die Definition der Seitenattribute hingegen über eine Seitenvorlage, die wiederum mit dem zugehörigen Dokument verknüpft wird.

Seitenhintergrund

Der Service `com.sun.star.style.PageProperties` definiert folgende Eigenschaften für die Definition eines Seitenhintergrundes:

- **`BackColor (long)`** – Farbe des Hintergrunds.
- **`BackGraphicURL (String)`** – URL der zu verwendenden Hintergrundgrafik.
- **`BackGraphicFilter (String)`** – Name des Filters zur Interpretation der Hintergrundgrafik.
- **`BackGraphicLocation (Enum)`** – Position der Hintergrundgrafik (Wert laut Aufzählung `com.sun.star.style.GraphicLocation`).
- **`BackTransparent (Boolean)`** - stellt den Hintergrund transparent dar.

Seitenformat

Die Definition des Seitenformates erfolgt über folgende Eigenschaften des Service `com.sun.star.style.PageProperties`:

- **IsLandscape (Boolean)** – Ausgabe im Querformat.
- **Width (long)** – Breite der Seite in 100stel Millimeter.
- **Height (long)** – Höhe der Seite in 100stel Millimeter.
- **PrinterPaperTray (String)** – Name des zu verwendenden Drucker-Ausgabeschachtes.

Das folgende Beispiel setzt die Seitengröße der Seitenvorlage "Default" auf die Werte für das Format DIN A5 quer (Höhe 14,8 cm, Breite 21 cm):

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.IsLandscape = True
DefPage.Width = 21000
DefPage.Height = 14800
```

Seitenrand, -rahmen und -schatten

Zur Anpassung des Seitenrandes sowie eventueller Rahmen und Schatten hält der Service `com.sun.star.style.PageProperties` folgende Eigenschaften bereit:

- **LeftMargin (long)** – linker Seitenrand in 100stel Millimeter.
- **RightMargin (long)** – rechter Seitenrand in 100stel Millimeter.
- **TopMargin (long)** – oberer Seitenrand in 100stel Millimeter.
- **BottomMargin (long)** – unterer Seitenrand in 100stel Millimeter.
- **LeftBorder (struct)** – Angaben zur linken Linie des Seitenrahmens (Struktur `com.sun.star.table.BorderLine`).
- **RightBorder (struct)** – Angaben zur rechten Linie des Seitenrahmens (Struktur `com.sun.star.table.BorderLine`).
- **TopBorder (struct)** – Angaben zur oberen Linie des Seitenrahmens (Struktur `com.sun.star.table.BorderLine`).
- **BottomBorder (struct)** – Angaben zur unteren Linie des Seitenrahmens (Struktur `com.sun.star.table.BorderLine`).
- **LeftBorderDistance (long)** – Abstand zwischen linkem Seitenrahmen und Seiteninhalt in 100stel Millimeter.
- **RightBorderDistance (long)** – Abstand zwischen rechtem Seitenrahmen und Seiteninhalt in 100stel Millimeter.
- **TopBorderDistance (long)** – Abstand zwischen oberem Seitenrahmen und Seiteninhalt in 100stel Millimeter.

- **BottomBorderDistance (long)** – Abstand zwischen unterem Seitenrahmen und Seiteninhalt in 100stel Millimeter.
- **ShadowFormat (struct)** – Angaben zum Schatten des `Content`-Bereiches der Seite (Struktur `com.sun.star.table.ShadowFormat`).

Das folgende Beispiel setzt den linken und rechten Rand der Seitenvorlage "Default" auf 1 Zentimeter.

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.LeftMargin = 1000
DefPage.RightMargin = 1000
```

Kopf- und Fußzeilen

Die Kopf- und Fußzeilen eines Dokumentes gehören zu den Seitenattributen und werden ebenfalls über den Service `com.sun.star.style.PageProperties` definiert. Die zentralen Eigenschaften für Kopfzeilen lauten:

- **HeaderIsOn (Boolean)** – Kopfzeile ist aktiviert.
- **HeaderLeftMargin (long)** – Abstand zwischen Kopfzeile und linkem Seitenrand in 100stel Millimeter.
- **HeaderRightMargin (long)** – Abstand zwischen Kopfzeile und rechtem Seitenrand in 100stel Millimeter.
- **HeaderBodyDistance (long)** – Abstand zwischen Kopfzeile und Hauptbereich des Dokumentes in 100stel Millimeter.
- **HeaderHeight (long)** – Höhe der Kopfzeile in 100stel Millimeter.
- **HeaderIsDynamicHeight (Boolean)** – die Höhe der Kopfzeile wird automatisch an den Inhalt angepasst.
- **HeaderLeftBorder (struct)** - Angaben zur linken Linie des Rahmens um die Kopfzeile (Struktur `com.sun.star.table.BorderLine`).
- **HeaderRightBorder (struct)** - Angaben zur rechten Linie des Rahmens um die Kopfzeile (Struktur `com.sun.star.table.BorderLine`).
- **HeaderTopBorder (struct)** - Angaben zur oberen Linie des Rahmens um die Kopfzeile (Struktur `com.sun.star.table.BorderLine`).
- **HeaderBottomBorder (struct)** - Angaben zur unteren Linie des Rahmens um die Kopfzeile (Struktur `com.sun.star.table.BorderLine`).

- **HeaderLeftBorderDistance (long)** – Abstand zwischen linker Rahmenlinie und Inhalt der Kopfzeile in 100stel Millimeter.
- **HeaderRightBorderDistance (long)** – Abstand zwischen rechter Rahmenlinie und Inhalt der Kopfzeile in 100stel Millimeter.
- **HeaderTopBorderDistance (long)** – Abstand zwischen oberer Rahmenlinie und Inhalt der Kopfzeile in 100stel Millimeter.
- **HeaderBottomBorderDistance (long)** – Abstand zwischen unterer Rahmenlinie und Inhalt der Kopfzeile in 100stel Millimeter.
- **HeaderIsShared (Boolean)** – die Kopfzeilen auf den linken und rechten Seiten enthalten den gleichen Inhalt (siehe `HeaderText`, `HeaderTextLeft` und `HeaderTextRight`).
- **HeaderBackColor (long)** – Hintergrundfarbe der Kopfzeile.
- **HeaderBackGraphicURL (String)** – URL der zu verwendenden Hintergrundgrafik.
- **HeaderBackGraphicFilter (String)** – Name des Filters zur Interpretation der Hintergrundgrafik für die Kopfzeile.
- **HeaderBackGraphicLocation (Enum)** – Position der Hintergrundgrafik für die Kopfzeile (Wert laut Aufzählung `com.sun.star.style.GraphicLocation`).
- **HeaderBackTransparent (Boolean) –** zeigt den Hintergrund der Kopfzeile transparent an.
- **HeaderShadowFormat (struct)** – Angaben zum Schatten der Kopfzeile (Struktur `com.sun.star.table.ShadowFormat`).

Die Eigenschaften zur Formatierung der Fußzeilen lauten:

- **FooterIsOn (Boolean)** – Fußzeile ist aktiviert.
- **FooterLeftMargin (long)** – Abstand zwischen Fußzeile und linkem Seitenrand in 100stel Millimeter.
- **FooterRightMargin (long)** – Abstand zwischen Fußzeile und rechtem Seitenrand in 100stel Millimeter.
- **FooterBodyDistance (long)** – Abstand zwischen Fußzeile und Hauptbereich des Dokumentes in 100stel Millimeter.
- **FooterHeight (long)** – Höhe der Fußzeile in 100stel Millimeter.
- **FooterIsDynamicHeight (Boolean)** – die Höhe der Fußzeile wird automatisch an den Inhalt angepasst.
- **FooterLeftBorder (struct)** - Angaben zur linken Linie des Rahmens um die Fußzeile (Struktur `com.sun.star.table.BorderLine`).
- **FooterRightBorder (struct)** - Angaben zur rechten Linie des Rahmens um die Fußzeile (Struktur `com.sun.star.table.BorderLine`).
- **FooterTopBorder (struct)** - Angaben zur oberenLinie des Rahmens um die Fußzeile (Struktur `com.sun.star.table.BorderLine`).

- **FooterBottomBorder (struct)** - Angaben zur unteren Linie des Rahmens um die Fußzeile (Struktur `com.sun.star.table.BorderLine`).
- **FooterLeftBorderDistance (long)** – Abstand zwischen linker Rahmenlinie und Inhalt der Fußzeile in 100stel Millimeter.
- **FooterRightBorderDistance (long)** – Abstand zwischen rechter Rahmenlinie und Inhalt der Fußzeile in 100stel Millimeter.
- **FooterTopBorderDistance (long)** – Abstand zwischen oberer Rahmenlinie und Inhalt der Fußzeile in 100stel Millimeter.
- **FooterBottomBorderDistance (long)** – Abstand zwischen unterer Rahmenlinie und Inhalt der Fußzeile in 100stel Millimeter.
- **FooterIsShared (Boolean)** – die Fußzeilen auf den linken und rechten Seiten enthalten den gleichen Inhalt (siehe `FooterText`, `FooterTextLeft` und `FooterTextRight`).
- **FooterBackColor (long)** – Hintergrundfarbe der Fußzeile.
- **FooterBackGraphicURL (String)** – URL der zu verwendenden Hintergrundgrafik.
- **FooterBackGraphicFilter (String)** – Name des Filters zur Interpretation der Hintergrundgrafik für die Fußzeile.
- **FooterBackGraphicLocation (Enum)** – Position der Hintergrundgrafik für die Fußzeile (Wert laut Aufzählung `com.sun.star.style.GraphicLocation`).
- **FooterBackTransparent (Boolean) –** zeigt den Hintergrund der Fußzeile transparent an.
- **FooterShadowFormat (struct)** – Angaben zum Schatten des Fußzeile (Struktur `com.sun.star.table.ShadowFormat`).

Texte von Kopf- und Fußzeilen ändern

Beim Einfügen von Text in die Kopf- und Fußzeilen sind einige Besonderheiten zu beachten. Der Zugriff auf die Kopf- und Fußzeilen eines Tabellendokumentes erfolgt über die Eigenschaften

- **LeftPageHeaderContent (Object) –** Inhalt der Kopfzeilen für die linken Seiten des Dokumentes (Service `com.sun.star.sheet.HeaderFooterContent`).
- **RightPageHeaderContent (Object) –** Inhalt der Kopfzeilen für die rechten Seiten des Dokumentes (Service `com.sun.star.sheet.HeaderFooterContent`).
- **LeftPageFooterContent (Object) –** Inhalt der Fußzeilen für die linken Seiten des Dokumentes (Service `com.sun.star.sheet.HeaderFooterContent`).
- **RightPageFooterContent (Object) –** Inhalt der Fußzeilen für die rechten Seiten des Dokumentes (Service `com.sun.star.sheet.HeaderFooterContent`).

Ist keine Unterscheidung zwischen rechter und linker Kopf- beziehungsweise Fußzeile gewünscht (die Eigenschaft `FooterIsShared` ist `False`), so sind die Eigenschaften für die rechte Kopf- beziehungsweise Fußzeile zu verwenden.

Alle genannten Objekte liefern ein Objekt zurück, das den Service `com.sun.star.sheet.HeaderFooterContent` unterstützt. Dieser Service stellt über die (unechten) Eigenschaften `LeftText`, `CenterText` und `RightText` drei Textelemente für die Kopf- und Fußzeilen von StarOffice Calc zur Verfügung.

Das folgende Beispiel schreibt in das linke Textfeld der Kopfzeile aus der Vorlage "Default" den Wert "Just a Test.".

```
Dim Doc As Object
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object
Dim HText As Object
Dim HContent As Object
Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.HeaderIsOn = True
HContent = DefPage.RightPageHeaderContent
HText = HContent.LeftText
HText.String = "Just a Test."
DefPage.RightPageHeaderContent = HContent
```

An dem Beispiel ist insbesondere die letzte Zeile zu beachten. Nach dem Ändern des Textes muss das `TextContent`-Objekt erneut der Kopfzeile zugewiesen werden, damit die Änderung wirksam wird.

Für Textdokumente (StarOffice Writer) existiert ein anderer Mechanismus zum Ändern der Kopf- und Fußzeilentexte, da diese dort aus einem einzigen Textblock bestehen. Dort werden die folgenden, im Service `com.sun.star.style.PageProperties` definierten Eigenschaften eingesetzt:

- **`HeaderText (Object)`** – Text-Objekt mit Inhalt der Kopfzeile (Service `com.sun.star.text.XText`).
- **`HeaderTextLeft (Object)`** – Text-Objekt mit Inhalt der Kopfzeilen auf den linken Seiten (Service `com.sun.star.text.XText`).
- **`HeaderTextRight (Object)`** – Text-Objekt mit Inhalt der Kopfzeilen auf den rechten Seiten (Service `com.sun.star.text.XText`).
- **`FooterText (Object)`** – Text-Objekt mit Inhalt der Fußzeile (Service `com.sun.star.text.XText`).
- **`FooterTextLeft (Object)`** – Text-Objekt mit Inhalt der Fußzeilen auf den linken Seiten (Service `com.sun.star.text.XText`).
- **`FooterTextRight (Object)`** – Text-Objekt mit Inhalt der Fußzeilen auf den rechten Seiten (Service `com.sun.star.text.XText`).

Das folgende Beispiel legt in der Vorlage "Default" eines Textdokumentes eine Kopfzeile an, die ebenfalls den Text "Just a Test." enthält.

```
Dim Doc As Object
```

```
Dim Sheet As Object
Dim StyleFamilies As Object
Dim PageStyles As Object
Dim DefPage As Object
Dim HText As Object

Doc = StarDesktop.CurrentComponent
StyleFamilies = Doc.StyleFamilies
PageStyles = StyleFamilies.getByName("PageStyles")
DefPage = PageStyles.getByName("Default")

DefPage.HeaderIsOn = True
HText = DefPage.HeaderText

HText.String = "Just a Test."
```

Der Zugriff erfolgt in diesem Falle anstelle des `HeaderFooterContent`-Objektes direkt über die Eigenschaft `HeaderText` der Seitenvorlage.

Zentrierung (nur Tabellendokumente)

Der ausschließlich in den Seitenvorlagen von StarOffice Calc zum Einsatz kommende Service `com.sun.star.sheet.TablePageStyle` gestattet es, den zu druckenden Tabellenbereich auf der Seite zu zentrieren. Hierfür bietet er folgende Eigenschaften an:

- **`CenterHorizontally (Boolean)`** – Tabelleninhalt wird horizontal zentriert.
- **`CenterVertically (Boolean)`** – Tabelleninhalt wird vertikal zentriert.

Festlegung der auszudruckenden Elemente (nur Tabellendokumente)

Bei der Seitenformatierung von Tabellen kann der Anwender festlegen, welche Elemente einer Seite bei einem Ausdruck sichtbar sein sollen und welche nicht. Hierzu hält der Service `com.sun.star.sheet.TablePageStyle` folgende Eigenschaften bereit:

- **`PrintAnnotations (Boolean)`** – druckt Zellanmerkungen mit aus.
- **`PrintGrid (Boolean)`** – druckt das Gitternetz der Tabelle mit aus.
- **`PrintHeaders (Boolean)`** – druckt die Zeilen- und Spaltentitel mit aus.
- **`PrintCharts (Boolean)`** – druckt Diagramme innerhalb der Tabellen mit aus.
- **`PrintObjects (Boolean)`** – druckt eingebettete Objekte mit aus.
- **`PrintDrawing (Boolean)`** – druckt Zeichenelement-Objekte mit aus.
- **`PrintDownFirst (Boolean)`** – druckt bei mehrseitigen Tabellen zuerst die unten anschließenden Seiten und dann die seitlich anschließenden Seiten aus.
- **`PrintFormulas (Boolean)`** – druckt die Formeln statt der berechneten Werte aus.
- **`PrintZeroValues (Boolean)`** – druckt Nullwerte mit aus.

Tabellendokumente effizient bearbeiten

In den vorstehenden Abschnitten ging es um den prinzipiellen Aufbau von Spreadsheet-Dokumenten. Die genannten Services bieten zwar Zugriff auf alle Zellen eines Spreadsheets, der Weg über das Abfragen einzelner Zellen kann jedoch umständlich sein. StarOffice stellt daher einige Services zur Verfügung, die einen vereinfachten Zugriff auf größere Tabellenbereiche gestatten.

Zellbereiche

Neben einem Objekt für Einzelzellen (Service `com.sun.star.table.Cell`) stellt StarOffice Objekte zur Verfügung, die einen größeren Tabellenbereich repräsentieren. Die Erstellung von solchen `CellRange`-Objekten erfolgt über den Aufruf `getCellRangeByName` des Spreadsheet-Objektes:

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
CellRange = Sheet.getCellRangeByName("A1:C15")
```

Die Angabe des Zellbereiches erfolgt über die in Tabellenkalkulationen übliche Doppelpunkt-Notation. Der Zellbereich A1:C15 beispielsweise umfasst alle Zellen in den Spalten A, B und C, die gleichzeitig in den Zeilen 1 bis 15 liegen.

Die einzelnen Zellen eines Zellbereiches lassen sich über die Methode `getCellByPosition` ermitteln, wobei die linke obere Zelle des Zellbereiches die Position 0, 0 besitzt. Das folgende Beispiel erzeugt mit dieser Methode ein Objekt der Zelle C3.

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object
Dim Cell As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
CellRange = Sheet.getCellRangeByName("B2:D4")
Cell = CellRange.GetCellByPosition(1, 1)
```

Formatierung von Zellbereichen

Zellbereiche unterstützen wie Einzelzellen den Service `com.sun.star.table.CellProperties`, der eine Formatierung von Zellbereichen gestattet. Informationen und Beispiele zu diesem Service finden sich im Abschnitt *Formatierungen*.

Rechnen mit Zellbereichen

Zellbereiche bieten mit der Methode `computeFunction` eine einfache Möglichkeit zur Durchführung mathematischer Operationen auf der Grundlage des Zellbereiches an. `ComputeFunction` erwartet dazu eine Konstante als Parameter, die die anzuwendende mathematische Funktion beschreibt. Die zugehörigen Konstanten sind in der Aufzählung `com.sun.star.sheet.GeneralFunction` definiert. Folgende Werte stehen zur Verfügung:

- **SUM** - Summe aller nummerischen Werte
- **COUNT** - Anzahl aller Werte (einschließlich nicht-nummerischer Werte)
- **COUNTNUMS** - Anzahl aller nummerischen Werte
- **AVERAGE** - Durchschnitt aller nummerischen Werte
- **MAX** - größter nummerischer Wert
- **MIN** - kleinster nummerischer Wert
- **PRODUCT** - Produkt aller nummerischen Werte
- **STDEV** - Standard-Abweichung
- **VAR** - Varianz
- **STDEVP** - Standard-Abweichung auf der Grundlage der Gesamtpopulation
- **VARP** - Varianz auf der Grundlage der Gesamtpopulation

Das folgende Beispiel berechnet den Mittelwert des Bereiches A1:C3 und gibt das Ergebnis in einem Meldungsfenster aus:

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
CellRange = Sheet.getCellRangeByName("A1:C3")

MsgBox CellRange.computeFunction(com.sun.star.sheet.GeneralFunction.AVERAGE)
```

Zellinhalte löschen

Um das Löschen von Zellinhalten zu vereinfachen halten Zellbereiche die Methode `clearContents` bereit. Sie löscht Inhalte eines bestimmten Typs in einem Zellbereich.

Das folgende Beispiel entfernt alle Zeichenfolgen sowie die direkten Formatierungsinformationen aus dem Bereich B2:C3.

```
Dim Doc As Object
Dim Sheet As Object
Dim CellRange As Object
Dim Flags As Long

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)
CellRange = Sheet.getCellRangeByName("B2:C3")

Flags = com.sun.star.sheet.CellFlags.STRING + _
            com.sun.star.sheet.CellFlags.HARDATTR

CellRange.clearContents(Flags)
```

Die in `clearContents` angegebenen Flags entstammen der Konstantenliste `com.sun.star.sheet.CellFlags`, die folgende Elemente bereit hält:

- **VALUE –** nummerische Werte, die nicht als Datum oder Uhrzeit formatiert sind.

- **DATETIME** – nummerische Werte, die als Datum oder Uhrzeit formatiert sind
- **STRING** - Zeichenfolgen
- **ANNOTATION** – mit den Zellen verknüpfte Anmerkungen
- **FORMULA** – Formeln
- **HARDATTR** – direkte Formatierungen der Zellen
- **STYLES** – indirekte Formatierungen
- **OBJECTS** – Zeichenobjekte, die mit den Zellen verbunden sind
- **EDITATTR** – Zeichenformatierungen, die nur für Teile der Zellen gelten

Sollen verschiedene Informationen mit einem Aufruf von `clearContents` gelöscht werden, so können die Konstanten addiert werden.

Suchen und Ersetzen von Zellinhalten

Tabellendokumente bieten ähnlich wie Textdokumente eine Funktion zum Suchen und Ersetzen. Das Vorgehen orientiert sich dabei an dem in Textdokumenten eingesetzten Verfahren.

Die Deskriptor-Objekte zum Suchen und Ersetzen werden in Tabellendokumenten jedoch nicht direkt über das Dokumentobjekt erzeugt, sondern über die `Sheets`-Liste. Das folgende Beispiel verdeutlicht einen Ersetzungsvorgang:

```
Dim Doc As Object
Dim Sheet As Object
Dim ReplaceDescriptor As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets(0)

ReplaceDescriptor = Sheet.createReplaceDescriptor()
ReplaceDescriptor.SearchString = "is"
ReplaceDescriptor.ReplaceString = "was"

For I = 0 to Doc.Sheets.Count - 1
      Sheet = Doc.Sheets(I)
      Sheet.ReplaceAll(ReplaceDescriptor)
Next I
```

Das Beispiel erzeugt über die erste Seite des Dokumentes einen `ReplaceDescriptor` und wendet diesen anschließend in einer Schleife auf alle Seiten an.

Zeichnungen und Präsentationen

Dieses Kapitel führt in die programmgesteuerte Erstellung und Bearbeitung von Zeichnungen ein. Im ersten Abschnitt geht es zunächst um den prinzipiellen Aufbau von Zeichnungen und die Basiselemente, aus denen sich eine Zeichnung zusammen setzt.

Im zweiten Abschnitten werden komplexere Bearbeitungsfunktionen wie das Gruppieren, Drehen und Skalieren von Objekten besprochen.

Hinweise zum Erzeugen, Öffnen und Speichern von Zeichnungen finden sich im Kapitel5, *Die Arbeit mit StarOffice-Dokumenten.*

Der Aufbau von Zeichnungen

Ein Zeichendokument umfasst eine beliebige Anzahl von Seiten. Jede dieser Seiten kann getrennt gestaltet werden. Auf jeder dieser Seiten lassen sich beliebig viele Zeichenelemente platzieren.

Ein wenig verkompliziert sich das Bild jedoch, da neben den Seiten auch noch verschiedene *Zeichenebenen* zur Verfügung stehen. Die per Voreinstellung vorhandenen Ebenen heißen *Layout*, *Controls* und *Dimension Lines*. Zu diesen vorhandenen Ebenen lassen sich neue Ebenen hinzufügen.

Per Voreinstellung befinden sich die Zeichenelemente auf der Ebene *Layout*. Diese lässt sich jedoch ändern, so dass die Zeichenelemente auch in andere Ebenen eingefügt werden können. Aus Platzgründen wird an dieser Stelle auf eine Beschreibung dieser Möglichkeiten verzichtet. Weitere Informationen zu diesem Thema finden Sie im StarOffice Developer's Guide.

Seiten

Die Seiten eines Zeichendokumentes sind über die Liste `DrawPages` verfügbar. Der Zugriff auf die einzelnen Seiten erfolgt wahlweise über ihre Nummer oder Ihren Namen. Enthält ein Dokument eine Seite und heißt diese *Slide 1*, so sind die folgenden Beispiele identisch.

Beispiel 1:

```
Dim Doc As Object
Dim Page As Object

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)
```

Beispiel 2:

```
Dim Doc As Object
```

```
Dim Page As Object

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages.getByName("Slide 1")
```

Im Beispiel 1 wird die Seite über ihre Nummer angesprochen (Zählung beginnt hier mit 0). Im zweiten Beispiel erfolgt der Zugriff über den Namen und die Methode `getByName`.

```
Dim sUrl As String, sFilter As String
Dim sOptions As String
Dim oSheets As Object, oSheet As Object

      oSheets = oDocument.Sheets

      If oSheets.hasByName("Link") Then
            oSheet = oSheets.getByName("Link")
      Else
            oSheet = oDocument.createInstance("com.sun.star.sheet.Spreadsheet")
            oSheets.insertByName("Link", oSheet)
            oSheet.IsVisible = False
      End If
```

Der oben stehende Aufruf liefert ein Seitenobjekt zurück, das den Service `com.sun.star.drawing.DrawPage` unterstützt. Der Service kennt folgende Eigenschaften:

- **`BorderLeft (Long)`** – Linker Seitenrand in 100stel Millimeter.
- **`BorderRight (Long)`** – Rechter Seitenrand in 100stel Millimeter.
- **`BorderTop (Long)`** – Oberer Seitenrand in 100stel Millimeter.
- **`BorderBottom (Long)`** – Unterer Seitenrand in 100stel Millimeter.
- **`Width (Long)`** – Seitenbreite in 100stel Millimeter.
- **`Height (Long)`** – Seitenhöhe in 100stel Millimeter.
- **`Number (Short)`** – Nummer der Seite (hier beginnt die Nummerierung mit 1), schreibgeschützt.
- **`Orientation (Enum)`** – Seitenausrichtung (gemäß Aufzählung `com.sun.star.view.PaperOrientation`).

Ändern Sie diese Einstellungen, so wirkt sich das auf *alle* Seiten des Dokumentes aus.

Das folgende Beispiel setzt die Seitengröße des gerade geöffneten Zeichendokumentes auf 20 × 20 Zentimeter bei einem Seitenrand von jeweils 0,5 Zentimenter:

```
Dim Doc As Object
Dim Page As Object

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

Page.BorderLeft = 500
Page.BorderRight = 500
Page.BorderTop = 500
Page.BorderBottom = 500

Page.Width = 20000
Page.Height = 20000
```

Zeichenobjekte und ihre elementaren Eigenschaften

Zu den Zeichenobjekten zählen Formen(Rechtecke, Kreise usw.), Linien und Textobjekte. Alle diese Objekte besitzen eine Reihe von Gemeinsamkeiten und unterstützen den Service `com.sun.star.drawing.Shape`, der die nachgebildeten Eigenschaften `Size` und `Position` zur Verfügung stellt.

StarOffice Basic bietet außerdem weitere Services an, über die man Eigenschaften wie Formatierungen oder Füllungen verändern kann. Die zur Verfügung stehenden Formatierungsmöglichkeiten sind vom Typ des Zeichenobjektes abhängig.

Das folgende Beispiel zeigt, wie sich ein Rechteck erstellen und in ein Zeichendokument einfügen lässt:

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

Page.add(RectangleShape)
```

Das Beispiel ermittelt über den Aufruf `StarDesktop.CurrentComponent` das aktuell geöffnete Dokument. Das so gewonnene Dokumentobjekt liefert über den Aufruf `drawPages(0)` die erste Seite der Zeichnung zurück.

Im Anschluss gilt es, die Strukturen `Point` und `Size` mit Ursprungspunkt (linke obere Ecke) und Größe des Zeichenobjektes zu initialisieren. Die Längenangaben erfolgen in 100stel Millimeter.

Sodann erzeugt der Programmcode über den bereits mehrfach verwendeten Aufruf `Doc.createInstance` ein Objekt mit dem gewünschten Zeichenelement. Der Service-Name `com.sun.star.drawing.RectangleShape` legt dabei fest, dass es sich um ein Rechteck handelt. Schließlich wird das Zeichenobjekt über einen Aufruf `Page.add` der Seite zugewiesen.

Flächenattribute

Die Flächenattribute sind in dem Service `com.sun.star.drawing.FillProperties` zusammen gefasst.

StarOffice kennt vier prinzipielle Arten, eine Fläche zu formatieren. Die einfachste Variante stellt eine einfarbige Fläche dar. Zusätzliche Farben bringen die Möglichkeiten zur Definition von Farbverlauf und Linienmuster ins Spiel. Schließlich besteht als vierte Variante die Möglichkeit, eine bereits vorhandene Grafik in die Fläche hinein zu projizieren.

Der Füllmodus eines Zeichenobjektes wird über die Eigenschaft `FillStyle` festgelegt. Die zugelassenen Werte sind in `com.sun.star.drawing.FillStyle` definiert.

Einfarbige Flächen

Die zentrale Eigenschaft für einfarbige Flächen lautet

- **`FillColor (Long)`** – Füllfarbe der Fläche.

Um den Füllmodus zu verwenden, muss der Eigenschaft `FillStyle` des weiteren der Füllmodus `SOLID` zugewiesen werden.

Das folgende Beispiel erzeugt ein Rechteck mit der Füllfarbe Rot (RGB-Wert 255, 0, 0):

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.SOLID
RectangleShape.FillColor = RGB(255,0,0)

Page.add(RectangleShape)
```

Farbverlauf

Jede Fläche eines StarOffice-Dokumentes lässt sich auch mit einem Farbverlauf ausstatten. Hierzu muss die Eigenschaft `FillStyle` auf `GRADIENT` gesetzt werden.

Ist der gewünschte Farbverlauf bereits in StarOffice definiert, so genügt es, den zugehörigen Namen der Eigenschaft `FillTransparenceGradientName` zuzuweisen. Wer hingegen einen eigenen Farbverlauf definieren möchte, muss eine Struktur `com.sun.star.awt.Gradient` ausfüllen, um die Eigenschaft `FillGradient` zuzuweisen. Die Eigenschaft hält folgende Attribute bereit:

- **`Style (Enum)`** - Gradientenart, zum Beispiel linear oder radial (Vorgabewerte gemäß `com.sun.star.awt.GradientStyle`).
- **StartColor (Long)** - Anfangsfarbe des Farbverlaufs.
- **EndColor (Long)** - Endfarbe des Farbverlaufes.
- **Angle (Short)** - Drehwinkel des Farbverlaufs in 10tel Grad.
- **XOffset (Short)** - X-Koordinate, bei der der Farbverlauf beginnt, Angaben in 100stel Millimeter.

- **YOffset (Short)** - Y-Koordinate, bei der der Farbverlauf beginnt, Angaben in 100stel Millimeter.
- **StartIntensity (Short)** - Helligkeit der `StartColor` in Prozent (in StarOffice Basic sind im Gegensatz zum GUI auch Werte über 100 Prozent zugelassen).
- **EndIntensity (Short)** - Helligkeit der `EndColor` in Prozent (in StarOffice Basic sind im Gegensatz zum GUI auch Werte über 100 Prozent zugelassen).
- **StepCount (Short)** - Anzahl der Farbabstufungen, die StarOffice für den Gradienten berechnen soll.

Das folgende Beispiel demonstriert den Einsatz von Farbverläufen unter Zuhilfenahme der Struktur `com.sun.star.awt.Gradient`.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Dim Gradient As New com.sun.star.awt.Gradient

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

Gradient.Style = com.sun.star.awt.GradientStyle.LINEAR
Gradient.StartColor = RGB(255,0,0)
Gradient.EndColor = RGB(0,255,0)
Gradient.StartIntensity = 150
Gradient.EndIntensity = 150
Gradient.Angle = 450
Gradient.StepCount = 100

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.GRADIENT
RectangleShape.FillGradient = Gradient

Page.add(RectangleShape)
```

Es erzeugt einen linearen Farbverlauf (`Style = LINEAR`), der in der linken oberen Ecke mit der Farbe rot (`StartColor`) beginnt und mit einem Winkel von 45 Grad (`Angle`) nach rechts unten verläuft, wo er die Farbe Grün (`EndColor`) annimmt. Die Farbintensität der Start- und Endfarben beträgt jeweils 150 Prozent (`StartIntensity` und `EndIntensity`), was dazu führt, dass die Farben heller erscheinen, als die in `StartColor` und `EndColor` angegebenen Grundwerte. Die Darstellung des Farbverlaufes erfolgt über 100 abgestufte Einzelfarben (`StepCount`).

Linienmuster

Um eine Fläche mit einem Linienmuster darzustellen, muss ihrer Eigenschaft `FillStyle` der Wert `HATCH` zugewiesen werden. Der Programmcode zur Definition des Linienmusters ähnelt stark dem von Farbverläufen. Auch hier kommt eine Hilfstruktur zum Einsatz, die das Aussehen

festlegt. Die Struktur für Linienmuster lautet `com.sun.star.drawing.Hatch` und besitzt folgende Attribute:

- **`Style (Enum)`** - Art des Linienmusters: einfach, kariert oder kariert mit Diagonalen (Vorgabewerte gemäß `com.sun.star.awt.HatchStyle`).
- **`Color (Long)`** - Farbe der Linien.
- **`Distance (Long)`** - Abstand zwischen den Linien in 100stel Millimeter.
- **`Angle (Short)`** - Drehwinkel des Linienmusters in 10tel Grad.

Das folgende Beispiel demonstriert den Einsatz einer Hatch-Struktur:

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Dim Hatch As New com.sun.star.drawing.Hatch

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.HATCH

Hatch.Style = com.sun.star.drawing.HatchStyle.SINGLE
Hatch.Color = RGB(64,64,64)
Hatch.Distance = 20
Hatch.Angle = 450

RectangleShape.FillHatch = Hatch

Page.add(RectangleShape)
```

Es erzeugt ein einfache Schraffur (`HatchStyle` = `SINGLE`), deren Linien um 45 Grad gedreht sind (`Angle`). Die Linien sind in einem dunklen grau (`Color`) dargestellt und ihr Abstand (`Distance`) beträgt 0,2 Millimeter.

Bitmaps

Bleibt die vierte Variante zur Formatierung von Flächen: Die Bitmap-Projektion. Um diese Art der Formatierung zu verwenden, muss die Eigenschaft `FillStyle` den Wert `BITMAP` besitzen. Ist das Bitmap in StarOffice bereits eingebunden, so genügt es, seinen Namen in die Eigenschaft `FillBitMapName` einzutragen und die gewünschte Darstellungsart (einfach, gekachelt oder gestreckt) über die Eigenschaft `FillBitmapMode` festzulegen (Vorgabewerte gemäß `com.sun.star.drawing.BitmapMode`).

Möchten Sie hingegen mit einer externen Datei arbeiten, so können Sie die zugehörige URL über die Eigenschaft `FillBitmapURL` definieren.

Das folgende Beispiel erzeugt ein Rechteck, auf dessen Fläche das in StarOffice vorhandene Bitmap Sky projiziert wird. Das Bitmap wird dabei in Form von Kacheln neben- und untereinander so oft wiederholt, dass es die komplette Fläche des Rechtecks bedeckt.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.BITMAP

RectangleShape.FillBitmapName = "Sky"
RectangleShape.FillBitmapMode = com.sun.star.drawing.BitmapMode.REPEAT


Page.add(RectangleShape)
```

Transparenz

StarOffice bietet die Möglichkeit, Flächen transparent darzustellen, so dass die dahinter liegenden Objekte durchschimmern. Die einfachste Möglichkeit, ein Zeichenelement transparent darzustellen bietet die Eigenschaft `FillTransparence`, über die sich die prozentuale Transparenz einer Fläche festlegen lässt.

Das folgende Beispiel erzeugt ein einfarbig rotes Rechteck mit einer Transparenz von 50 Prozent.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.FillStyle = com.sun.star.drawing.FillStyle.SOLID
RectangleShape.FillTransparence = 50
RectangleShape.FillColor = RGB(255,0,0)

Page.add(RectangleShape)
```

Erhält die Eigenschaft `FillTransparence` den Wert 100, so ist die Fläche des zugehörigen Zeichenobjektes komplett durchsichtig.

Neben der Eigenschaft `FillTransparence` hält der Service `com.sun.star.drawing.FillProperties` auch die Eigenschaft `FillTransparenceGradient` bereit, über die sich ein Verlauf definieren lässt, der die Transparenz einer Fläche angibt.

Linienattribute

Alle Zeichenobjekte, die eine äußere Begrenzungslinie enthalten können, unterstützen den Service `com.sun.star.drawing.LineStyle`. Er bietet unter anderem folgende Eigenschaften an, die das Aussehen einer Linie bestimmen können:

- **`LineStyle (Enum)`** - Linienart (Vorgabewerte gemäß `com.sun.star.drawing.LineStyle`).
- **`LineColor (Long)`** - Linienfarbe.
- **`LineTransparence (Short)`** - Linientransparenz.
- **`LineWidth (Long)`** - Linienstärke in 100stel Millimeter.
- **`LineJoint (Enum)`** - Übergänge an Verbindungsstellen (Vorgabewerte gemäß `com.sun.star.drawing.LineJoint`).

Das nachfolgende Beispiel erzeugt ein Rechteck mit einer durchgezogenen Linie (`LineStyle` = `SOLID`), die eine Stärke (`LineWidth`) von 5 Millimetern aufweist und zu 50 Prozent transparent ist. Das rechte und linke Ende der Linie sind bis zu ihren Schnittpunkten mit der jeweils anderen Linie durchgeführt (`LineJoint` = `MITER`), so dass sich eine rechtwinklige Ecke ergibt.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.LineColor = RGB(128,128,128)
RectangleShape.LineTransparence = 50
RectangleShape.LineWidth = 500
RectangleShape.LineJoint = com.sun.star.drawing.LineJoint.MITER

RectangleShape.LineStyle = com.sun.star.drawing.LineStyle.SOLID

Page.add(RectangleShape)
```

Neben den aufgeführten Eigenschaften bietet der Service `com.sun.star.drawing.LineStyle` Möglichkeiten zur Darstellung gepunkteter und gestrichelter Linien. Details dazu finden Sie in der StarOffice API-Referenz.

Textattribute (von Zeichenobjekten)

Für die Formatierung des Textes in Zeichenobjekten stehen die Services `com.sun.star.style.CharacterProperties` und `com.sun.star.style.ParagraphProperties` zur Verfügung, die sich auf einzelne Zeichen sowie Absätze beziehen und bereits in Kapitel 6 (*Textdokumente*) ausführlich beschrieben wurden.

Das nachfolgende Beispiel zeigt, wie sich ein Text in ein Rechteck einfügen und die Schriftart über den Service `com.sun.star.style.CharacterProperties` ändern lässt.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000
Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

Page.add(RectangleShape)

RectangleShape.String = "Das ist ein Test"
RectangleShape.CharWeight = com.sun.star.awt.FontWeight.BOLD
RectangleShape.CharFontName = "Arial"
```

Zunächst wird der eigentliche Text über die `String`-Eigenschaft des Rechtecks eingefügt. Daraufhin kommen die Eigenschaften `CharWeight` und `CharFontName` aus dem Service `com.sun.star.style.CharacterProperties` für die Anpassung der Schriftart zum Einsatz.

Der Text darf erst nach dem Einfügen in das Zeichenobjekt in die zugehörige Zeichenseite eingefügt werden. Weitere Besonderheiten existieren bei der Positionierung des Textes innerhalb des Absatzes. Hier sollte auf den Einsatz der entsprechenden Eigenschaften des Services `com.sun.star.style.ParagraphProperties` (insbesondere der Eigenschaft `ParaAdjust`) verzichtet werden, da diese zu unerwarteten Ergebnissen führen. Zuständig für derartige Aufgaben ist viel mehr der Service `com.sun.star.drawing.Text`. Er sorgt für das richtige Zusammenspiel zwischen Zeichenobjekt und Text.

Hier eine Übersicht zu den wichtigsten Eigenschaften von `com.sun.star.drawing.Text`:

- **`TextAutoGrowHeight (Boolean)`** - passt die Höhe des Zeichenelements an den enthaltenen Text an.
- **`TextAutoGrowWidth (Boolean)`** - passt die Breite des Zeichenelements an den enthaltenen Text an.

- **`TextHorizontalAdjust (Enum)`** - horizontale Position des Textes innerhalb des Zeichenelementes (Vorgabewerte gemäß `com.sun.star.drawing.TextHorizontalAdjust`).
- **`TextVerticalAdjust (Enum)`** - vertikale Position des Textes innerhalb des Zeichenelmentes (Vorgabewerte gemäß `com.sun.star.drawing.TextVerticalAdjust`).
- **`TextLeftDistance (Long)`** - linker Abstand zwischen Zeichenelement und Text in 100stel Millimeter.
- **`TextRightDistance (Long)`** - rechter Abstand zwischen Zeichenelement und Text in 100stel Millimeter.
- **`TextUpperDistance (Long)`** - oberer Abstand zwischen Zeichenelement und Text in 100stel Millimeter.
- **`TextLowerDistance (Long)`** - unterer Abstand zwischen Zeichenelement und Text in 100stel Millimeter.

Das folgende Beispiel demonstriert den Einsatz der genannten Eigenschaften.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

Page.add(RectangleShape)

RectangleShape.String = "Das ist ein Test"' Darf erst nach Page.add erfolgen!

RectangleShape.TextVerticalAdjust = com.sun.star.drawing.TextVerticalAdjust.TOP
RectangleShape.TextHorizontalAdjust = com.sun.star.drawing.TextHorizontalAdjust.LEFT

RectangleShape.TextLeftDistance = 300
RectangleShape.TextRightDistance = 300
RectangleShape.TextUpperDistance = 300
RectangleShape.TextLowerDistance = 300
```

Es fügt ein Zeichenelement in eine Seite ein und ergänzt es um einen Text. Der Text wird über die Eigenschaften `TextVerticalAdjust` und `TextHorizontalAdjust` in der linken oberen Ecke des Zeichenelementes platziert und der minimale Abstand zu allen vier Seiten wird auf 3 Millimeter gesetzt.

Schattenattribute

Eine ganze Reihe von Zeichenelementen in StarOffice können einen Schatten besitzen. Dieser erscheint etwas versetzt, zum eigentlichen Zeichenobjekt und gibt die Konturen des Objektes wieder.

Zuständig für die Darstellung des Schattens ist der Service `com.sun.star.drawing.ShadowProperties`. Hier die Eigenschaften des Service:

- **`Shadow (Boolean)`** - aktiviert den Schatten.
- **`ShadowColor (Long)`** - Schattenfarbe.
- **`ShadowTransparence (Short)`** - Transparenz des Schattens.
- **`ShadowXDistance (Long)`** - vertikale Verschiebung des Schattens gegenüber dem Basisobjekt in 100stel Millimeter.
- **`ShadowYDistance (Long)`** - horizontale Verschiebung des Schattens gegenüber dem Basisobjekt in 100stel Millimeter.

Das folgende Beispiel erzeugt ein Rechteck mit einem Schatten, der vertikal und horizontal um 2 Millimeter verschoben ist. Der Schatten erscheint Hellgrau und ist zu 50 Prozent transparent.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.Shadow = True
RectangleShape.ShadowColor = RGB(192,192,192)
RectangleShape.ShadowTransparence = 50
RectangleShape.ShadowXDistance = 200
RectangleShape.ShadowYDistance = 200

Page.add(RectangleShape)
```

Die verschiedenen Zeichenobjekte im Überblick

Rechtecke

Rechteck-Objekte (`com.sun.star.drawing.RectangleShape`) unterstützen alle Services zur Formatierung von Objekten. Dies sind:

- **Flächenattribute** – `com.sun.star.drawing.FillProperties`
- **Linienattribute** – `com.sun.star.drawing.LineProperties`
- **Textattribute** – `com.sun.star.drawing.Text` (mit `com.sun.star.style.CharacterProperties` und `com.sun.star.style.ParagraphProperties`)
- **Schattenattribute** – `com.sun.star.drawing.ShadowProperties`

- **CornerRadius (Long) –** Radius zur Abrundung der Ecken in 100stel Millimeter.

Kreise und Ellipsen

Für Kreise und Ellipsen ist der Service `com.sun.star.drawing.EllipseShape` zuständig. Kreise und Ellipsen unterstützen alle im vorigen Abschnitt beschriebenen Formatierungs-Services:

- **Flächenattribute –** `com.sun.star.drawing.FillProperties`
- **Linienattribute –** `com.sun.star.drawing.LineProperties`
- **Textattribute –** `com.sun.star.drawing.Text` (mit `com.sun.star.style.CharacterProperties` und `com.sun.star.style.ParagraphProperties`)
- **Schattenattribute –** `com.sun.star.drawing.ShadowProperties`

Darüber hinaus halten Kreise und Ellipsen drei weitere Eigenschaften bereit:

- **`CircleKind (Enum)`** - Art des Kreises bzw. der Ellipse (Vorgabewerte gemäß `com.sun.star.drawing.CircleKind`).
- **`CircleStartAngle (Long)`** - Startwinkel in 10tel Grad (nur bei Kreis- bzw. Ellipsenausschnitten).
- **`CircleEndAngle (Long)`** - Endwinkel in 10tel Grad (nur bei Kreis- bzw. Ellipsenausschnitten).

Für weitere Vielfalt sorgt die Eigenschaft `CircleKind`. Sie bestimmt, ob es sich um eine Vollellipse, einen Ellipsenausschnitt oder einen Ellipsenabschnitt handelt. Folgende Werte sind möglich:

- **`com.sun.star.drawing.CircleKind.FULL`** – Vollkreis beziehungsweise volle Ellipse.
- **`com.sun.star.drawing.CircleKind.CUT`** – Kreisabschnitt (Teilkreis, dessen Schnittstellen direkt miteinander verbunden sind).
- **`com.sun.star.drawing.CircleKind.SECTION`** – Kreisausschnitt ("Tortenstück")
- **`com.sun.star.drawing.CircleKind.ARC`** – Winkel (ohne Kreislinie).

Das folgende Beispiel erzeugt einen Kreisausschnitt mit einem Winkel von 70 Grad (ergibt sich durch die Differenz zwischen dem Startwinkel von 20 Grad und dem Endwinkel von 90 Grad).

```
Dim Doc As Object
Dim Page As Object
Dim EllipseShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)
```

```
EllipseShape = Doc.createInstance("com.sun.star.drawing.EllipseShape")
EllipseShape.Size = Size
EllipseShape.Position = Point

EllipseShape.CircleStartAngle = 2000
EllipseShape.CircleEndAngle = 9000
EllipseShape.CircleKind =  com.sun.star.drawing.CircleKind.SECTION

Page.add(EllipseShape)
```

Linien

Für Linienobjekte hält StarOffice den Service `com.sun.star.drawing.LineShape` bereit. Linienobjekte unterstützen alle generellen Formatierungsdienste, mit Ausnahme des Dienstes für Flächen:

- **Linienattribute** – `com.sun.star.drawing.LineProperties`
- **Textattribute** – `com.sun.star.drawing.Text` (mit `com.sun.star.style.CharacterProperties` und `com.sun.star.style.ParagraphProperties`)
- **Schattenattribute** – `com.sun.star.drawing.ShadowProperties`

Darüber hinausgehende Eigenschaften stellt der Service `LineShape` nicht zur Verfügung.

Das folgende Beispiel zeigt, wie sich mit Hilfe der genannten Eigenschaften eine Linie erzeugen und formatieren lässt. Der als `Location` angegebene Punkt bestimmt den Ursprung der Linie. Die in `Size` aufgeführten Koordinaten den Endpunkt.

```
Dim Doc As Object
Dim Page As Object
Dim LineShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

LineShape = Doc.createInstance("com.sun.star.drawing.LineShape")
LineShape.Size = Size
LineShape.Position = Point

Page.add(LineShape)
```

Vielecke

Neben den geometrischen Basisformen wie Kreisen und Rechtecken unterstützt StarOffice auch komplexe Vielecke. Zuständig für die Darstellung dieser Objekte ist der Service `com.sun.star.drawing.PolyPolygonShape`. Ein solches Objekt kann beliebig viele Eckpunkte haben. Streng genommen handelt es sich bei einem *PolyPolygon* nicht einmal um ein einfaches Vieleck sondern ein mehrfaches. So lassen sich mehrere unabhängige Listen mit Punkten angeben, die zu einem Gesamtobjekt zusammengefasst werden.

Analog zu den Rechtecken stehen auch für Vielecke alle Formatierungsattribute von Zeichenobjekten zur Verfügung:

- **Flächenattribute** – `com.sun.star.drawing.FillProperties`
- **Linienattribute** – `com.sun.star.drawing.LineProperties`
- **Textattribute** – `com.sun.star.drawing.Text` (mit `com.sun.star.style.CharacterProperties` und `com.sun.star.style.ParagraphProperties`)
- **Schattenattribute** – `com.sun.star.drawing.ShadowProperties`

Darüber hinaus besitzt der Service `PolyPolygonShape` eine zusätzliche Eigenschaft, über die sich die Koordinaten des Objektes definieren lassen:

- `PolyPolygon (Array)` – Feld mit den Koordinaten des Vielecks (Doppel-Array mit Punkten vom Typ `com.sun.star.awt.Point`)

Das folgende Beispiel zeigt, wie sich mit Hilfe eines *PolyPolygons* ein Dreieck definieren lässt.

```
Dim Doc As Object
Dim Page As Object
Dim PolyPolygonShape As Object
Dim PolyPolygon As Variant
Dim Coordinates(2) As New com.sun.star.awt.Point

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

PolyPolygonShape = Doc.createInstance("com.sun.star.drawing.PolyPolygonShape")
Page.add(PolyPolygonShape)    ' Page.add muss vor dem Setzen der Koordinaten erfolgen

Coordinates(0).x = 1000
Coordinates(1).x = 7500
Coordinates(2).x = 10000
Coordinates(0).y = 1000
Coordinates(1).y = 7500
Coordinates(2).y = 5000

PolyPolygonShape.PolyPolygon = Array(Coordinates())
```

Die Angabe einer Größe und einer Startposition sind bei der Arbeit mit PolyPolygonen überflüssig, da die eigentlichen Punkte des PolyPolygons in Absolutwerten aufgeführt werden. Für die Punkte ist ein Array aufzubauen, das in ein zweites Array verpackt (über den Aufruf `Array (Coordinates())` und anschließend dem `PolyPolygon` zugewiesen wird. Vor den betreffenden Aufruf muss das `PolyPolygon` allerdings bereits in die Zeichenebene eingefügt worden sein. Erst danach lässt sich die Eigenschaft `PolyPolygon` setzen.

Das Doppel-Array bei der Definition ist wichtig, da einem PolyPolygon mehrere unabhängige Polygone zugeordnet werden können. So ist es beispielsweise möglich, Flächen mit Löchern aufzubauen. Das folgende Beispiel enthält zwei ineinander geschachtelte Rechtecke. Das äußere Rechteck dient dabei als äußere Begrenzungslinie, das innere Rechteck als Innere Begrenzungslinie. Die Figur hat damit quasi ein Loch.

```
Dim Doc As Object
Dim Page As Object
Dim PolyPolygonShape As Object
```

```
Dim PolyPolygon As Variant
Dim Square1(3) As New com.sun.star.awt.Point
Dim Square2(3) As New com.sun.star.awt.Point
Dim Square3(3) As New com.sun.star.awt.Point

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

PolyPolygonShape = Doc.createInstance("com.sun.star.drawing.PolyPolygonShape")

Page.add(PolyPolygonShape)    ' Page.add muss vor dem Setzen der Koordinaten erfolgen

Square1(0).x = 5000
Square1(1).x = 10000
Square1(2).x = 10000
Square1(3).x = 5000
Square1(0).y = 5000
Square1(1).y = 5000
Square1(2).y = 10000
Square1(3).y = 10000

Square2(0).x = 6500
Square2(1).x = 8500
Square2(2).x = 8500
Square2(3).x = 6500
Square2(0).y = 6500
Square2(1).y = 6500
Square2(2).y = 8500
Square2(3).y = 8500

Square3(0).x = 6500
Square3(1).x = 8500
Square3(2).x = 8500
Square3(3).x = 6500
Square3(0).y = 9000
Square3(1).y = 9000
Square3(2).y = 9500
Square3(3).y = 9500

PolyPolygonShape.PolyPolygon = Array(Square1(), Square2(), Square3())
```

Durch das Einfügen mehrerer Polygone können sich beliebig komplexe Figuren ergeben. Auf den ersten Blick mag es bei solchen Figuren unklar sein, welche der Teilbereiche als Fläche und welche als Löcher dargestellt werden. StarOffice wendet dafür jedoch eine einfache Regel an: Die äußerste Linie gilt stets als äußere Begrenzungslinie des PolyPolygons. Die nächst innen liegende Linie stellt eine innere Begrenzungslinie dar und markiert somit einen Übergang zum ersten Loch. Folgt daraufhin eine weitere Linie, so markiert diese wiederum den Übergang zu einer Fläche.

Grafiken

Das letzte der hier vorgestellten Zeichenelemente sind Grafik-Objekte auf der Grundlage des Services `com.sun.star.drawing.GraphicObjectShape`. Sie stellen eine beliebige Grafik innerhalb von StarOffice dar, deren Aussehen über eine ganze Reihe von Eigenschaften angepasst werden kann.

Grafikobjekte unterstützen zwei der vier Services mit generellen Formatierungsattributen:

- **Textattribute** – `com.sun.star.drawing.Text` (mit `com.sun.star.style.CharacterProperties` und `com.sun.star.style.ParagraphProperties`)
- **Schattenattribute** – `com.sun.star.drawing.ShadowProperties`

Darüber hinaus bieten sie eine Reihe von Eigenschaften zur Anpassung der dargestellten Bilder:

- **`GraphicURL (String)`** - URL der Grafik.
- **`AdjustLuminance (Short)`** - Leuchtkraft der Farben in Prozent (auch negative Werte zugelassen).
- **`AdjustContrast (Short)`** - Kontrast in Prozent (auch negative Werte zugelassen).
- **`AdjustRed (Short)`** - Rotanteil in Prozent (auch negative Werte zugelassen).
- **`AdjustGreen (Short)`** - Grünanteil in Prozent (auch negative Werte zugelassen).
- **`AdjustBlue (Short)`** - Blauanteil in Prozent (auch negative Werte zugelassen).
- **`Gamma (Short)`** - Gamma-Wert der Grafik.
- **`Transparency (Short)`** - Transparenz der Grafik in Prozent.
- **`GraphicColorMode (enum)`** - Farbmodus, z.B. Standard, Graustufen, Schwarzweiß (Vorgabewert gemäß `com.sun.star.drawing.ColorMode`).

Das folgende Beispiel zeigt, wie sich ein Grafikobjekt in eine Seite einfügen lässt.

```
Dim Doc As Object
Dim Page As Object
Dim GraphicObjectShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000                    ' Angaben, unerheblich, da spätere Koordinaten
verbindlich
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

GraphicObjectShape = Doc.createInstance("com.sun.star.drawing.GraphicObjectShape")

GraphicObjectShape.Size = Size
GraphicObjectShape.Position = Point

GraphicObjectShape.GraphicURL = "file:///c:/test.jpg"
GraphicObjectShape.AdjustBlue = -50
GraphicObjectShape.AdjustGreen = 5
GraphicObjectShape.AdjustBlue = 10
GraphicObjectShape.AdjustContrast = 20
GraphicObjectShape.AdjustLuminance = 50
GraphicObjectShape.Transparency = 40
GraphicObjectShape.GraphicColorMode = com.sun.star.drawing.ColorMode.STANDARD

Page.add(GraphicObjectShape)
```

Das Beispiel bindet die Grafik `test.jpg` ein und passt das Aussehen über die `Adjust`-Eigenschaften an. Die Grafik wird dabei zu 40 Prozent transparent dargestellt. Eine zusätzliche Farbumwandlung findet nicht statt (`GraphicColorMode` = `STANDARD`).

Zeichenobjekte bearbeiten

Objekte gruppieren

In vielen Situationen ist es hilfreich, mehrere einzelne Zeichenobjekte zu einem großen Objekt zusammen zu fassen. So kann der Anwender die einzelnen Elemente zusammen verschieben, vergrößern und verkleinern.

Das nachfolgende Beispiel zeigt, wie Sie zwei Zeichenobjekte miteinander verbinden können.

```
Dim Doc As Object
Dim Page As Object
Dim Square As Object
Dim Circle As Object
Dim Shapes As Object
Dim Group As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size
Dim NewPos As New com.sun.star.awt.Point
Dim Height As Long
Dim Width As Long

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)
Point.x = 3000
Point.y = 3000
Size.Width = 3000
Size.Height = 3000
' Zeichenelement Square erzeugen
Square = Doc.createInstance("com.sun.star.drawing.RectangleShape")
Square.Size = Size
Square.Position = Point
Square.FillColor = RGB(255,128,128)
Page.add(Square)
' Zeichenelement Circle erzeugen
Circle = Doc.createInstance("com.sun.star.drawing.EllipseShape")
Circle.Size = Size
Circle.Position = Point
Circle.FillColor = RGB(255,128,128)
Circle.FillColor = RGB(0,255,0)
Page.add(Circle)
' Zeichenelemente Square und Circle zusammen fassen
Shapes = createUnoService("com.sun.star.drawing.ShapeCollection")
Shapes.add(Square)
Shapes.add(Circle)
Group = Page.group(Shapes)
' Zusammengefasste Zeichenelemente zentrieren
Height = Page.Height
Width = Page.Width
NewPos.X = Width / 2
NewPos.Y = Height / 2
Height = Group.Size.Height
Width = Group.Size.Width
NewPos.X = NewPos.X - Width / 2
NewPos.Y = NewPos.Y - Height / 2
Group.Position = NewPos
```

Der Beispielcode erzeugt zunächst ein Rechteck und einen Kreis und fügt diese Elemente unabhängig voneinander in die Zeichenseite ein. Anschließend erzeugt er ein Objekt, das den Service `com.sun.star.drawing.ShapeCollection` unterstützt. Diesem Objekt werden nun die beiden zu gruppierenden Zeichenobjekte über die Methode `Add` hinzu gefügt. Die so erstellte

`ShapeCollection` lässt sich der Zeichenseite über die Methode `Group` hinzu fügen. Sie liefert das eigentliche `Group`-Objekt zurück, das sich dann wie ein einzelnes `Shape` bearbeiten lässt. Um dies zu demonstrieren, wird das Gruppenobjekt zentriert in der Mitte der Seite positioniert.

Bei der Arbeit mit gruppierten Zeichenelementen ist darauf zu achten, dass diese vor dem Einfügen in die Gruppe bereits die gewünschte Formatierung aufweisen, da die Formatierungsattribute nach dem Hinzufügen des Objektes in die Gruppe nicht mehr verändert werden können.

Rotieren und Scheren von Zeichenobjekten

Alle im vorigen Abschnitt aufgeführten Zeichenobjekte lassen sich drehen und scheren. Dazu unterstützen sie den Service `com.sun.star.drawing.RotationDescriptor`.

Der Service stellt folgende Eigenschaften zur Verfügung:

- **`RotateAngle (Long)`** – Drehwinkel in 100stel Grad.
- **`ShearAngle (Long)`** – Scherwinkel in 100stel Grad.

Das folgende Beispiel erzeugt ein Rechteck und dreht es über `RotateAngle` um 30 Grad.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.RotateAngle = 3000

Page.add(RectangleShape)
```

Nachfolgend ein Beispiel, das das gleiche Rechteck wie im vorstehenden Fall erzeugt, dieses jedoch nicht dreht sondern um 30 Grad schert. Hierzu verwendet es die Eigenschaft `ShearAngle`.

```
Dim Doc As Object
Dim Page As Object
Dim RectangleShape As Object
Dim Point As New com.sun.star.awt.Point
Dim Size As New com.sun.star.awt.Size

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

RectangleShape = Doc.createInstance("com.sun.star.drawing.RectangleShape")
RectangleShape.Size = Size
RectangleShape.Position = Point

RectangleShape.ShearAngle = 3000

Page.add(RectangleShape)
```

Suchen und Ersetzen

Zeichendokumente bieten ähnlich wie Textdokumente eine Funktion zum Suchen und Ersetzen. Das Vorgehen orientiert sich dabei an dem in Textdokumenten eingesetzten Verfahren (siehe Kapitel 6, *Textdokumente*).

Die Deskriptor-Objekte zum Suchen und Ersetzen werden in Zeichendokumenten jedoch nicht direkt über das Dokumentobjekt, sondern über die zugehörige Zeichenebene erzeugt. Das folgende Beispiel verdeutlicht einen Ersetzungsvorgang innerhalb einer Zeichnung:

```
Dim Doc As Object
Dim Page As Object
Dim ReplaceDescriptor As Object
Dim I As Integer

Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)

ReplaceDescriptor = Page.createReplaceDescriptor()
ReplaceDescriptor.SearchString = "is"
ReplaceDescriptor.ReplaceString = "was"

For I = 0 to Doc.drawPages.Count - 1
      Page = Doc.drawPages(I)
      Page.ReplaceAll(ReplaceDescriptor)
Next I
```

Das Beispiel erzeugt über die erste `DrawPage` des Dokumentes einen `ReplaceDescriptor` und wendet diesen anschließend in einer Schleife auf alle Seiten der Zeichnung an.

Präsentationen

StarOffice-Präsentationen basieren auf Zeichendokumenten. Jede Seite der Zeichnung stellt eine eigene Präsentationsfolie dar. Der Zugriff auf die Seiten erfolgt wie bei einer gewöhnlichen Zeichnung über die Liste `DrawPages` des Dokument-Objektes. Der für Präsentationsdokumente zuständige Service `com.sun.star.presentation.PresentationDocument` stellt dazu den kompletten Service `com.sun.star.drawing.DrawingDocument` zur Verfügung.

Arbeiten mit Präsentationen

Über die Zeichenfunktionen hinaus liefert das Präsentationsdokument über die (nachgebildete) Eigenschaft `Presentation` ein Präsentationsobjekt zurück, dass Zugriff auf die zentralen Eigenschaften und Steuerungsmechanismen von Präsentationen bietet. Es hält beispielsweise eine Methode `start` bereit, die zum Starten von Präsentationen dient.

```
Dim Doc As Object
Dim Presentation As Object

Doc = StarDesktop.CurrentComponent
Presentation = Doc.Presentation
Presentation.start()
```

Der Beispielcode erzeugt ein Objekt `Doc`, das auf das aktuelle Präsentationsdokument verweist, und ermittelt das dazugehörige Präsentationsobjekt. Über die Methode `start()` des Objektes startet das Beispiel anschließend die Bildschirmpräsentation.

Das Präsentationsobjekt hält folgende Methoden bereit:

- **`start`** - startet die Präsentation.
- **`end`** - beendet die Präsentation.
- **`rehearseTimings`** - startet die Präsentation vom Beginn und ermittelt dabei deren Laufzeit.

Darüber hinaus stehen folgende Eigenschaften zur Verfügung:

- **`AllowAnimations (Boolean)`** – führt Animationen in der Präsentation aus.
- **`CustomShow (String)`** – gestattet die Angabe eines individuellen Namens für die Vorführung, auf den in der Präsentation Bezug genommen werden kann.
- **`FirstPage (String)`** – Name der Folie, mit der die Präsentation starten soll.
- **`IsAlwaysOnTop (Boolean)`** – zeigt das Präsentationsfenster immer als oberstes Fenster auf dem Bildschirm an.
- **`IsAutomatic (Boolean)`** – durchläuft die Präsentation automatisch.
- **`IsEndless (Boolean)`** – beginnt die Präsentation nach dem Ende wieder von vorne.
- **`IsFullScreen (Boolean)`** – startet die Präsentation automatisch im Vollbildmodus.
- **`IsMouseVisible (Boolean)`** – zeigt die Maus während der Präsentation an.
- **`Pause (long)`** – Zeit, für die das Präsentationsmodul nach dem Ablauf der Präsentation einen schwarzen Bildschirm anzeigt.
- **`StartWithNavigator (Boolean)`** – zeigt das Navigatorfenster beim Start der Präsentation an.

- **`UsePen (Boolean)`** – blendet einen Zeichenstift während der Präsentation ein.

Diagramme (Charts)

StarOffice kann Daten als Diagramm darstellen und so in Form von Balken, Tortenstücken, Linien oder anderen Elementen grafisch zueinander in Bezug setzen. Wahlweise erfolgt die Ausgabe als 2D- oder 3D-Grafik, wobei das Aussehen der Diagramm-Elemente ähnlich wie das von Zeichenelementen individuell angepasst werden kann.

Liegen die darzustellenden Daten als Spreadsheet vor, so lassen sie sich dynamisch mit dem Diagramm verknüpfen. Änderungen der Basisdaten werden in diesem Fall sofort im zugeordneten Diagramm sichtbar. Dieses Kapitel liefert einen Überblick zur Programmierschnittstelle des Diagramm-Moduls von StarOffice, wobei der Schwerpunkt auf dem Einsatz von Diagrammen innerhalb von Spreadsheet-Dokumenten liegt.

Diagramme in Spreadsheets nutzen

Diagramme werden in StarOffice nicht als eigenständige Dokumente behandelt, sondern als Objekte, die in ein existierendes Dokument eingebettet werden.

Während Diagramme in Text- und Zeichendokumenten isoliert vom Inhalt des Dokumentes stehen, können sie in Spreadsheet-Dokumenten mit den Dokumentdaten verknüpft werden. Das folgende Beispiel erläutert das Zusammenspiel zwischen Spreadsheet-Dokument und Diagramm.

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object

Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Charts = Doc.Sheets(0).Charts

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000

RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
```

Obwohl der Beispielcode recht komplex erscheint, beschränken sich die zentralen Vorgänge auf drei Zeilen: Erste zentrale Zeile ist die Erzeugung der Dokumentvariable `Doc`, die auf das aktuelle

Spreadsheet-Dokument verweist (Zeile `Doc = StarDesktop.CurrentComponent`). Im Anschluss daran ermittelt der Beispielcode eine Liste mit allen Charts des ersten Spreadsheets (Zeile `Charts = Doc.Sheets(0).Charts`). Dieser Liste wird schließlich in der letzten Zeile des Beispiels über die Methode `addNewByName` ein neues Chart hinzu gefügt, das dann für den Anwender sichtbar wird.

Die letzte Zeile dient lediglich dazu, die Hilfsstrukturen `Rect` und `RangeAddress` zu initialisieren, die der Methode `addNewByName` als Parameter mit gegeben werden. `Rect` bestimmt die Position des Charts innerhalb des Spreadsheets. `RangeAddress` den Bereich, dessen Daten mit dem Chart verknüpft werden sollen.

Das obige Beispiel erzeugt eine Balkengrafik. Wird eine andere Grafik benötigt, so muss das Balkendiagramm explizit ausgetauscht werden:

```
Chart = Charts.getByName("MyChart").embeddedObject
Chart.Diagram = Chart.createInstance("com.sun.star.chart.LineDiagram")
```

Die erste der genannten Zeilen dient dazu, das betreffende Chart-Objekt zu ermitteln. Die zweite Zeile tauscht das aktuelle Diagramm durch ein neues aus – im Beispiel ein Liniendiagramm.

In Excel wird unterschieden zwischen Charts, die als eigenständige Seite in ein Excel-Dokument eingefügt wurden und Charts, die in eine Tabellenseite eingebettet sind. Dementsprechend sind dort zwei verschiedene Zugriffsmöglichkeiten für Charts definiert. Diese Unterscheidung entfällt in StarOffice Basic, da Charts in StarOffice Calc prinzipiell als eingebettete Objekte einer Tabellenseite erzeugt werden. Der Zugriff auf die Charts erfolgt hier immer über die Liste `Charts` des zugehörigen `Sheet`-Objektes.

Der Aufbau von Diagrammen

Der Aufbau eines Diagramms – und damit die Liste der von ihm unterstützten Services und Schnittstellen – hängt von seinem Typ ab. So stehen die Methoden und Eigenschaften der Z-Achse beispielsweise nur in 3D-Diagrammen zur Verfügung, nicht jedoch in 2D-Diagrammen. In Tortendiagrammen beispielsweise fehlen jegliche Schnittstellen für den Umgang mit Achsen, da sie sinnlos wären.

Die Einzelelemente eines Diagramms

Titel, Untertitel und Legende

Zu den Basiselementen eines jeden Diagramms gehört ein Titel, ein Untertitel und eine Legende. Diagramme stellen für jedes dieser Elemente eigene Objekte zur Verfügung. Das `Chart`-Objekt bietet zu deren Verwaltung folgende Eigenschaften an:

- **`HasMainTitle (Boolean)`** – aktiviert den Titel.
- **`Title (Object)`** – Objekt mit Detail-Angaben zum Diagramm-Titel (unterstützt den Service `com.sun.star.chart.ChartTitle`).
- **`HasSubTitle(Boolean)`** – aktiviert den Untertitel.

- **`Subtitle (Object)`** – Objekt mit Detail-Angaben zum Untertitel des Diagramms (unterstützt den `Service com.sun.star.chart.ChartTitle`).
- **`HasLegend (Boolean)`** – aktiviert die Legende.
- **`Legend (Object)`** – Objekt mit Detail-Angaben zur Legende des Diagramms (unterstützt den Service `com.sun.star.chart.ChartLegendPosition`).

Die genannten Elemente entsprechen in vieler Hinsicht einem Zeichenelement. Dies liegt daran, dass sowohl der Service `com.sun.star.chart.ChartTitle` als auch `com.sun.star.chart.ChartLegendPosition` den Service `com.sun.star.drawing.Shape` unterstützen, der die programmtechnische Grundlage für Zeichenelemente legt.

So besteht die Möglichkeit, die Position und Größe der Elemente über die Eigenschaften `Size` und `Position` zu bestimmen.

Für die Formatierung der Elemente stehen des weiteren Flächen- und Linienattribute (Services `com.sun.star.drawing.FillProperties` und `com.sun.star.drawing.LineStyle`) sowie Zeichenattribute (Service `com.sun.star.style.CharacterProperties`) zur Verfügung.

`com.sun.star.chart.ChartTitle` besitzt neben den genannten Format-Attributen zwei weitere Eigenschaften:

- **`TextRotation (Long)`** – Drehwinkel des Textes in 100stel Grad.
- **`String (String)`** – Text, der als Titel beziehungsweise Untertitel dargestellt werden soll.

Die Diagramm-Legende (Service `com.sun.star.chart.ChartLegend`) besitzt folgende zusätzliche Eigenschaft:

- **`Alignment (Enum)`** – Position, an der die Legende erscheinen soll (Vorgabewert gemäß `com.sun.star.chart.ChartLegendPosition`).

Das nachfolgende Bespiel erzeugt ein Diagramm und weist ihm den Titel "Test", den Untertitel "Test 2" und eine Legende zu. Die Legende erhält eine hellgraue Hintergrundfarbe, wird unten innerhalb des Diagramms platziert und erhält eine Schriftgröße von 7 Punkt.

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object

Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Rect.X = 8000
Rect.Y = 1000
```

```
Rect.Width = 10000
Rect.Height = 7000

RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Doc = StarDesktop.CurrentComponent
Charts = Doc.Sheets(0).Charts
Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
Chart = Charts.getByName("MyChart").EmbeddedObject

Chart.HasMainTitle = True
Chart.Title.String = "Test"

Chart.HasSubTitle = True
Chart.Subtitle.String = "Test 2"

Chart.HasLegend = True
Chart.Legend.Alignment = com.sun.star.chart.ChartLegendPosition.BOTTOM
Chart.Legend.FillStyle = com.sun.star.drawing.FillStyle.SOLID
Chart.Legend.FillColor = RGB(210, 210, 210)
Chart.Legend.CharHeight = 7
```

Hintergrund

Jedes Diagramm besitzt eine Hintergrundfläche. Für jede der Flächen existiert ein Objekt, das über folgende Eigenschaft des Diagramm-Objektes erreichbar ist:

- **`Area (Object)`** – Hintergrundfläche des Diagramms (unterstützt Service `com.sun.star.chart.ChartArea`).

Der Hintergrund eines Diagramms umfasst seine komplette Fläche, also auch den Bereich hinter Titel, Untertitel und Diagramm-Legende. Der zugehörige Service `com.sun.star.chart.ChartArea` unterstützt Linien- und Flächenattribute und stellt keine weitergehenden Eigenschaften zur Verfügung.

Diagrammwände und Bodenfläche

Während der Diagrammhintergrund die komplette Fläche des Diagramms umfasst, beschränkt sich die Diagrammrückwand auf die Fläche unmittelbar hinter dem Datenbereich.

Bei 3D-Diagrammen existieren für gewöhnlich zwei Diagrammwände: eine hinter dem Datenbereich und eine als linke Begrenzung zur Y-Achse. Zusätzlich enthalten 3D-Diagramme in der Regel eine Bodenfläche.

- **`Floor (Object)`** – Bodenplatte des Diagramms (nur bei 3D-Diagrammen, unterstützt Service `com.sun.star.chart.ChartArea`).
- **`Wall (Object)`** – Diagrammwände (nur bei 3D-Diagrammen, unterstützt Service `com.sun.star.chart.ChartArea`).

Die genannten Objekte unterstützen den Service `com.sun.star.chart.ChartArea`, der wiederum die üblichen Flächen- und Linienattribute bereit hält (Services `com.sun.star.drawing.FillProperties` und `com.sun.star.drawing.LineStyle`, siehe Kapitel 8).

Der Zugriff auf Diagrammwände und Bodenfläche erfolgt über das `Chart`-Objekt:

```
Chart.Area.FillBitmapName = "Sky"
```

Das folgende Beispiel zeigt, wie sich eine bereits in StarOffice eingebundene Grafik mit dem Namen Sky als ein Hintergrund eines Diagrammes einsetzen lässt.

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object

Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000

RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Doc = StarDesktop.CurrentComponent
Charts = Doc.Sheets(0).Charts

Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)
Chart = Charts.getByName("MyChart").EmbeddedObject

Chart.Area.FillStyle = com.sun.star.drawing.FillStyle.BITMAP
Chart.Area.FillBitmapName = "Sky"
Chart.Area.FillBitmapMode = com.sun.star.drawing.BitmapMode.REPEAT
```

Achsen

StarOffice kennt fünf verschiedene Achsen, die in einem Diagramm zum Einsatz kommen können. Im einfachsten Falle sind dies die X- und Y-Achse. Bei 3D-Diagrammen steht teilweise

zusätzlich eine Z-Achse zur Verfügung. Für Diagramme, bei denen die Werte der verschiedenen Datenreihen stark voneinander abweichen, bietet StarOffice eine zweite X- und Y-Achse für eine zweite Skalierung an.

Erste X-, Y- und Z-Achse

Für jede der ersten X-, Y- und Z-Achsen können neben der eigentlichen Achse ein Titel, eine Beschreibung, ein Gitter und ein Hilfsgitter existieren. Alle diese Elemente können wahlweise ein- und ausgeblendet werden. Das Diagramm-Objekt bietet zu deren Verwaltung folgende Eigenschaften an (Beispiel X-Achse; Eigenschaften für Y- und Z-Achse sind analog aufgebaut):

- **`HasXAxis (Boolean)`** – aktiviert die X-Achse.
- **`XAxis (Object)`** – Objekt mit Detail-Informationen zur X-Achse (unterstützt Service `com.sun.star.chart.ChartAxis`).
- **`HasXAxisDescription (Boolean)`** – aktiviert die Beschreibung der X-Achse.
- **`HasXAxisGrid (Boolean)`** – aktiviert das Hauptgitter für die X-Achse.
- **`XMainGrid (Object)`** – Objekt mit Detail-Informationen zum Hauptgitter für die X-Achse (unterstützt Service `com.sun.star.chart.ChartGrid`).
- **`HasXAxisHelpGrid (Boolean)`** – aktiviert Hilfsgitter für die X-Achse.
- **`XHelpGrid (Object)`** – Objekt mit Detail-Informationen zum Hilfsgitter für die X-Achse (unterstützt `Service com.sun.star.chart.ChartGrid`).
- **`HasXAxisTitle (Boolean)`** – aktiviert den Titel der X-Achse.
- **`XAxisTitle (Object)`** – Objekt mit Detail-Informationen für den Titel der X-Achse (unterstützt Service `com.sun.star.chart.ChartTitle`).

Zweite X- und Y-Achse

Für die zweite X- und Y-Achse stehen folgende Eigenschaften zur Verfügung (Eigenschaften am Beispiel der zweiten X-Achse):

- **`HasSecondaryXAxis (Boolean)`** – aktiviert die zweite X-Achse.
- **`SecondaryXAxis (Object)`** – Objekt mit Detail-Informationen zur zweiten X-Achse (unterstützt Service `com.sun.star.chart.ChartAxis`).
- **`HasSecondaryXAxisDescription (Boolean)`** – aktiviert die Beschreibung der X-Achse.

Eigenschaften der Achsen

Die Achsen-Objekte eines StarOffice-Diagramms unterstützen den Service `com.sun.star.chart.ChartAxis`. Er stellt neben den Attributen für Zeichen (Service `com.sun.star.style.CharacterProperties`, siehe Kapitel 6) und Linien (Services `com.sun.star.drawing.LineStyle`, siehe Kapitel 8) folgende Eigenschaften zur Verfügung:

- **`Max (Double)`** - Maximalwert für die Achse.
- **`Min (Double)`** - Minimalwert für die Achse.

- **`Origin (Double)`** - Schnittpunkt für kreuzende Achsen.
- **`StepMain (Double)`** - Abstand zwischen zwei Hauptstrichen der Achse.
- **`StepHelp (Double)`** - Abstand zwischen zwei Hilfsstrichen der Achse.
- **`AutoMax (Boolean)`** - ermittelt automatisch den Maximalwert für die Achse.
- **`AutoMin (Boolean)`** - ermittelt automatisch den Minimialwert für die Achse.
- **`AutoOrigin (Boolean)`** - ermittelt automatisch den Schnittpunkt für kreuzende Achsen.
- **`AutoStepMain (Boolean)`** - ermittelt automatisch den Abstand zwischen Hauptstrichen einer Achse.
- **`AutoStepHelp (Boolean)`** - ermittelt automatisch den Abstand zwischen Hilfsstrichen einer Achse.
- **`Logarithmic (Boolean)`** - skaliert die Achsen logarithmisch (statt linear).
- **`DisplayLabels (Boolean)`** - aktiviert die Textbeschriftung für Achsen.
- **`TextRotation (Long)`** - Drehwinkel der Textbeschriftung der Achsen in 100stel Grad.
- **`Marks (Const)`** - Konstante, die angibt, ob die Hauptstriche der Achse innerhalb und/oder außerhalb der Diagrammfläche liegen sollen (Vorgabewerte gemäß `com.sun.star.chart.ChartAxisMarks`)
- **`HelpMarks (Const)`** - Konstante, die angibt, ob die Hilfsstriche der Achse innerhalb und/oder außerhalb der Diagrammfläche liegen sollen (Vorgabewerte gemäß `com.sun.star.chart.ChartAxisMarks`)
- **`Overlap (Long)`** - Prozentwert, der angibt, wie weit sich die Balken von verschiedenen Serien überlappen dürfen (bei 100% werden die Balken komplett übereinander dargestellt, bei -100% existiert ein Abstand von einer Balkenbreite zwischen ihnen).
- **`GapWidth (long)`** - Prozentwert, der angibt, welcher Abstand zwischen den verschiedenen Balkengruppen eines Charts existieren darf (bei 100% Prozent existiert ein Abstand von einer Balkenbreite).
- **`ArrangeOrder (enum)`** - Angaben zur Positionierung der Beschriftung; neben der Positionierung auf einer Linie besteht die Möglichkeit, die Beschriftung alternierend auf zwei Linien zu verteilen (Vorgabewert laut `com.sun.star.chart.ChartAxisArrangeOrderType`)
- **`TextBreak (Boolean)`** - erlaubt Zeilenumbrüche.
- **`TextCanOverlap (Boolean)`** - erlaubt Textüberlappungen.
- **`NumberFormat (Long)`** - Nummernformat (siehe Kapitel 7, Abschnitt *Zahlen-, Datums- und Textformat*)

Eigenschaften des Achsengitters

Das Objekt für das Achsengitter basiert auf dem Service `com.sun.star.chart.ChartGrid`, der wiederum die Linienattribute des Service `com.sun.star.drawing.LineStyle` unterstützt (siehe Kapitel 8).

Eigenschaften der Achsentitel

Die Objekte zur Formatierung des Achsentitels basieren auf dem Service `com.sun.star.chart.ChartTitle`, der auch beim Diagramm-Titel zum Einsatz kommt.

Beispiel

Das folgende Beispiel erstellt ein Linien-Diagramm. Die Farbe für die Rückwand des Diagramms wird auf weiß gesetzt. Sowohl die X- als auch die Y-Achse erhalten ein hellgraues Hilfsgitter als optische Orientierung. Der Minimalwert der Y-Achse wird fest auf 0 und der Maximalwert fest auf 100 gesetzt, so dass die Auflösung des Diagramms auch bei Änderungen der Werte erhalten bleibt.

```
Dim Doc As Object
Dim Charts As Object
Dim Chart as Object

Dim Rect As New com.sun.star.awt.Rectangle
Dim RangeAddress(0) As New com.sun.star.table.CellRangeAddress

Doc = StarDesktop.CurrentComponent
Charts = Doc.Sheets(0).Charts

Rect.X = 8000
Rect.Y = 1000
Rect.Width = 10000
Rect.Height = 7000

RangeAddress(0).Sheet = 0
RangeAddress(0).StartColumn = 0
RangeAddress(0).StartRow = 0
RangeAddress(0).EndColumn = 2
RangeAddress(0).EndRow = 12

Charts.addNewByName("MyChart", Rect, RangeAddress(), True, True)

Chart = Charts.getByName("MyChart").embeddedObject
Chart.Diagram = Chart.createInstance("com.sun.star.chart.LineDiagram")

Chart.Diagram.Wall.FillColor = RGB(255, 255, 255)

Chart.Diagram.HasXAxisGrid = True
Chart.Diagram.XMainGrid.LineColor = RGB(192, 192, 192)

Chart.Diagram.HasYAxisGrid = True
Chart.Diagram.YMainGrid.LineColor = RGB(192, 192, 192)
Chart.Diagram.YAxis.Min = 0
Chart.Diagram.YAxis.Max = 100
```

3D-Diagramme

Die meisten Diagramme in StarOffice lassen sich nicht nur als Fläche darstellen, sondern auch als 3D-Grafik. Alle Diagrammtypen, die diese Möglichkeit bieten, unterstützen den Service `com.sun.star.chart.Dim3DDiagram`. Der Service stellt lediglich eine Eigenschaft zur Verfügung:

- **`Dim3D (Boolean)`** – aktiviert die 3D-Darstellung.

Gestapelte Diagramme

Unter einer gestapelten Diagrammdarstellung versteht man das Übereinanderordnen mehrerer Einzelwerte zu einem Gesamtwert. Sie vermittelt damit neben den Einzelwerten einen Überblick zur Gesamtentwicklung.

In StarOffice lassen sich verschiedene Diagrammarten in gestapelter Form darstellen. Alle diese Diagramme unterstützen den Service `com.sun.star.chart.StackableDiagram`, der wiederum folgende Eigenschaften bereit hält:

- **`Stacked (Boolean)`** – aktiviert die gestapelte Ansicht.
- **`Percent (Boolean)`** – stellt statt Absolutwerten ihre prozentuale Verteilung dar.

Die verschiedenen Diagrammtypen

Liniendiagramme

Liniendiagramme (Service `com.sun.star.chart.LineDiagram`) unterstützen eine X-Achse, zwei Y-Achsen und eine Z-Achse. Die Darstellung ist als 2D- oder 3D-Grafik möglich (Service `com.sun.star.chart.Dim3DDiagram`). Die Linien sind stapelbar (`com.sun.star.chart.StackableDiagram`).

Darüber hinaus stellen Liniendiagramme folgende Eigenschaften zur Verfügung:

- **`SymbolType (const)`** - Symbol zur Visualisierung der Datenpunkte (Konstante gemäß `com.sun.star.chart.ChartSymbolType`).
- **`SymbolSize (Long)`** - Größe des Symbols zur Visualisierung der Datenpunkte in 100stel Millimeter.
- **`SymbolBitmapURL (String)`** - Dateiname der Grafik zur Visualisierung der Datenpunkte.
- **`Lines (Boolean)`** - verbindet die Datenpunkte durch Linien.
- **`SplineType (Long)`** - Spline-Funktion zur Glättung der Linien (0: keine Spline-Funktion, 1: kubische Splines, 2: B-Splines).
- **`SplineOrder (Long)`** - Polynomial-Gewicht für Splines (nur für B-Splines).
- **`SplineResolution (Long)`** - Anzahl der Stützpunkte für Spline-Berechnung.

Flächendiagramme

Flächendiagramme (Service `com.sun.star.chart.AreaDiagram`) unterstützen eine X-Achse, zwei Y-Achsen und eine Z-Achse. Die Darstellung ist als 2D- oder 3D-Grafik möglich (Service `com.sun.star.chart.Dim3DDiagram`). Die Flächen sind stapelbar (`com.sun.star.chart.StackableDiagram`).

Balkendiagramme

Balkendiagramme (Service `com.sun.star.chart.BarDiagram`) unterstützen eine X-Achse, zwei Y-Achsen und eine Z-Achse. Die Darstellung ist als 2D- oder 3D-Grafik möglich (Service `com.sun.star.chart.Dim3DDiagram`). Die Balken sind stapelbar (`com.sun.star.chart.StackableDiagram`).

Sie stellen folgende Eigenschaften zur Verfügung:

- **`Vertical (Boolean)`** - stellt die Balken senkrecht dar, ansonsten erfolgt die Darstellung waagerecht.
- **`Deep (Boolean)`** - positioniert die Balken bei 3D-Darstellung hintereinander statt nebeneinander.
- **`StackedBarsConnected (Boolean)`** - verbindet die zugehörigen Balken in einem gestapelten Diagramm über Linien (nur bei horizontalen Charts verfügbar).
- **NumberOfLines (Long)** - Anzahl der Zeilen, die in einem gestapelten Diagramm als Linien statt als Balken dargestellt werden sollen.

Tortendiagramme

Tortendiagramme (Service `com.sun.star.chart.PieDiagram`) enthalten keine Achsen und sind nicht stapelbar. Die Darstellung ist als 2D- oder 3D-Grafik möglich (Service `com.sun.star.chart.Dim3DDiagram`).

KAPITEL 10

Datenbankzugriff

StarOffice verfügt über eine integrierte, systemunabhängige Datenbankschnittstelle namens Star Database Connectivity (SDBC). Ziel bei der Entwicklung der Schnittstelle war es, Zugriff auf möglichst viele unterschiedliche Datenquellen zu gewähren.

Um dies zu ermöglichen, erfolgt der Zugriff auf die Datenquellen über Treiber. Woher die Treiber ihre Daten nehmen, ist aus der Sicht eines SDBC-Anwenders unerheblich. Einige Treiber greifen auf dateibasierte Datenbanken zu und entnehmen ihnen die Daten direkt. Andere setzen auf Standardschnittstellen wie JDBC oder ODBC auf. Es existieren aber auch Spezialtreiber, die beispielsweise auf das MAPI-Adressbuch, LDAP-Verzeichnisse oder StarOffice-Tabellendokumente als Datenquellen zugreifen.

Da die Treiber auf UNO-Komponenten basieren ist es möglich, weitere Treiber zu entwickeln und so abermals neue Datenquellen zu erschließen. Details dazu finden Sie im StarOffice Developer's Guide.

SDBC ist vom Konzept her mit den in VBA vorhandenen Bibliotheken ADO beziehungsweise DAO vergleichbar. Es gestattet einen Highlevel-Zugriff auf Datenbanken, unabhängig von den darunter liegenden Datenbank-Backends.

Mit der Einführung von StarOffice 7 ist die Datenbank-Schnittstelle von StarOffice gewachsen. Während der Zugriff bisher primär über eine Reihe von Methoden des `Application`-Objektes erfolgte, untergliedert sich die Schnittstelle in StarOffice 7 in mehrere Objekte. Als Wurzelobjekt für die Datenbankfunktionen dient ein `DatabaseContext`.

SQL als Abfragesprache

Als Abfragesprache steht den Anwendern von SDBC die Sprache SQL zur Verfügung. Um die Unterschiede verschiedener SQL-Dialekte auszugleichen besitzt die SDBC-Komponente von StarOffice einen eigenen SQL-Parser. Dieser prüft die über das Abfragefenster eingegebenen SQL-Befehle und korrigiert einfache Syntaxfehler, etwa im Zusammenhang mit der Groß- und Kleinschreibung.

Ermöglicht ein Treiber Zugriff auf eine Datenquelle die kein SQL unterstützt, so muss er die übergebenen SQL-Befehle eigenständig auf die notwendigen nativen Zugriffe umsetzen.

Die SQL-Implementation von SDBC orientiert sich am SQL-ANSI-Standard. Microsoft-spezifische

Erweiterungen wie das `INNER JOIN`-Konstrukt werden nicht unterstützt. Diese sind durch Standard-Befehle zu ersetzen (`INNER JOIN` beispielsweise durch eine entsprechende `WHERE`-Klausel).

Arten des Datenbankzugriffs

Die Datenbankschnittstelle von StarOffice steht in den Anwendungen StarOffice Writer und StarOffice Calc sowie in den Datenbankformularen zur Verfügung.

In StarOffice Writer ist es möglich, Serienbriefe mit Hilfe von SDBC-Datenquellen zu erzeugen und auszudrucken. Außerdem ist es möglich, Daten über Drag&Drop aus dem Datenbankfenster in Textdokumente zu übernehmen.

Zieht der Anwender eine Datenbank-Tabelle in ein Tabellendokument, so erzeugt StarOffice einen Tabellenbereich, der sich per Mausklick aktualisieren lässt, wenn sich die Originaldaten geändert haben. Umgekehrt ist es möglich, Spreadsheet-Daten auf eine Datenbanktabelle zu ziehen und einen Datenbank-Import durchzuführen.

Schließlich stellt StarOffice einen Mechanismus für datenbankbasierte Formulare zur Verfügung. Dazu erstellt der Anwender zunächst ein gewöhnliches StarOffice Writer- beziehungsweise StarOffice Calc-Formular und verknüpft die Felder anschließend mit einer Datenbank.

Alle genannten Möglichkeiten basieren ausschließlich auf der Benutzerschnittstelle von StarOffice. Für die Nutzung der betreffenden Funktionen sind keine Programmierkenntnisse notwendig.

In diesem Kapitel geht es jedoch weniger um die genannten Funktionen. Viel mehr steht die Programmierschnittstelle von SDBC im Vordergund, die eine automatisierte Abfrage von Datenbanken gestattet und so eine viel größere Bandbreite an Anwendungen ermöglicht.

Für das Verständnis der folgenden Seiten sind Grundkenntnisse über die Funktionsweise von Datenbanken sowie die Abfragesprache SQL notwendig.

Datenquellen

Die Einbindung einer Datenbank in StarOffice erfolgt über die Erstellung einer sogenannten *Datenquelle*. Die Benutzeroberfläche stellt dazu im Menüpunkt **Extras** eine entsprechende Möglichkeit zur Verfügung. Doch auch der Programmierer hat die Möglichkeit, Datenquellen anzulegen und mit ihnen zu arbeiten.

Ausgangspunkt für den Zugriff auf eine Datenquelle bildet ein Datenbank-Kontext-Objekt, das über die Funktion `createUnoService` erzeugt wird. Es basiert auf dem Service `com.sun.star.sdb.DatabaseContext` und stellt das Wurzelobjekt für alle Datenbankoperationen dar.

Das folgende Beispiel zeigt, wie sich ein Datenbank-Kontext erstellen und zur Ermittlung der Namen aller verfügbaren Datenquellen verwenden lässt. Es gibt die Namen in einer Schleife über ein Meldungsfenster aus.

```
Dim DatabaseContext As Object
Dim Names
Dim I As Integer
```

```
DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")

Names = DatabaseContext.getElementNames()
For I = 0 To UBound(Names())
      MsgBox Names(I)
Next I
```

Die einzelnen Datenquellen basieren auf dem Service `com.sun.star.sdb.DataSource` und lassen sich aus dem Datenbank-Kontext über die Methode `getByName` ermitteln:

```
Dim DatabaseContext As Object
Dim DataSource As Object

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
```

Das Beispiel erzeugt ein `DataSource`-Objekt für eine Datenquelle namens *Customers*.

Datenquellen stellen eine Reihe von Eigenschaften zur Verfügung, die allgemeine Informationen über den Ursprung der Daten sowie Informationen über die Zugriffsmöglichkeiten liefern. Die Eigenschaften lauten:

- **`Name (String)`** – Name der Datenquelle.
- **`URL (String)`** – URL der Datenquelle in der Form *jdbc: subprotocol : subname* oder *sdbc: subprotocol : subname*.
- **`Info (Array)`** – Array mit `PropertyValue`-Paaren und Verbindungsparametern (im Regelfall zumindest Benutzername und Password).
- **`User (String)`** – Name des Benutzers.
- **`Password (String)`** – Benutzerpasswort (wird nicht gespeichert).
- **`IsPasswordRequired (Boolean)`** – das Passwort wird ständig benötigt und interaktiv vom Benutzer erfragt.
- **`IsReadOnly (Boolean)`** – ermöglicht ausschließlich lesende Zugriffe auf die Datenbank.
- **`NumberFormatsSupplier (Object)`** – Objekt mit den für die Datenbank verfügbaren Nummernformaten (unterstützt die Schnittstelle `com.sun.star.util.XNumberFormatsSupplier`, siehe Kapitel 7, Abschnitt *Zahlen-, Datums- und Textformat*).
- **`TableFilter (Array)`** – Liste der anzuzeigenden Tabellennamen.
- **`TableTypeFilter (Array)`** – Liste der anzuzeigenden Tabellentypen. Möglich sind die Werte `TABLE`, `VIEW` und `SYSTEM TABLE`.
- **`SuppressVersionColumns (Boolean)`** - unterdrückt die Anzeige von Spalten, die zur Versionsverwaltung dienen.

Die Datenquellen von StarOffice sind nicht 1:1 mit den Datenquellen in ODBC vergleichbar. Während eine ODBC-Datenquelle ausschließlich Informationen über den Ursprung der Daten umfasst, enthält eine Datenquelle in StarOffice darüber hinaus eine Reihe von Informationen zur Darstellung der Daten innerhalb des Datenbank-Fensters von StarOffice.

Abfragen

Einer Datenquelle lassen sich vordefinierte Abfragen zuordnen. StarOffice merkt sich die SQL-Befehle der Abfragen, so dass sie zu jedem Zeitpunkt zur Verfügung stehen. Abfragen dienen zur Vereinfachung der Arbeit mit Datenbanken, da sie sich mit einem einfachen Mausklick öffnen lassen und auch Anwendern ohne SQL-Kenntnissen die Möglichkeit zur Absetzung von SQL-Befehlen ermöglichen.

Hinter einer Abfrage verbirgt sich ein Objekt, das den Service `com.sun.star.sdb.QueryDefinition` unterstützt. Der Zugriff auf die Abfragen erfolgt über die Methode `QueryDefinitions` der Datenquelle.

Das folgende Beispiel zeigt, wie sich die Abfragen einer Datenquelle ermitteln lassen und gibt ihre Namen in einer Schleife aus.

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim QueryDefinitions As Object
Dim QueryDefinition As Object
Dim I As Integer

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
QueryDefinitions = DataSource.getQueryDefinitions()

For I = 0 To QueryDefinitions.Count() - 1
      QueryDefinition = QueryDefinitions(I)
      MsgBox QueryDefinition.Name
Next I
```

Neben der im Beispiel verwendeten Eigenschaft `Name` hält `com.sun.star.sdb.QueryDefinition` eine ganze Reihe weiterer Eigenschaften bereit:

- **`Name (String)`** – Name der Abfrage.
- **`Command (String)`** – SQL-Befehl der Abfrage (typischerweise ein `SELECT`-Befehl).
- **`UpdateTableName (String)`** – Für Abfragen, die auf mehreren Tabellen basieren: Name der Tabelle, in der Änderungen der Werte möglich sein sollen.
- **`UpdateCatalogName (String)`** – Name des Update-Tabellen-Katalogs.
- **`UpdateSchemaName (String)`** – Name des Update-Tabellen-Schemas.

Das folgende Beispiel zeigt, wie sich ein Abfrageobjekt programmgesteuert erstellen und einer Datenquelle zuweisen lässt.

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim QueryDefinitions As Object
Dim QueryDefinition As Object
Dim I As Integer

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
QueryDefinitions = DataSource.getQueryDefinitions()

QueryDefinition = createUnoService("com.sun.star.sdb.QueryDefinition")
QueryDefinition.Command = "SELECT * FROM Customer"

QueryDefinitions.insertByName("NewQuery", QueryDefinition)
```

Das Abfrageobjekt wird zunächst über den Aufruf `createUnoService` erzeugt, anschließend initialisiert und dann über `insertByName` in das `QueryDefinitions`-Objekt eingefügt.

Verknüpfungen mit Datenbankformularen

Um die Arbeit mit Datenquellen zu vereinfachen, bietet StarOffice eine Möglichkeit, die Datenquellen mit Datenbankformularen zu verknüpfen. Die Verknüpfungen sind über die Methode `getBookmarks()` verfügbar. Sie liefert einen benannten Container (`com.sun.star.sdb.DefinitionContainer`) zurück, der alle Verknüpfungen der Datenquelle enthält. Ein Zugriff auf die Bookmarks ist wahlweise über `Name` oder `Index` möglich.

Das folgende Beispiel ermittelt die URL des Bookmarks *MyBookmark*.

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim Bookmarks As Object
Dim URL As String
Dim I As Integer

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
Bookmarks = DataSource.Bookmarks()

URL = Bookmarks.getByName("MyBookmark")
MsgBox URL
```

Datenbank-Zugriffe

Für den Zugriff auf eine Datenbank ist eine Datenbankverbindung notwendig. Bei ihr handelt es sich um einen Übertragungskanal, der eine direkte Kommunikation mit der Datenbank ermöglicht. Im Gegensatz zu den im vorigen Abschnitt vorgestellten Datenquellen muss die Datenbankverbindung daher bei jedem Programmstart neu hergestellt werden.

StarOffice bietet viele verschiedene Arten an, Datenbankverbindungen aufzubauen. An dieser Stelle sei der Weg erklärt, der auf einer existierenden Datenquelle aufsetzt.

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim Connection As Object
Dim InteractionHandler as Object

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")

If Not DataSource.IsPasswordRequired Then
      Connection = DataSource.GetConnection("","")
Else
      InteractionHandler = createUnoService("com.sun.star.sdb.InteractionHandler")
      Connection = DataSource.ConnectWithCompletion(InteractionHandler)
End If
```

Der Beispielcode prüft zunächst, ob die Datenbank passwortgeschützt ist. Falls nicht, erzeugt er die gewünschte Datenbankverbindung über den Aufruf `GetConnection`. Die beiden leeren Zeichenfolgen in der Kommandozeile stehen für den Benutzernamen und das Passwort.

Ist die Datenbank passwortgeschützt, so erzeugt das Beispiel einen `InteractionHandler` und öffnet die Datenbankverbindung über die Methode `ConnectWithCompletion`. Der InteractionHandler sorgt dafür, dass StarOffice den Benutzer nach den gewünschten Anmeldedaten fragt.

Tabellen iterieren

Der eigentliche Zugriff auf eine Tabelle erfolgt in StarOffice im Regelfall über `ResultSet`-Objekte. Ein `ResultSet` ist eine Art Zeiger, der auf einen aktuellen Datensatz innerhalb einer über `SELECT`-Befehl abgesetzten Ergebnismenge zeigt.

Das Beispiel zeigt, wie sich mit einem `ResultSet` Werte aus einer Datenbanktabelle abfragen lassen.

```
Dim DatabaseContext As Object
Dim DataSource As Object
Dim Connection As Object
Dim InteractionHandler as Object
Dim Statement As Object
Dim ResultSet As Object

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")

If Not DataSource.IsPasswordRequired Then
      Connection = DataSource.GetConnection("","")
Else
      InteractionHandler = createUnoService("com.sun.star.sdb.InteractionHandler")
      Connection = DataSource.ConnectWithCompletion(InteractionHandler)
```

```
End If

Statement = Connection.createStatement()
ResultSet = Statement.executeQuery("SELECT CustomerNumber FROM Customer")

If Not IsNull(ResultSet) Then
      While ResultSet.next
            MsgBox ResultSet.getString(1)
      Wend
End If
```

Nachdem die Datenbankverbindung steht, erzeugt der Beispielcode zunächst über den Aufruf `Connection.createObject` ein `Statement`-Objekt. Dieses `Statement`-Objekt liefert daraufhin über den Aufruf `executeQuery` das eigentliche `ResultSet`. Der Programmcode prüft nun, ob das `ResultSet` tatsächlich existiert und durchwandert die Datensätze in diesem Fall über eine Schleife. Die gewünschten Werte (im Beispiel aus dem Feld `CustomerNumber`) gibt das `ResultSet` über die Methode `getString` zurück, wobei der Parameter 1 bestimmt, dass sich der Aufruf auf die Werte der ersten Spalte bezieht.

Das `ResultSet`-Objekt von SDBC ist mit dem `Recordset`-Objekt von DAO beziehungsweise ADO vergleichbar, das ebenfalls einen iterativen Zugriff auf eine Datenbank gewährt.

Der eigentliche Datenbankzugriff erfolgt in StarOffice 7 über ein `ResultSet`-Objekt. Es spiegelt den Inhalt einer Tabelle beziehungsweise das Ergebnis eines SQL-SELECT-Befehls wieder. Das `ResultSet`-Objekt stellte bisher die im `Application`-Objekt angesiedelten Methoden zur Navigation innerhalb der Daten zur Verfügung (z. B. `DataNextRecord`).

Typspezifische Methoden zum Auslesen von Werten

Wie aus dem Beispiel des vorstehenden Abschnittes ersichtlich, stellt StarOffice für den Zugriff auf Tabelleninhalte eine `getString`-Methode zur Verfügung. Die Methode liefert das Ergebnis in Form einer Zeichenfolge. Folgende `get`-Methoden stehen zur Verfügung:

- `getByte()` – unterstützt die SQL-Datentypen für Zahlen, Zeichen und Zeichenfolgen.
- `getShort()` – unterstützt die SQL-Datentypen für Zahlen, Zeichen und Zeichenfolgen.
- `getInt()` – unterstützt die SQL-Datentypen für Zahlen, Zeichen und Zeichenfolgen.
- `getLong()` – unterstützt die SQL-Datentypen für Zahlen, Zeichen und Zeichenfolgen.
- `getFloat()` – unterstützt die SQL-Datentypen für Zahlen, Zeichen und Zeichenfolgen.
- `getDouble()` – unterstützt die SQL-Datentypen für Zahlen, Zeichen und Zeichenfolgen.
- `getBoolean()` – unterstützt die SQL-Datentypen für Zahlen, Zeichen und Zeichenfolgen.
- `getString()` – unterstützt alle SQL-Datentypen.
- `getBytes()` – unterstützt die SQL-Datentypen für binäre Werte.
- `getDate()` – unterstützt die SQL-Datentypen für Zeichen, Zeichenfolgen, Datum und Timestamp.

- `getTime()` – unterstützt die SQL-Datentypen für Zeichen, Zeichenfolgen, Uhrzeit und Timestamp.
- `getTimestamp()` – unterstützt die SQL-Datentypen für Zeichen, Zeichenfolgen, Datum, Uhrzeit und Timestamp.
- `getCharacterStream()` – unterstützt die SQL-Datentypen für Zeichen, Zeichenfolgen und binäre Werte.
- `getUnicodeStream()` – unterstützt die SQL-Datentypen für Zeichen, Zeichenfolgen und binäre Werte.
- `getBinaryStream()` – Binäre Werte.
- `getObject()` – unterstützt alle SQL-Datentypen.

In allen Fällen ist die Nummer der Spalte als Parameter aufzuführen, deren Werte abgefragt werden sollen.

Die verschiedene ResultSet-Varianten

Datenbankzugriffe sind häufig eine geschwindigkeitskritische Angelegenheit. StarOffice bietet daher einige Möglichkeiten, `ResultSets` zu optimieren und damit die Zugriffsgeschwindigkeit zu steuern. Je mehr Funktionen ein `ResultSet` zur Verfügung stellt, um so komplexer ist deren Umsetzung im Regelfall und umso langsamer sind sie daher.

Ein einfaches `ResultSet`, so wie es im Abschnitt "Tabellen iterieren" vorgestellt wurde, bietet den kleinstmöglichen Funktionsumfang. Es gestattet ausschließlich das vorwärtsgewandte iterieren und Auslesen von Werten. Weitergehende Navigationsmöglichkeiten und das Ändern von Werten sind nicht vorgesehen.

Das für die Erzeugung des `ResultSet` verwendete `Statement`-Objekt bietet einige Eigenschaften an, die es ermöglichen, Einfluss auf die Funktionen des `ResultSet` zu nehmen:

- **`ResultSetConcurrency (const)`** – Angabe, ob Änderungen der Daten möglich sind oder nicht (Vorgaben gemäß `com.sun.star.sdbc.ResultSetConcurrency`).
- **`ResultSetType (const)`** – Angaben zur Art des `ResultSets` (Vorgaben gemäß `com.sun.star.sdbc.ResultSetType`).

Die in `com.sun.star.sdbc.ResultSetConcurrency` definierten Werte lauten:

- **`UPDATABLE`** - `ResultSet` erlaubt Änderungen der Werte.
- **`READ_ONLY`** – `ResultSet` erlaubt keine Änderungen.

Die Konstantengruppe `com.sun.star.sdbc.ResultSetConcurrency` hält folgende Vorgaben bereit:

- **`FORWARD_ONLY`** – `ResultSet` erlaubt nur eine vorwärtsgewandte Navigation.
- **`SCROLL_INSENSITIVE`** – `ResultSet` erlaubt beliebige Navigationen, Änderungen der Originaldaten bleiben jedoch unbemerkt.
- **`SCROLL_SENSITIVE`** – `ResultSet` erlaubt beliebige Navigationen, Änderungen der Originaldaten wirken sich auf das `ResultSet` aus.

Ein `ResultSet` mit den Attributen `READ_ONLY` und `SCROLL_INSENSITIVE` entspricht einem Recordset des Typs `Snapshot` in ADO beziehungsweise DAO.

Bei Verwendung der `ResultSet`-Attribute `UPDATABLE` und `SCROLL_SENSITIVE` ist der Funktionsumfang eines `ResultSets` mit einem `Recordset` des Typs `Dynaset` aus ADO beziehungsweise DAO vergleichbar.

Methoden zur Navigation in ResultSets

Hat ein `ResultSet` den Typ `SCROLL_INSENSITIVE` oder `SCROLL_SENSITIVE`, so unterstützt es eine ganze Reihe von Methoden zur Navigation im Datenbestand. Die zentralen Methoden lauten:

- **`next()`** – Navigation zum nächsten Datensatz.
- **`previous()`** – Navigation zum vorstehenden Datensatz.
- **`first()`** – Navigation zum ersten Datensatz.
- **`last()`** – Navigation zum letzten Datensatz.
- **`bevoreFirst()`** – Navigation vor den ersten Datensatz.
- **`afterLast()`** – Navigation hinter den letzten Datensatz.

Alle Methoden liefern einen Booleschen Parameter, der angibt, ob die Navigation erfolgreich war.

Zur Ermittlung der aktuellen Cursor-Position stehen folgende Prüfmethoden zur Verfügung, die alle einen Booleschen Wert zurückliefern:

- **`isBeforeFirst()`** – `ResultSet` steht vor dem ersten Datensatz.
- **`isAfterLast()`** – `ResultSet` steht hinter dem letzten Datensatz.
- **`isFirst()`** – `ResultSet` steht auf dem ersten Datensatz.
- **`isLast()`** – `ResultSet` steht hinter dem letzten Datensatz.

Ändern von Datensätzen

Wurde ein `ResultSet` mit dem Wert `ResultSetConcurrency = UPDATABLE` erzeugt, so ist sein Inhalt editierbar. Dies gilt natürlich nur so lange, wie der SQL-Befehl prinzipbedingt ein Rückschreiben der Daten in die Datenbank erlaubt. Bei komplexen SQL-Befehlen mit verknüpften Spalten oder akkumulierten Werten ist dies beispielsweise nicht möglich.

Für das Ändern der Werte bietet das `ResultSet`-Objekt `Update`-Methoden an, die analog zu den `get`-Methoden für das Auslesen von Werten aufgebaut sind. Die Methode `updateString` gestattet beispielsweise das Schreiben eines Strings.

Nach dem Ändern müssen die Werte durch einen Aufruf der Methode `updateRow()` in die Datenbank übertragen werden. Der Aufruf muss vor dem nächsten Navigationsbefehl erfolgen, da die Werte ansonsten verloren sind.

Handelt es sich bei den Änderungen um einen Irrtum, so lassen sie sich durch den Aufruf der Methode `cancelRowUpdates()` rückgängig machen. Dieser Aufruf ist jedoch nur so lange möglich, wie die Daten nicht mit `updateRow()` in die Datenbank zurück geschrieben wurden.

Dialoge

StarOffice-Dokumente können selbst definierte Dialogfenster und Formulare enthalten. Diese lassen sich mit StarOffice Basic-Makros verbinden und erweitern das Spektrum für den Einsatz von StarOffice Basic damit erheblich.

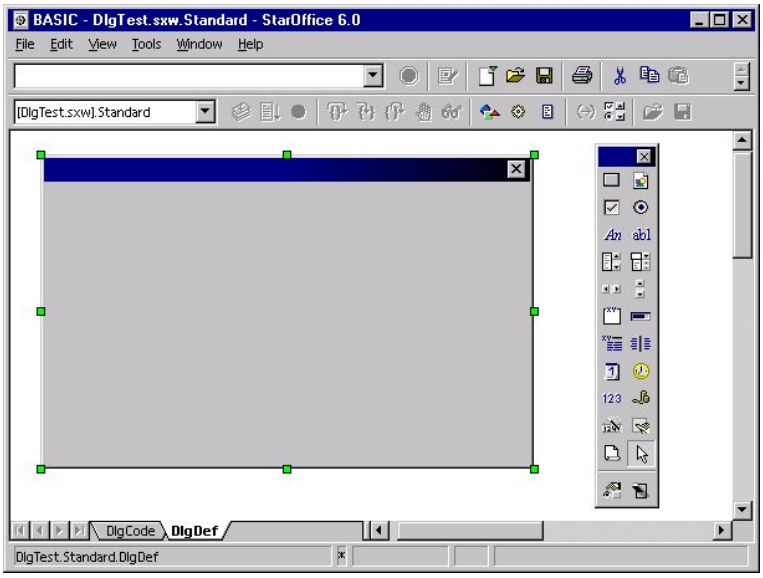
Der Phantasie des Programmierers sind bei der Arbeit mit Dialogen und Formularen keine Grenzen gesetzt. Sie können beispielsweise zur Visualisierung von Datenbankinformationen eingesetzt werden, Backup-Tätigkeiten übernehmen oder dem Anwender in Form von Autopiloten Schritt für Schritt durch die Erzeugung neuer Dokumente führen.

Arbeiten mit Dialogen

StarOffice Basic-Dialoge bestehen aus einem Dialogfenster, das Textfelder, Listboxen, Radio-Buttons und eine Reihe anderer Steuerelemente enthalten kann.

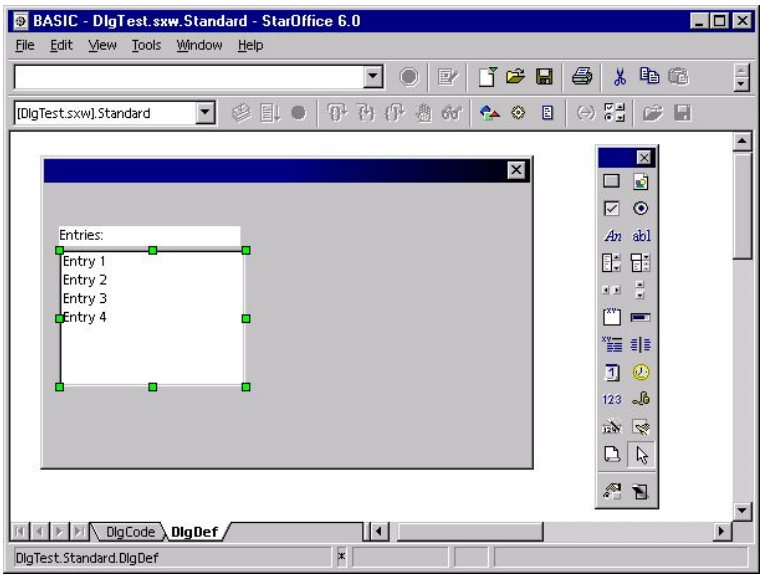
Erzeugen von Dialogen

Die Erzeugung von Dialogen erfolgt über den in StarOffice integrierten Dialog-Editor. Er dient zur Definition des Dialog-Aufbaus. Der Programmierer arbeitet dabei ähnlich wie mit einem Zeichenprogramm:



Er wählt die gewünschten Steuerelemente aus einer Design-Palette (rechts im Bild) aus und zieht sie mit der Maus in den Dialogbereich. Dort kann er dann ihre Position und ihre Größe festlegen.

Das Beispiel zeigt einen Dialog, der ein Beschriftungsfeld (Label) sowie eine Listbox enthält.



Ein definierter Dialog lässt sich über folgende Programmsequenz öffnen:

```
Dim Dlg As Object

DialogLibraries.LoadLibrary("Standard")
Dlg = CreateUnoDialog(DialogLibraries.Standard.DlgDef)

Dlg.Execute()
Dlg.dispose()
```

`CreateUnoDialog` erzeugt dabei ein Objekt namens `Dlg`, das auf den zugehörigen Dialog verweist. Vor dem Erzeugen des Dialoges muss sichergestellt sein, dass die verwendete Bibliothek (im Beispiel trägt sie den Namen `Standard`) geladen wurde. Ist dies nicht der Fall, so übernimmt dies die Methode `LoadLibrary`.

Ist das Dialog-Objekt `Dlg` initialisiert, so kann der Dialog über die Methode `Execute` angezeigt werden. Die Programmausführung bleibt so lange innerhalb des `Execute`-Aufrufes bis der Dialog geschlossen wird. Derartige Dialoge werden als *modal* bezeichnet, da sie das Programm (in StarOffice das entsprechende Dokumentfenster) in einen Warte*modus* versetzen und bis zur Beendigung des Dialoges keine Aktionen neben der Dialogbearbeitung zulassen.

An den Aufruf der Methode `Execute` schließt sich der Aufruf der Methode `dispose` an, die die vom Dialog belegten Ressourcen nach dessen Beendigung frei gibt.

Beenden von Dialogen

Beenden über eine Ok- oder Abbrechen-Schaltfläche

Enthält ein Dialog eine **Ok**- und/oder **Abbrechen**-Schaltfläche, so beendet ein Druck auf diese Schaltflächen den Dialog automatisch. Details zur Arbeit mit entsprechenden Schaltflächen finden Sie weiter unten in diesem Kapitel im Abschnitt über Steuerelemente.

Wird der Dialog unter Verwendung einer **Ok**-Schaltfläche geschlossen, liefert die `Execute`-Methode einen Rückgabewert `1`, ansonsten den Wert 0.

```
Dim Dlg As Object

DialogLibraries.LoadLibrary("Standard")
Dlg = CreateUnoDialog(DialogLibraries.Standard.MyDialog)

Select Case Dlg.Execute()
Case 1
      MsgBox "Ok pressed"
Case 0
      MsgBox "Cancel pressed"
End Select
```

Beenden über die Schließfläche im Fensterbalken

Klickt der Anwender auf die Schließfläche im Fensterbalken des Dialoges, so wird dieser ebenfalls automatisch beendet. In diesem Fall liefert die `Execute`-Methode des Dialoges wie beim Drücken einer Abbrechen-Schaltfläche den Wert 0 zurück.

Beenden über expliziten Programmaufruf

Auch der Programmierer hat die Möglichkeit, ein geöffnetes Dialogfenster zu schließen. Der entsprechende Programmaufruf lautet:

```
Dlg.endExecute()
```

Die Methode `endExecute` beendet den Dialog.

Zugriff auf einzelne Steuerelemente

Wie bereits erwähnt, kann ein Dialog beliebige Steuerelemente enthalten. Die Steuerelement-Objekte sind über die Methode `getControl` erreichbar. Als Parameter erhält `getControl` dazu den Namen des Steuerelementes.

```
Dim Ctl As Object

Ctl = Dlg.getControl("MyButton")
Ctl.Label = "New Label"
```

Das Beispiel ermittelt das Objekt für das Steuerelement `MyButton` und initialisiert die Objektvariable `Ctl` mit einer Referenz darauf. Anschließend setzt es die Eigenschaft `Label` des Steuerelementes auf den Wert `New Label`.

Bei den Namen von Steuerelementen achtet StarOffice Basic auf Groß- und Kleinschreibung.

Arbeiten mit dem *Model* von Dialogen und Steuerelementen

In den voran gegangenen Abschnitten ging es bereits mehrfach um die Objekte von Dialogen und Steuerelementen sowie ihre Eigenschaften und Methoden.

Über diese Methoden und Eigenschaften hinaus bieten sowohl die Dialog- als auch Steuerelement-Objekte ein untergeordnetes `Model`-Objekt an, das einen direkten Zugriff auf die Inhalte des Dialoges oder Steuerelementes ermöglicht. Hier spiegelt sich die in StarOffice an vielen Stellen zu findende Aufteilung zwischen den sichtbaren Programm-Elementen (*View*) und den dahinter liegenden Daten bzw. Dokumenten (*Model*) wider.

Im Zusammenhang mit Dialogen verläuft die Grenze zwischen Daten und Darstellung jedoch nicht immer so klar wie in den anderen API-Bereichen von StarOffice. Und so kommt es vor, dass Teile des APIs sowohl über das View als auch über das Model verfügbar sind.

Der programmgesteuerte Zugriff auf das Model von Dialog- und Steuerelement-Objekten erfolgt über die Eigenschaft `Model`.

```
Dim cmdNext As Object

cmdNext = Dlg.getControl("cmdNext")
cmdNext.Model.Enabled = False
```

Das Beispiel deaktiviert die Schaltfläche `cmdNtext` innerhalb des Dialoges `Dlg` unter Zuhilfenahme des Model-Objektes von `cmdNtext`.

Eigenschaften

Name und Titel

Jedes Steuerelement hat einen eindeutigen Namen. Dieser kann über die Model-Eigenschaft

- **`Model.Name (String)`** – Steuerelementname

erfragt werden. Ein Dialog besitzt des weiteren einen Titel, der in der Titelleiste des Dialoges aufgeführt wird. Er kann festgelegt und ausgelesen werden über die Model-Eigenschaft

- **`Model.Title (String)`** – Dialogtitel (gilt nur bei Dialogen).

Position und Größe

Größe und Position eines Steuerelementes lassen sich über folgende Eigenschaften abfragen:

- **`Model.Height (long)`** – Höhe des Steuerelementes (in ma-Einheiten).
- **`Model.Width (long)`** – Breite des Steuerelementes (in ma-Einheiten).
- **`Model.PositionX (long)`** – X-Position des Steuerelementes, gemessen von der linken Innenkante des Dialoges (in ma-Einheiten).
- **`Model.PositionY (long)`** – Y-Position des Steuerelementes, gemessen von der oberen Innenkante des Dialoges (in ma-Einheiten).

Um eine möglichst hohe Plattformunabhängigkeit zu gewährleisten verwendet StarOffice zur Angabe von Positionen und Größen innerhalb von Dialogen die interne Einheit Namens *Map AppFont (ma)*. Ein ma entspricht einem Achtel der durchschnittlichen Höhe eines Zeichens aus dem im Betriebssystem definierten Systemfonts sowie einem Viertel der Breite eines entsprechenden Zeichens. Durch die Verwendung von ma-Einheiten stellt StarOffice sicher, dass ein Dialog trotz unterschiedlichen Systemen und bei verschiedenen Systemeinstellungen stets gleich erscheint.

Soll die Größe oder Position von Steuerelementen zur Laufzeit geändert werden, so empfiehlt es sich, die Gesamtgröße des Dialoges zu ermitteln und die Werte für die Steuerelemente an den entsprechenden Teilverhältnissen auszurichten.

Die Einheit Map AppFont (ma) löst die bisher verwendete Einheit Twips ab, um eine bessere Plattformunabhängigkeit zu garantieren.

Focus und Tabulator-Reihenfolge

Der Anwender kann die Steuerelemente durch Drücken der Tabulatortaste nacheinander durchwandern. Folgende Eigenschaften stehen in diesem Zusammenhang im Model der Steuerelemente zur Verfügung:

- **`Model.Enabled (Boolean)`** – aktiviert das Steuerelement.
- **`Model.Tabstop (Boolean)`** – ermöglicht die Erreichbarkeit des Steuerelementes über die Tabulatortaste.
- **`Model.TabIndex (Long)`** – Position des Steuerelementes in der Aktivierungsreihenfolge.

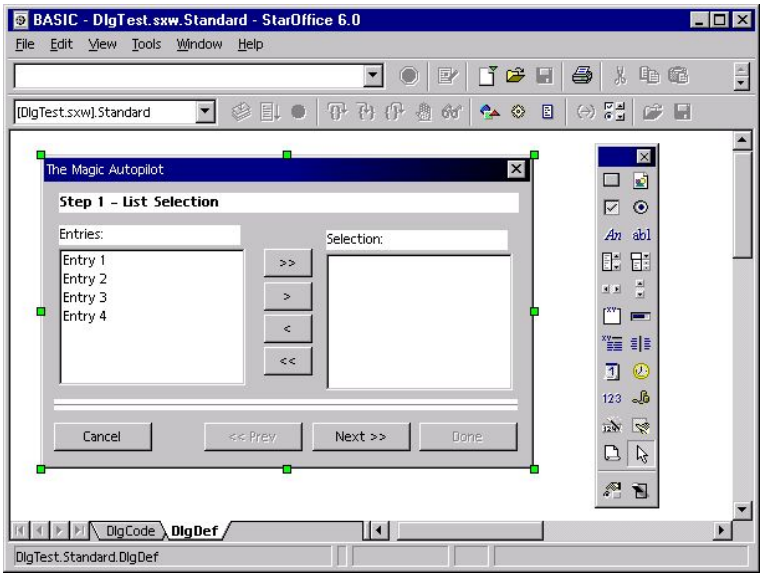
Schließlich bietet das Steuerelement selbst eine Methode `getFocus` an, die dafür sorgt, dass das zugrunde liegende Steuerelement den Fokus erhält:

- **`getFocus`** – Steuerelement erhält den Focus (nur für Dialoge).

Mehrseitige Dialoge

Ein Dialog kann in StarOffice mehrere Seiten umfassen, die nacheinander eingeblendet werden. Diese Funktion kommt unter anderem in den Autopiloten von StarOffice zum Einsatz. Der Dialog selbst bietet dazu eine Eigenschaft `Step` an, die die aktuelle Seite des Dialoges festlegt. In den Steuerelementen steht ebenfalls eine `Step`-Eigenschaft zur Verfügung. Sie bestimmt, auf welcher Seite das jeweilige Steuerelement sichtbar sein soll.

Einen Sonderfall bildet dabei der `Step`-Wert `0`. Wird er innerhalb des Dialoges gesetzt, so sind sämtliche Steuerelemente unabhängig von ihrem `Step`-Wert sichtbar. Umgekehrt ist ein Steuerelement mit `Step = 0` auf allen Seiten des Dialoges sichtbar.



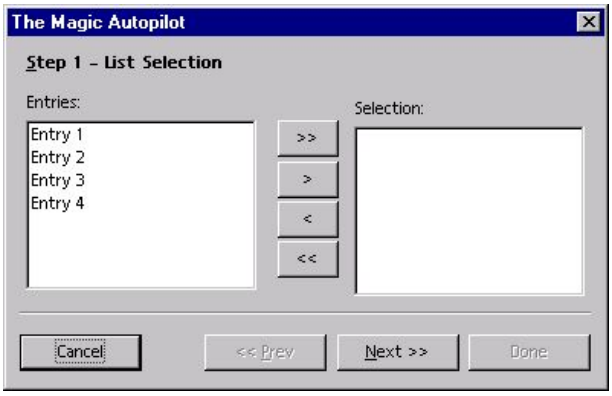
Bei dem vorstehenden Autopiloten bietet es sich z.B. an, der Trennlinie sowie den Schaltflächen `Cancel`, `Prev`, `Next` und `Done` den `Step`-Wert `0` zuzuweisen, um diese Elemente auf allen Seiten einzublenden. Die oberen Werte ließen sich dagegen einer Einzelseite zuweisen (etwa der Seite `1`).

Der nachfolgende Programmcode zeigt, wie sich der `Step`-Wert in den Ereignis-Handlern der Schaltflächen `Next` und `Prev` erhöhen beziehungsweise reduzieren lässt. Außerdem passt er den Status der Schaltflächen an.

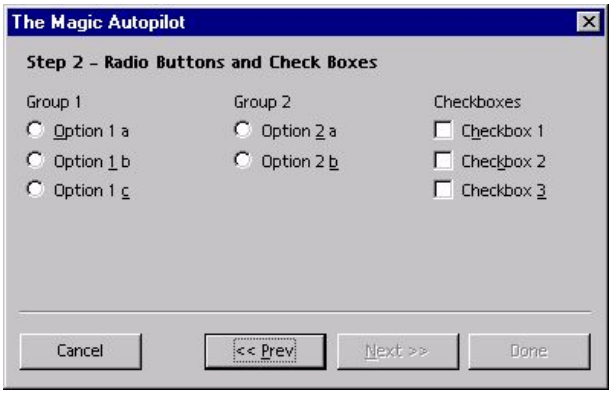
```
Sub cmdNext_Initiated
      Dim cmdNext As Object
      Dim cmdPrev As Object

      cmdPrev = Dlg.getControl("cmdPrev")
      cmdNext = Dlg.getControl("cmdNext")

      cmdPrev.Model.Enabled = Not cmdPrev.Model.Enabled
      cmdNext.Model.Enabled = False

      Dlg.Model.Step = Dlg.Model.Step + 1
End Sub

Sub cmdPrev_Initiated
      Dim cmdNext As Object
      Dim cmdPrev As Object

      cmdPrev = Dlg.getControl("cmdPrev")
      cmdNext = Dlg.getControl("cmdNext")

      cmdPrev.Model.Enabled = False
      cmdNext.Model.Enabled = True

      Dlg.Model.Step = Dlg.Model.Step - 1
End Sub
```

Das Beispiel setzt voraus, dass eine globale Variable `Dlg` existiert, die auf den geöffneten Dialog verweist. Der Dialog ändert dabei wie folgt sein Erscheinen:

Seite 1:



Seite 2:



Ereignisse

Die Dialoge und Formulare von StarOffice basieren auf einem ereignisorientierten Programmiermodell. Der Programmierer hat dabei die Möglichkeit, den Steuerelementen so genannte *Ereignis-Handler* zuzuweisen, die beim Auftreten einer bestimmten Aktion eine zuvor festgelegte Prozedur innerhalb des Programmcodes aufrufen.

Klickt der Anwender beispielsweise auf eine Schaltfläche, dann löst das Steuerelement das Ereignis `When initiating` aus. Hat der Programmierer dem entsprechenden Ereignis eine StarOffice Basic-Prozedur zugewiesen, so wird diese in diesem Moment ausgeführt.

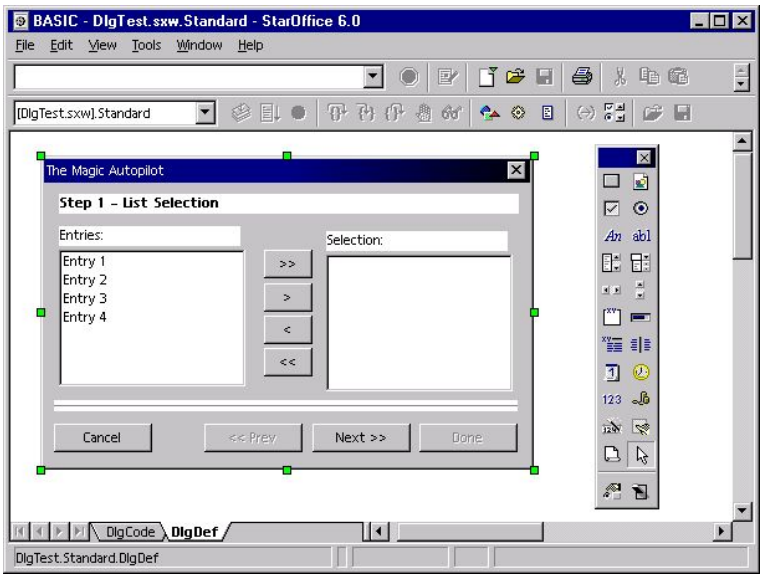
Im Rahmen einer Ereignisbehandlung kann man Dokumente bearbeiten oder Datenbanken öffnen. Es sind auch Zugriffe auf andere Steuerelemente möglich.

StarOffice-Steuerelemente kennen verschiedene Arten von Ereignissen, die jeweils in unterschiedlichen Situationen ausgelöst werden. Sie lassen sich in vier Gruppen unterteilen:

- **Maussteuerung**: Ereignisse, die Aktionen mit der Maus widerspiegeln (zum Beispiel einfache Mausbewegungen oder Klicks auf eine bestimmte Stelle des Bildschirms).
- **Tastatursteuerung**: Ereignisse, die durch Tastatureingaben des Anwenders ausgelöst werden.

- **Fokusänderung**: Ereignisse, die StarOffice beim Aktivieren beziehungsweise Deaktivieren von Steuerelementen veranlasst.
- **Steuerelemementspezifische Ereignisse**: Ereignisse, die nur im Zusammenhang mit bestimmten Steuerelementen auftreten.

Bei der Arbeit mit Ereignissen ist der zugehörige Dialog in der Entwicklungsumgebung von StarOffice anzulegen und mit den gewünschten Steuerelementen auszustatten (bzw. das Dokument, falls es sich um ein Formular handelt).



Das oben stehende Bild zeigt die StarOffice Basic-Entwicklungsumgebung mit einem Dialogfenster, das zwei Listboxen enthält, deren Daten über die Schaltflächen in der Mitte zwischen den beiden Listboxen hin und her geschoben werden können.

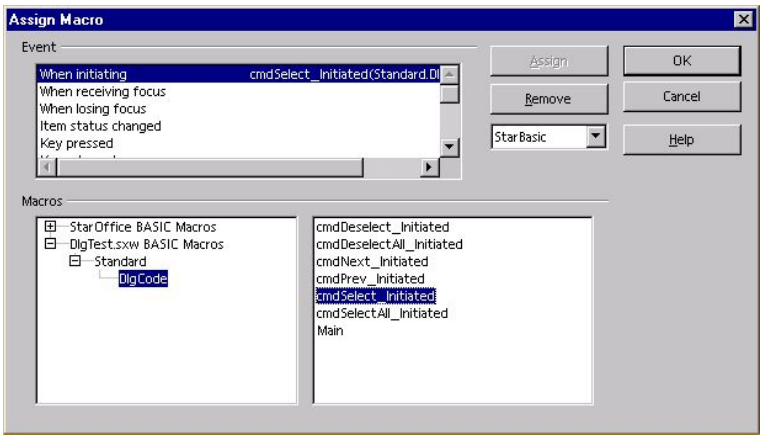
Steht das Layout, so gilt es, die zugehörigen StarOffice Basic-Prozeduren anzulegen, die im Rahmen der Ereignisbehandlung aufgerufen werden sollen. Die Prozeduren können zwar in einem beliebigen Modul stehen, es bietet sich jedoch an, sie auf eines oder zumindest wenige Module zu beschränken.

Die Namen der Prozeduren sind frei wählbar. Sie sollten aus Gründen der Übersichtlichkeit jedoch klar herausstellen, dass die Prozeduren zur Ereignisbehandlung dienen. Ein direkter Ansprung allgemeiner Programm-Prozeduren führt in der Regel zu unübersichtlichem Programmcode. In diesem Fall empfiehlt es sich, eine weitere Prozedur anzulegen, die als Startpunkt für die Ereignisbehandlung dient, selbst wenn sie nur einen einzigen Aufruf, nämlich den der Zielprozedur durchführt.

Der folgende Beispielcode zeigt eine Prozedur, die einen Eintrag von der linken in die rechte Listbox des Dialoges verschiebt.

```
Sub cmdSelect_Initiated
      Dim objList As Object
      lstEntries = Dlg.getControl("lstEntries")
      lstSelection = Dlg.getControl("lstSelection")

      If lstEntries.SelectedItem > 0 Then
            lstSelection.AddItem(lstEntries.SelectedItem, 0)
            lstEntries.removeItems(lstEntries.SelectedItemPos, 1)
      Else
            Beep
      End If
End Sub
```

Ist die Prozedur in StarOffice Basic angelegt, so lässt sie sich dem gewünschten Ereignis über das Eigenschaftsfenster im Dialog-Editor zuweisen.



Der Zuweisungsdialog listet sämtliche StarOffice Basic-Prozeduren auf und gestattet es, diese dem zugehörigen Ereignis über die Schaltfläche Assign zuzuweisen.

Die Zuweisung erfolgt durch einen expliziten Klick auf die Schaltfläche **Assign**. Wird der Dialog ohne Aufruf von **Assign** einfach nur mit **Ok** geschlossen, erfolgt *keine* Zuweisung.

Parameter

Das Auftreten eines einzelnen Ereignisses ist in vielen Fällen nicht ausreichend, um angemessen darauf zu reagieren. Häufig benötigt der Programmierer Zusatzinformationen. So wird zur Bearbeitung eines Mausklicks eventuell die Bildschirmposition benötigt, an der die Maustaste gedrückt wurde.

Die Prozeduren zur Ereignisbehandlung können in StarOffice Basic daher mit einem Objekt-Parameter ausgestattet werden. Die Definition könnte wie folgt aussehen:

```
Sub ProcessEvent(Event As Object)

End Sub
```

Wie genau das Objekt `Event` aufgebaut ist und welche Eigenschaften es enthält, hängt von der Art des Ereignisses ab, das den Prozeduraufruf ausgelöst hat. Details dazu erfahren Sie in den nächsten Abschnitten.

Unabhängig vom Typ des Ereignisses gewähren alle Objekte Zugriff auf das jeweilige Steuerelement und sein Model. Das Steuerelement ist über

```
Event.Source
```

erreichbar und sein Model über

```
Event.Source.Model
```

Über diese Eigenschaft lässt sich der Auslöser eines Ereignisses innerhalb von Ereignis-Handlern ermitteln. So ist es möglich, Ereignis-Handler zu schreiben, die für mehrere Steuerelemente zuständig sind.

Maus-Ereignisse

StarOffice Basic kennt folgende Mausereignisse:

- **`Mouse moved`** – Die Maus wurde bewegt.
- **`Mouse moved while key pressed`** – Die Maus wurde bewegt, während der Anwender eine Taste gedrückt hält (Drag-Bewegung).
- **`Mouse button pressed`** – Der Anwender hat eine Taste der Maus gedrückt.
- **`Mouse button released`** – Der Anwender hat eine Maustaste losgelassen.
- **`Mouse outside`** – Die Maus hat das aktuelle Fenster verlassen.

Der Aufbau des zugehörigen Event-Objektes wird in der Struktur `com.sun.star.awt.MouseEvent` definiert, die folgende Informationen bereit hält:

- **`Buttons (short)`** – Gedrückte Taste (eine oder mehrere Konstanten gemäß `com.sun.star.awt.MouseButton`).
- **`X (long)`** – X-Koordinate der Maus, gemessen in Pixel von der linken oberen Ecke des Steuerelementes.
- **`Y (long)`** – Y-Koordinate der Maus, gemessen in Pixel von der linken oberen Ecke des Steuerelementes.
- **`ClickCount (long)`** – Anzahl der Klicks, die mit dem Mausereignis verbunden sind (falls StarOffice hinreichend schnell reagieren kann, ist ClickCount auch bei einem Doppelklick 1, da für jeden Klick ein eigenes Ereignis abgesetzt wird.

Die in `com.sun.star.awt.MouseButton` definierten Konstanten für die Maustasten lauten:

- **`LEFT`** – linke Maustaste
- **`RIGHT`** – rechte Maustaste
- **`MIDDLE`** – mittlere Maustaste

Das folgende Beispiel gibt die Position sowie die gedrückte Taste der Maus aus:

```
Sub MouseUp(Event As Object)
      Dim Msg As String
      Msg = "Keys: "
      If Event.Buttons AND com.sun.star.awt.MouseButton.LEFT Then
```

```
            Msg = Msg & "LEFT "
      End If
      If Event.Buttons AND com.sun.star.awt.MouseButton.RIGHT Then
            Msg = Msg & "RIGHT "
      End If
      If Event.Buttons AND com.sun.star.awt.MouseButton.MIDDLE Then
            Msg = Msg & "MIDDLE "
      End If
      Msg = Msg & Chr(13) & "Position: "
      Msg = Msg & Event.X & "/" & Event.Y
      MsgBox Msg
End Sub
```

Die VBA-Ereignisse `Klick` und `Doppelklick` stehen in StarOffice Basic nicht zur Verfügung. Weichen Sie für das `Klick`-Ereignis auf das `MouseUp`-Ereignis von StarOffice Basic aus und bilden Sie das `Doppelklick`-Ereignis durch eine Änderung der Applikationslogik nach.

Tastatur-Ereignisse

Folgende Tastaturereignisse stehen in StarOffice Basic zur Verfügung:

- **`Key pressed`** – Der Anwender hat eine Taste gedrückt.
- **`Key released`** – Der Anwender hat eine Taste losgelassen.

Beide Ereignisse beziehen sich auf *logische* Tastendrücke und nicht auf *physikalische*. Drückt der Anwender beispielsweise mehrere Tasten, die zur Ausgabe eines einzigen Zeichens führen (z.B. beim Setzen eines Akzents), so erzeugt StarOffice Basic nur für das Zeichen ein Ereignis.

Ein einzelner Druck auf eine Modifizierertaste wie die Groß-/Kleintaste oder die Alt-Taste verursacht kein eigenständiges Ereignis.

Informationen über die gedrückte Taste finden sich im Event-Objekt, das StarOffice Basic der Prozedur zur Ereignisbehandlung mit gibt. Sie enthält folgende Eigenschaften:

- **`KeyCode (short)`** – Code der gedrückten Taste (Vorgabewerte gemäß `com.sun.star.awt.Key`).
- **`KeyChar (String)`** – eingegebenes Zeichen (unter Berücksichtigung von Modifizierer-Tasten).

Das folgende Beispiel zeigt, wie sich über die Eigenschaft `KeyCode` ermitteln lässt, ob die Eingabetaste, die Tabulatortaste, oder eine andere Steuertaste gedrückt wurde. In diesem Fall liefert es den Namen der Taste zurück, ansonsten das eingegebene Zeichen:

```
Sub KeyPressed(Event As Object)
      Dim Msg As String
      Select Case Event.KeyCode
      Case com.sun.star.awt.Key.RETURN
            Msg = "Return pressed"
      Case com.sun.star.awt.Key.TAB
            Msg = "Tab pressed"
      Case com.sun.star.awt.Key.DELETE
            Msg = "Delete pressed"
      Case com.sun.star.awt.Key.ESCAPE
            Msg = "Escape pressed"
      Case com.sun.star.awt.Key.DOWN
            Msg = "Down pressed"
      Case com.sun.star.awt.Key.UP
            Msg = "Up pressed"
      Case com.sun.star.awt.Key.LEFT
            Msg = "Left pressed"
      Case com.sun.star.awt.Key.RIGHT
            Msg = "Right pressed"
      Case Else
            Msg = "Character " & Event.KeyChar & " entered"
      End Select
      MsgBox Msg
End Sub
```

Informationen zu weiteren Tastatur-Konstanten finden Sie in der API-Referenz unter der Konstantengruppe `com.sun.star.awt.Key`.

Fokus-Ereignisse

Fokus-Ereignisse zeigen an, dass ein Steuerelement einen Fokus bekommen hat oder diesen verliert. Aus ihnen lässt sich beispielsweise ableiten, dass der Anwender mit der Bearbeitung eines Steuerelementes fertig ist und die Software nun andere Elemente eines Dialoges aktualisieren kann. Folgende Ereignisse existieren:

- **`When receiving focus`** – Element erhält den Fokus.
- **`When losing focus`** – Element verliert den Fokus.

Die `Event`-Objekte der Fokus-Ereignisse sind wie folgt aufgebaut:

- **`FocusFlags (short)`** – Ursache des Fokus-Wechsels (Vorgabewert gemäß `com.sun.star.awt.FocusChangeReason`).
- **`NextFocus (Object)`** – Objekt, das den Fokus erhält (nur beim Ereignis `When losing focus`)
- **`Temporary (Boolean)`** – der Fokus wurde nur vorübergehend abgegeben.

Steuerelementspezifische Ereignisse

Neben den vorstehenden Ereignissen, die von jedem Steuerelement unterstützt werden, existieren einige steuerelementspezifische Ereignisse, die nur für bestimmte Steuerelemente definiert sind. Die wichtigsten Ereignisse in diesem Zusammenhang lauten:

- **`When Item Changed`** – Der Wert eines Steuerelementes wurde geändert.
- **`Item Status Changed`** – Der Status eines Steuerelementes wurde geändert.
- **`Text modified`** – Der Text eines Steuerelementes wurde geändert.
- **`When initiating`** – Eine mit dem Steuerelement ausführbare Aktion wurde ausgelöst (zum Beispiel eine Schaltfläche gedrückt).

Bei der Arbeit mit Ereignissen sind einige Besonderheiten zu beachten. So wird das Ereignis `When initiating` bei einigen Steuerelementen bei jedem Mausklick ausgelöst (etwa bei Radio Buttons). Dabei wird nicht geprüft, ob sich der Status des Steuerelementes tatsächlich geändert hat. Möchte man derartige „Blindereignisse“ vermeiden, so empfiehlt es sich, den alten Wert des Steuerelementes in einer globalen Variable abzulegen und bei der Ausführung des Ereignisses zu prüfen, ob er tatsächlich geändert wurde.

Von den vorstehenden Ereignissen sind die Parameter insbesondere für das Ereignis `Item Status Changed` interessant. Sein Ereignis-Objekt umfasst folgende Eigenschaften:

- **`Selected (long)`** – Neu selektierter Eintrag.
- **`Highlighted (long)`** – Neu hervorgehobener Eintrag.
- **`ItemId (long)`** – Id des Eintrages.

Dialog-Steuerelemente im Detail

Wie die vorstehenden Beispiele bereits gezeigt haben, kennt StarOffice Basic eine Reihe von Steuerelementen. Sie lassen sich grob in folgende Gruppen einteilen:

Eingabefelder:

- Textfelder
- Datumsfelder
- Zeitfelder
- Zahlenfelder
- Währungsfelder
- Frei formatierbare Felder

Schalter:

- gewöhnliche Schaltflächen
- Checkboxen
- Radio Buttons

Auswahllisten:

- Listboxen
- Komboboxen

Sonstige Steuerelemente:

- Scrollbars (horizontal und vertikal)
- Gruppenfelder
- Verlaufsbalken
- Trennlinien (horizontal und vertikal)
- Grafiken
- Dateiauswahlfelder

Die wichtigsten dieser Steuerelemente werden im Nachfolgenden vorgestellt.

Schaltflächen

Eine Schaltfläche führt Aktionen auf einen Klick des Anwenders hin aus.

Im einfachsten Fall beschränkt sich eine Schaltfläche darauf, ein `When Initiating`-Ereignis auszulösen, wenn sie vom Benutzer aktiviert wird. Alternativ besteht jedoch die Möglichkeit, eine zusätzliche Aktion mit der Schaltfläche zu verknüpfen, die den übergeordneten Dialog mit Hilfe der Eigenschaft `PushButtonType` beendet. Hat die Eigenschaft den Wert 0, so hat ein Klick auf die Schaltfläche keine direkten Auswirkungen auf den Dialog. Trägt sie den Wert 1, so wird der Dialog beendet und die `Execute`-Methode des Dialoges liefert den Wert 1 zurück. Dies bedeutet, dass der Dialog ordnungsgemäß (d.h. mit **Ok**) beendet wurde. Erhält `PushButtonType` den Wert 2, so beendet dies den Dialog und die `Execute`-Methode liefert den Wert 0 zurück, was für einen Abbruch des Dialoges steht.

Hier eine Aufstellung sämtlicher Eigenschaften, die über das Model einer Schaltfläche verfügbar sind:

- **`Model.BackgroundColor (long)`** – Farbwert des Hintergrunds.
- **`Model.DefaultButton (Boolean)`** – die Schaltfläche dient als Vorgabewert. Sie reagiert in diesem Fall auch dann auf die Eingabetaste, wenn sie keinen Fokus besitzt.
- **`Model.FontDescriptor (struct)`** – Struktur mit Angabe zum zu verwendenden Font (gemäß Struktur `com.sun.star.awt.FontDescriptor`).
- **`Model.Label (String)`** – Beschriftung der Schaltfläche.
- **`Model.Printable (Boolean)`** – das Steuerelement ist druckbar.
- **`Model.TextColor (Long)`** – Textfarbe des Steuerelementes.
- **`Model.HelpText (String)`** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **`Model.HelpURL (String)`** – URL der Online-Hilfe für das betreffende Steuerelement.

- **`PushButtonType (short)`** – Aktion, die mit der Schaltfläche verknüpft ist (0: keine Aktion, 1: Ok, 2: Abbrechen).

Radio Buttons

Radio Buttons werden gewöhnlich in Gruppen eingesetzt und gestatten es dem Benutzer, eine von mehreren Optionen auszuwählen. Sobald eines der Elemente aktiviert wird, werden alle anderen Elemente der Gruppe deaktiviert. Dies sorgt dafür, dass stets genau ein Radio Button gesetzt ist.

Ein Radio-Button-Steuerelement stellt zwei Eigenschaften zur Verfügung:

- **`State (Boolean)`** – aktiviert den Button.
- **`Label (String)`** – Beschriftung des Steuerelementes.

Darüber hinaus stehen verschiedene Eigenschaften über das Model der Radio Buttons bereit:

- **`Model.FontDescriptor (struct)`** – Struktur mit Angaben zum zu verwendenden Font (gemäß `com.sun.star.awt.FontDescriptor`).
- **`Model.Label (String)`** - Beschriftung des Steuerelementes.
- **`Model.Printable (Boolean)`** – Steuerelement ist druckbar.
- **`Model.State (Short)`** – Falls 1, ist die Option aktiviert, ansonsten deaktiviert.
- **`Model.TextColor (Long)`** – Textfarbe des Steuerelementes.
- **`Model.HelpText (String)`** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **`Model.HelpURL (String)`** – URL der Online-Hilfe für das betreffende Steuerelement.

Sollen mehrere Radio Buttons zu einer Gruppe zusammengefasst werden, so müssen sie in der Aktivierungsreihenfolge (Eigenschaft `Model.TabIndex`, im Dialog-Editor als Order bezeichnet) ohne Lücke aufeinander folgen. Wird die Aktivierungsreihenfolge durch ein anderes Steuerelement unterbrochen, so beginnt StarOffice automatisch mit einer neuen Steuerelement-Gruppe, die unabhängig von der ersten Steuerelementgruppe aktivierbar ist.

Im Gegensatz zu VBA ist es in StarOffice Basic nicht möglich, Radio Buttons in ein Gruppen-Steuerelement einzufügen und so zusammen zu fassen. Das Gruppen-Steuerelement von StarOffice Basic sorgt ausschließlich für eine optische Gliederung, indem es einen Rahmen um die Steuerelemente zeichnet. Der Zusammenschluss mehrerer Radio Buttons zu einer Gruppe erfolgt in StarOffice Basic ausschließlich über den vorstehend erklärten Mechanismus auf der Basis der Aktivierungsreihenfolge.

Checkboxen

Checkboxen dienen zur Eingabe eines Ja-Nein-Wertes. Sie können je nach Modus zwei oder drei Zustände annehmen. Neben den Zuständen Ja und Nein ist dies ein *Schwebezustand*, der weder für Ja noch für Nein steht. Er kommt zum Einsatz, falls der jeweilige Ja-Nein-Status mehrdeutig beziehungsweise unklar ist.

Checkboxen stellen folgende Eigenschaften bereit:

- **State (Short)** – Zustand der Checkbox (0: nein, 1: ja, 2: Schwebezustand).
- **Label (String)** – Beschriftung des Steuerelementes.
- **enableTriState (Boolean)** – läßt neben den Zuständen aktiviert und deaktiviert auch den Schwebezustand zu.

Das Model-Objekt einer Checkbox bietet folgende Eigenschaften an:

- **Model.FontDescriptor (struct)** – Struktur mit Angaben zum zu verwendenden Font (gemäß Struktur `com.sun.star.awt.FontDescriptor`).
- **Model.Label (String)** – Beschriftung des Steuerelementes.
- **Model.Printable (Boolean)** – das Steuerelement ist druckbar.
- **Model.State (Short)** – Zustand der Checkbox (0: nein, 1: ja, 2: Schwebezustand).
- **Model.Tabstop (Boolean)** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **Model.TextColor (Long)** – Textfarbe des Steuerelementes.
- **Model.HelpText (String)** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **Model.HelpURL (String)** – URL der Online-Hilfe für das betreffende Steuerelement.

Textfelder

Textfelder dienen dazu, Zahlen und Text vom Benutzer zu erfragen. Die Grundlage für Textfelder bildet der Service `com.sun.star.awt.UnoControlEdit`.

Ein Textfeld kann einzeilig oder mehrzeilig sein, editierbar oder für die Benutzereingabe gesperrt. Für Spezialaufgaben stehen des weiteren spezielle Währungs- und Zahlenfelder sowie Maskenfelder zur Verfügung, die mit einer frei definierbaren Eingabemaske versehen werden können. Diese Steuerelemente basieren auf dem Uno-Service `UnoControlEdit`, so dass ihre programmgesteuerte Handhabung weitgehend identisch ist.

Textfelder stellen folgende Eigenschaften bereit:

- **Text (String)** – aktueller Text.
- **SelectedText (String)** – aktuell markierter Text.
- **Selection (Struct)** – Angaben zur Markierung (Struktur gemäß `com.sun.star.awt.Selection`, mit den Attributen `Min` und `Max`, die den Anfang und das Ende der aktuellen Markierung angeben), nur lesbar.
- **MaxTextLen (short)** – Maximale Länge des Textes, der in das Feld eingegeben werden kann.
- **Editable (Boolean)** – `True` aktiviert die Möglichkeit zur Eingabe von Text, `False` sperrt die Eingabemöglichkeit wieder (die Eigenschaft kann nicht direkt sondern nur über `IsEditable` ausgelesen werden).
- **IsEditable (Boolean)** – der Inhalt des Steuerelementes kann geändert werden, nur lesbar.

Über das zugehörige Model-Objekt stehen darüber hinaus folgende Eigenschaften zur Verfügung:

- **`Model.Align (short)`** – Ausrichtung des Textes (0: linksbündig, 1: zentriert, 2: rechtsbündig).
- **`Model.BackgroundColor (long)`** – Hintergrundfarbe des Steuerelementes.
- **`Model.Border (short)`** – Art des Rahmens (0: kein Rahmen, 1: 3D-Rahmen, 2: Einfacher Rahmen).
- **`Model.EchoChar (String)`** – Echo-Zeichen für Passwortfelder.
- **`Model.FontDescriptor (struct)`** – Struktur mit Angaben zum zu verwendenden Font (gemäß Struktur `com.sun.star.awt.FontDescriptor`).
- **`Model.HardLineBreaks (Boolean)`** – automatische Zeilenumbrüche werden fest in den Text des Steuerelementes eingefügt.
- **`Model.HScroll (Boolean)`** – der Text ist in horizontaler Richtung scrollbar.
- **`Model.MaxTextLen (Short)`** – Maximallänge des Textes, bei der Angabe 0 existiert keine Begrenzung.
- **`Model.MultiLine (Boolean)`** – erlaubt eine mehrzeilige Eingabe.
- **`Model.Printable (Boolean)`** – das Steuerelement ist druckbar.
- **`Model.ReadOnly (Boolean)`** – der Inhalt des Steuerelementes ist schreibgeschützt.
- **`Model.Tabstop (Boolean)`** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **`Model.Text (String)`** – Text des Steuerelementes.
- **`Model.TextColor (Long)`** – Textfarbe des Steuerelementes.
- **`Model.VScroll (Boolean)`** – der Text verfügt über eine vertikale Bildlaufleiste.
- **`Model.HelpText (String)`** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **`Model.HelpURL (String)`** – URL der Online-Hilfe für das betreffende Steuerelement.

Listboxen

Listboxen (Service `com.sun.star.awt.UnoControlListBox`) unterstützen folgende Eigenschaften:

- **`ItemCount (Short)`** – Anzahl der Elemente, nur lesbar.
- **`SelectedItem (String)`** – Text des markierten Eintrages, nur lesbar.
- **`SelectedItems (Array Of Strings)`** – Datenfeld mit den markierten Einträgen, nur lesbar.
- **`SelectedItemPos (Short)`** – Nummer des aktuell markierten Eintrags, nur lesbar.
- **`SelectedItemsPos (Array of Short)`** – Datenfeld mit den Nummern der markierten Einträge (für Listen, die eine Mehrfachauswahl unterstützen), nur lesbar.

- **MultipleMode (Boolean)** – `True` aktiviert die Möglichkeit zur Mehrfachauswahl von Einträgen, `False` sperrt die Mehrfachauswahl (die Eigenschaft kann nicht direkt sondern nur über `IsMultipleMode` ausgelesen werden).
- **IsMultipleMode (Boolean)** – erlaubt eine Mehrfachauswahl innerhalb der Liste , nur lesbar.

Darüber hinaus bieten Listboxen folgende Methoden an:

- **addItem (Item, Pos)** – fügt die in `Item` angegebene Zeichenfolge an der Position `Pos` in die Liste ein.
- **addItems (ItemArray, Pos)** – fügt die in dem Zeichenfolgen-Datenfeld `ItemArray` aufgeführten Einträge in die Liste an der Position `Pos` ein.
- **removeItems (Pos, Count)** – entfernt `Count` Einträge ab der Position `Pos`.
- **selectItem (Item, SelectMode)** – Schaltet die Markierung für das im String `Item` angegebene Element abhängig von der Booleschen Variable `SelectMode` ein oder aus.
- **makeVisible (Pos)** – Scrollt das Listenfeld so, dass der über `Pos` angegebene Eintrag sichtbar wird.

Das Model-Objekt der Listboxen hält folgende Eigenschaften bereit:

- **Model.BackgroundColor (long)** – Hintergrundfarbe des Steuerelementes.
- **Model.Border (short)** – Art des Rahmens (0: kein Rahmen, 1: 3D-Rahmen, 2: Einfacher Rahmen).
- **Model.FontDescriptor (struct)** – Struktur mit Angabe zum zu verwendenden Font (gemäß Struktur `com.sun.star.awt.FontDescriptor`).
- **Model.LineCount (Short)** – Anzahl der Linien des Steuerelementes.
- **Model.MultiSelection (Boolean)** – erlaubt eine Mehrfachauswahl von Einträgen.
- **Model.SelectedItems (Array of Strings)** – Liste der markierten Einträge.
- **Model.StringItemList (Array of Strings)** – Liste sämtlicher Einträge.
- **Model.Printable (Boolean)** – das Steuerelement ist druckbar.
- **Model.ReadOnly (Boolean)** – der Inhalt des Steuerelementes ist schreibgeschützt.
- **Model.Tabstop (Boolean)** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **Model.TextColor (Long)** – Textfarbe des Steuerelementes.
- **Model.HelpText (String)** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **Model.HelpURL (String)** – URL der Online-Hilfe für das betreffende Steuerelement.

Die in VBA vorhandene Möglichkeit, Listeneinträge mit einemnummerischen Zusatzwert zu versehen (`ItemData`) existiert in StarOffice Basic nicht. Soll neben dem Klartext ein zusätzlicher Zahlenwert (etwa eine Datenbank-Id) verwaltet werden, so muss hierfür ein Hilfsdatenfeld aufgebaut werden, das parallel zur Listbox verwaltet wird.

KAPITEL **12**

Formulare

Der Aufbau von StarOffice-Formularen entspricht in vielerlei Hinsicht den im vorstehenden Kapitel besprochenen Dialogen. Es existieren jedoch einige wesentliche Unterschiede:

- Dialoge erscheinen in Form eines eigenen Dialogfensters, das über dem Dokument eingeblendet wird und dieses im Regelfall so lange sperrt, bis die Bearbeitung des Dialoges abgeschlossen ist. Formulare hingegen werden wie Zeichenelemente (Rechtecke, Kreise etc.) direkt im Dokument eingeblendet.
- Für die Erstellung von Dialogen steht ein eigener Dialogeditor zur Verfügung, der sich in der StarOffice Basic Entwicklungsumgebung befindet. Die Erstellung von Formularen erfolgt hingegen über die Werkzeugleiste **Formularfunktionen** direkt innerhalb der betreffenden Dokumente.
- Während die Dialog-Funktionalität in allen StarOffice-Dokumenten zur Verfügung steht, ist der volle Umfang der Formularfunktionen ausschließlich in Text- und Tabellendokumenten verfügbar.
- Die Steuerelemente eines Formulares lassen sich mit einer externen Datenbanktabelle verknüpfen. So ist es möglich, eigene Datenbank-Frontends ohne jeglichen Programmieraufwand zu erstellen. In Dialogen steht diese Funktionalität nicht zur Verfügung.
- Schließlich unterscheiden sich die Steuerelemente von Dialogen und Formularen in einigen Punkten.

Programmierer, die ihre Formulare mit eigenen Methoden zur Ereignisbehandlung ausstatten möchten, seien auf das Kapitel 11 (*Dialoge*) verwiesen. Die dort erklärten Mechanismen sind identisch mit denen für Formulare.

Arbeiten mit Formularen

StarOffice-Formulare können sich zusammensetzen aus Textfeldern, Listboxen, Radio-Buttons und anderen Steuerelementen, die direkt in ein Text- oder Tabellendokument eingefügt werden. Für die Bearbeitung von Formularen steht die Werkzeugleiste **Formularfunktionen** zur Verfügung.

Ein StarOffice-Formular kann einen von zwei Modi haben: den Entwurfsmodus und den Anzeigemodus. Im Entwurfsmodus ist es möglich, die Position von Steuerelementen zu verändern und ihre Eigenschaften über ein Eigenschaftsfenster zu editieren.

Die Umschaltung zwischen den Modi erfolgt ebenfalls über die Werkzeugleiste **Formularfunktionen**.

Ermittlung der Formular-Objekte

StarOffice positioniert die Steuerelemente eines Formulars auf der Zeichenebene. Das eigentliche Formular-Objekt ist über die `Forms`-Auflistung der Zeichenebene erreichbar. In Textdokumenten erfolgt der Zugriff wie folgt:

```
Dim Doc As Object
Dim DrawPage As Object
Dim Form As Object

Doc = StarDesktop.CurrentComponent
DrawPage = Doc.DrawPage
Form = DrawPage.Forms.GetByIndex(0)
```

Die Methode `GetByIndex` liefert dabei das Formular mit der Indexnummer 0 zurück.

Bei Tabellendokumenten bedarf es des Zwischenschrittes über die `Sheets`-Auflistung, da die Zeichenebenen nicht direkt im Dokument, sondern in den einzelnen Tabellenseiten untergebracht sind:

```
Dim Doc As Object
Dim Sheet As Object
Dim DrawPage As Object
Dim Form As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.GetByIndex(0)
DrawPage = Sheet.DrawPage
Form = DrawPage.Forms.GetByIndex(0)
```

Wie die Methode `GetByIndex` bereits nahe legt, kann ein Dokument mehrere Formulare enthalten. Dies ist beispielsweise sinnvoll, wenn die Inhalte verschiedener Datenbanken innerhalb eines Dokumentes visualisiert werden sollen oder wenn eine 1:n-Beziehung einer Datenbank innerhalb eines Formulares angezeigt werden soll. Hierzu besteht auch die Möglichkeit, Unterformulare zu erzeugen.

Die drei Seiten eines Formular-Steuerelementes

Ein Steuerelement eines Formulares hat drei Seiten:

- Zunächst einmal gibt es das *Model* des Steuerelementes. Es stellt das Kernobjekt für den StarOffice Basic-Programmierer bei der Arbeit mit Formular-Steuerelementen dar.
- Das Gegenstück dazu bildet das *View* des Steuerelementes, das Visualisierungsinformationen verwaltet.
- Da Formular-Steuerelemente innerhalb der Dokumente wie ein spezielles Zeichenelement verwaltet werden, existiert des weiteren ein *Shape-Objekt*, das die Zeichenelement-spezifischen Eigenschaften des Steuerelementes widerspiegelt (insbesondere seine Position und Größe).

Zugriff auf das Model von Formular-Steuerelementen

Die Models der Steuerelemente eines Formulars sind über die Methode `GetByName` des `Form`-Objektes verfügbar:

```
Dim Doc As Object
Dim Form As Object
Dim Ctl As Object

Doc = StarDesktop.CurrentComponent
Form = Doc.DrawPage.Forms.GetByIndex(0)
Ctl = Form.getByName("MyListBox")
```

Das Beispiel ermittelt das Model des Steuerelementes `MyListBox`, das sich im ersten Formular des aktuell geöffneten Textdokumentes befindet.

Ist unklar, in welchem Formular sich ein Steuerelement befindet, so besteht die Möglichkeit, alle Formulare nach dem gewünschten Steuerelement zu durchsuchen:

```
Dim Doc As Object
Dim Forms As Object
Dim Form As Object
Dim Ctl As Object
Dim I as Integer

Doc = StarDesktop.CurrentComponent
Forms = Doc.Drawpage.Forms

For I = 0 To Forms.Count - 1
      Form = Forms.GetbyIndex(I)
      If Form.HasByName("MyListBox") Then
            Ctl = Form.GetbyName("MyListBox")
            Exit Function
      End If
Next I
```

Das Beispiel prüft über die Methode `HasByName` in allen Formularen eines Textdokumentes, ob diese ein Steuerelement-Model namens `MyListBox` enthalten. Findet es ein entsprechendes Model, so speichert es eine Referenz darauf in der Variable `Ctl` und bricht die Suche ab.

Zugriff auf das View von Formular-Steuerelementen

Für den Zugriff auf das View eines Formular-Steuerelementes wird zunächst das zugehörige Model benötigt. Mit seiner Hilfe und über den Controller des Dokumentes lässt sich anschließend das View des Steuerelementes ermitteln.

```
Dim Doc As Object
Dim DocCrl As Object
Dim Forms As Object
Dim Form As Object
Dim Ctl As Object
Dim CtlView As Object
Dim I as Integer

Doc = StarDesktop.CurrentComponent
DocCrl = Doc.getCurrentControler()
Forms = Doc.Drawpage.Forms

For I = 0 To Forms.Count - 1
      Form = Forms.GetbyIndex(I)
      If Form.HasByName("MyListBox") Then
            Ctl = Form.GetbyName("MyListBox")
            CtlView = DocCrl.GetControl(Ctl)
            Exit Function
      End If
Next I
```

Der aufgeführte Beispielcode orientiert sich stark an dem weiter oben aufgeführten Beispielcode zur Ermittlung eines Steuerelement-Models. Er verwendet neben dem Dokument-Objekt `Doc` jedoch auch das Dokument-Controller-Objekt `DocCrl`, das auf das aktuelle Dokumentfenster verweist. Mit Hilfe dieses Controller-Objektes und dem Model des Steuerelementes ermittelt es schließlich über die Methode `GetControl` das View (Variable `CtlView`) des Formular-Steuerelementes.

Zugriff auf das Shape-Objekt von Formular-Steuerelementen

Auch für den Zugriff auf die Shape-Objekte eines Steuerelementes führt der Weg über die betreffende Zeichenebene des Dokumentes. Zur Ermittlung eines speziellen Steuerelementes müssen sämtliche Zeichenelemente der Zeichenebene durchsucht werden.

```
Dim Doc As Object
Dim Shape as Object
Dim I as integer

Doc = StarDesktop.CurrentComponent

For i = 0 to Doc.DrawPage.Count - 1
      Shape = Doc.DrawPage(i)

      If HasUnoInterfaces(Shape, _
            "com.sun.star.drawing.XControlShape") Then
            If Shape.Control.Name = "MyListBox" Then
                  Exit Function
            End If
      End If
Next
```

Das Beispiel prüft alle Zeichenelemente nacheinander, ob sie die für Formular-Steuerelemente obligatorische Schnittstelle `com.sun.star.drawing.XControlShape` unterstützen. Ist dies der Fall, so überprüft die Funktion über die Eigenschaft `Control.Name` des weiteren, ob der Name des Steuerelementes `MyListBox` heißt. Ist dies der Fall, so beendet die Funktion die Suche, da sie das gesuchte Zeichenelement des Steuerelementes gefunden hat.

Ermittlung der Größe und Position von Steuerelementen

Wie bereits erwähnt, lässt sich die Größe und Position von Steuerelementen über das zugehörige `Shape`-Objekt ermitteln. Hierzu stellt das Steuerelemente-Shape wie alle anderen `Shape`-Objekte die Eigenschaften `Size` und `Position` zur Verfügung:

- **`Size (struct)`** – Größe des Steuerelementes (Datenstruktur `com.sun.star.awt.Size`).
- **`Position (struct)`** – Position des Steuerelementes (Datenstruktur `com.sun.star.awt.Point`).

Das folgende Beispiel zeigt, wie sich die Position und Größe eines Steuerelementes über das zugehörige `Shape`-Objekt setzen lässt:

```
Dim Shape As Object

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Shape.Size = Size
Shape.Position = Point
```

Der Code setzt voraus, dass das `Shape`-Objekt des Steuerelementes bereits bekannt ist. Ist dies nicht der Fall, so muss es über den weiter oben stehenden Code ermittelt werden.

Formular-Steuerelemente im Detail

Die in Formularen verfügbaren Steuerelemente ähneln denen von Dialogen. Die Auswahl reicht von einfachen Textfeldern über List- und Comboboxen bis hin zu verschiedenen Schaltflächen.

Nachfolgend finden Sie eine Aufstellung der wichtigsten Eigenschaften für Formular-Steuerelemente. Sämtliche Eigenschaften sind Bestandteil der jeweiligen Model-Objekte.

Über die Standardsteuerelemente hinaus steht für Formulare ein Tabellensteuerelement zur Verfügung, das die Einbindung kompletter Datenbanktabellen gestattet. Es wird im Abschnitt *Datenbank-Formulare* behandelt.

Schaltflächen

Das *Model*-Objekt einer Formular-Schaltfläche hält folgende Eigenschaften bereit:

- **`BackgroundColor (long)`** – Hintergrundfarbe.
- **`DefaultButton (Boolean)`** – die Schaltfläche dient als Vorgabewert. Sie reagiert in diesem Fall auch dann auf die Eingabetaste, wenn sie keinen Fokus besitzt.

- **Enabled (Boolean)** – das Steuerelement ist aktivierbar.
- **Tabstop (Boolean)** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **TabIndex (Long)** – Position des Steuerelementes in der Aktivierungsreihenfolge.
- **FontName (String)** – Name der Schriftart.
- **FontHeight (Single)** – Höhe der Zeichen in Punkten (pt).
- **Tag (String)** – Zeichenfolge mit Zusatzinformationen, die in der Schaltfläche für einen programmgesteuerten Zugriff hinterlegt werden kann.
- **TargetURL (String)** – Ziel-URL für Schaltflächen des Typs URL.
- **TargetFrame (String)** – Name des Fensters (bzw. Rahmens), in dem `TargetURL` bei der Aktivierung der Schaltfläche geöffnet werden soll (für Schaltflächen des Typs `URL`).
- **Label (String)** – Beschriftung der Schaltfläche.
- **TextColor (Long)** – Textfarbe des Steuerelementes.
- **HelpText (String)** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **HelpURL (String)** – URL der Online-Hilfe für das betreffende Steuerelement.
- **ButtonType (Enum)** – Aktion, die mit der Schaltfläche verknüpft ist (Vorgabewert aus `com.sun.star.form.FormButtonType`).

Über die Eigenschaft `ButtonType` besteht die Möglichkeit, eine Aktion zu definieren, die beim Druck auf die Schaltfläche automatisch ausgeführt wird. Die zugehörige Konstantengruppe `com.sun.star.form.FormButtonType` hält folgende Werte bereit:

- **PUSH** – Standard-Schaltfläche.
- **SUBMIT** – Abschluss der Formular-Eingabe (insbesondere für HTML-Formulare relevant).
- **RESET** – Setzt sämtliche Werte innerhalb des Formulares auf ihre Ausgangswerte zurück.
- **URL** – Aufruf der in `TargetURL` definierten URL (wird innerhalb des Fensters geöffnet, der über TargetFrame angegeben wurde).

Die in Dialogen vorhandenen Schaltflächentypen **Ok** und **Abbrechen** werden in Formularen nicht unterstüzt.

Radio Buttons

Folgende Eigenschaften eines Radio Buttons sind über dessen *Model*-Objekt verfügbar:

- **Enabled (Boolean)** – das Steuerelement ist aktivierbar.
- **Tabstop (Boolean)** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **TabIndex (Long)** – Position des Steuerelementes in der Aktivierungsreihenfolge.
- **FontName (String)** – Name der Schriftart.
- **FontHeight (Single)** – Höhe der Zeichen in Punkten (pt).

- **`Tag (String)`** – Zeichenfolge mit Zusatzinformationen, die in der Schaltfläche für einen programmgesteuerten Zugriff hinterlegt werden kann.
- **`Label (String)`** – Beschriftung der Schaltfläche.
- **`Printable (Boolean)`** – das Steuerelement ist druckbar.
- **`State (Short)`** – Falls 1, ist die Option aktiviert, ansonsten deaktiviert.
- **`RefValue (String)`** – Zeichenfolge zum Abspeichern von Zusatzinformationen (etwa zum Verwalten von Datensatz-IDs).
- **`TextColor (Long)`** – Textfarbe des Steuerelementes.
- **`HelpText (String)`** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **`HelpURL (String)`** – URL der Online-Hilfe für das betreffende Steuerelement.

Der Mechanismus zum Gruppieren von Radio Buttons unterscheidet sich zwischen den Steuerelementen für Dialoge und Formulare. Während in Dialogen aufeinander folgende Steuerelemente automatisch zu einer Gruppe zusammengefasst werden, erfolgt die Gruppierung in Formularen aufgrund des Namens. Hierzu müssen alle Radio Buttons einer Gruppe den gleichen Namen erhalten. StarOffice fasst die so gruppierten Steuerelemente für den StarOffice Basic-Programmierer in einem Array zusammen, so dass die einzelnen Buttons von einem StarOffice Basic-Programm aus nach wie vor erreichbar bleiben.

Das folgende Beispiel zeigt, wie sich das Model einer Steuerelemente-Gruppe ermitteln lässt.

```
Dim Doc As Object
Dim Forms As Object
Dim Form As Object
Dim Ctl As Object
Dim I as Integer

Doc = StarDesktop.CurrentComponent
Forms = Doc.Drawpage.Forms

For I = 0 To Forms.Count - 1
      Form = Forms.GetbyIndex(I)
      If Form.HasByName("MyOptions") Then
            Ctl = Form. GetGroupbyName("MyOptions")
            Exit Function
      End If
Next I
```

Der Code entspricht im wesentlichen dem weiter oben aufgeführten Beispiel zur Ermittlung eines einfachen Steuerelement-Models. In einer Schleife durchsucht er alle Formulare des aktuell geöffneten (Text-)Dokumentes und prüft über die Methode `HasByName`, ob das betreffende Formular ein Element mit dem gesuchten Namen `MyOptions` enthält. Ist dies der Fall, so erfolgt der Zugriff auf das Model-*Array* über die Methode `GetGroupByName` (anstelle der Methode `GetByName` zur Ermittlung einfacher Models).

Checkboxen

Das *Model*-Objekt einer Formular-Checkbox bietet folgende Eigenschaften an:

- **Enabled (Boolean)** – das Steuerelement ist aktivierbar.
- **Tabstop (Boolean)** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **TabIndex (Long)** – Position des Steuerelementes in der Aktivierungsreihenfolge.
- **FontName (String)** – Name der Schriftart.
- **FontHeight (Single)** – Höhe der Zeichen in Punkten (pt).
- **Tag (String)** – Zeichenfolge mit Zusatzinformationen, die in der Schaltfläche für einen programmgesteuerten Zugriff hinterlegt werden kann.
- **Label (String)** – Beschriftung der Schaltfläche.
- **Printable (Boolean)** – das Steuerelement ist druckbar.
- **State (Short)** – Falls 1, ist die Option aktiviert, ansonsten deaktiviert.
- **RefValue (String)** – Zeichenfolge zum Abspeichern von Zusatzinformationen (etwa zum Verwalten von Datensatz-IDs).
- **TextColor (Long)** – Textfarbe des Steuerelementes.
- **HelpText (String)** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **HelpURL (String)** – URL der Online-Hilfe für das betreffende Steuerelement.

Textfelder

Die *Model*-Objekte von Formular-Textfeldern bieten folgende Eigenschaften an:

- **Align (short)** – Ausrichtung des Textes (0: linksbündig, 1: zentriert, 2: rechtsbündig).
- **BackgroundColor (long)** – Hintergrundfarbe des Steuerelementes.
- **Border (short)** – Art des Rahmens (0: kein Rahmen, 1: 3D-Rahmen, 2: Einfacher Rahmen).
- **EchoChar (String)** – Echo-Zeichen für Passwortfelder.
- **FontName (String)** – Name der Schriftart.
- **FontHeight (Single)** – Höhe der Zeichen in Punkten (pt).
- **HardLineBreaks (Boolean)** – die automatischen Zeilenumbrüche werden fest in den Text des Steuerelementes eingefügt.
- **HScroll (Boolean)** – der Text ist in horizontaler Richtung scrollbar.
- **MaxTextLen (Short)** – Maximallänge des Textes, bei der Angabe 0 existiert keine Begrenzung.
- **MultiLine (Boolean)** – erlaubt eine mehrzeilige Eingabe.
- **Printable (Boolean)** – das Steuerelement ist druckbar.
- **ReadOnly (Boolean)** – der Inhalt des Steuerelementes ist schreibgeschützt.
- **Enabled (Boolean)** – das Steuerelement ist aktivierbar.

- **Tabstop (Boolean)** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **TabIndex (Long)** – Position des Steuerelementes in der Aktivierungsreihenfolge.
- **FontName (String)** – Name der Schriftart.
- **FontHeight (Single)** – Höhe der Zeichen in Punkten (pt).
- **Text (String)** – Text des Steuerelementes.
- **TextColor (Long)** – Textfarbe des Steuerelementes.
- **VScroll (Boolean)** – der Text verfügt über eine vertikale Bildlaufleiste.
- **HelpText (String)** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **HelpURL (String)** – URL der Online-Hilfe für das betreffende Steuerelement.

Listboxen

Das *Model*-Objekt der Formular-Listboxen hält folgende Eigenschaften bereit:

- **BackgroundColor (long)** – Hintergrundfarbe des Steuerelementes.
- **Border (short)** – Art des Rahmens (0: kein Rahmen, 1: 3D-Rahmen, 2: Einfacher Rahmen).
- **FontDescriptor (struct)** – Struktur mit Angabe zum zu verwendenden Font (gemäß Struktur `com.sun.star.awt.FontDescriptor`).
- **LineCount (Short)** – Anzahl der Linien des Steuerelementes.
- **MultiSelection (Boolean)** – erlaubt eine Mehrfachauswahl von Einträgen.
- **SelectedItems (Array of Strings)** – List der markierten Einträge.
- **StringItemList (Array of Strings)** – Liste sämtlicher Einträge.
- **ValueItemList (Array of Variant)** – Liste mit Zusatzinformationen für jeden Eintrag (etwa zum Verwalten von Datensatz-IDs).
- **Printable (Boolean)** – das Steuerelement ist druckbar.
- **ReadOnly (Boolean)** – der Inhalt des Steuerelementes ist schreibgeschützt.
- **Enabled (Boolean)** – das Steuerelement ist aktivierbar.
- **Tabstop (Boolean)** – das Steuerelement ist über die Tabulatortaste erreichbar.
- **TabIndex (Long)** – Position des Steuerelementes in der Aktivierungsreihenfolge.
- **FontName (String)** – Name der Schriftart.
- **FontHeight (Single)** – Höhe der Zeichen in Punkten (pt).
- **Tag (String)** – Zeichenfolge mit Zusatzinformationen, die in der Schaltfläche für einen programmgesteuerten Zugriff hinterlegt werden kann.
- **TextColor (Long)** – Textfarbe des Steuerelementes.

- **`HelpText (String)`** – automatisch eingeblendeter Hilfetext, der angezeigt wird, wenn der Mauszeiger über dem Steuerelement zum Stehen kommt.
- **`HelpURL (String)`** – URL der Online-Hilfe für das betreffende Steuerelement.

> Formular-Listboxen bieten mit der Eigenschaft `ValueItemList` ein Gegenstück zu der VBA-Eigenschaft `ItemData`, über die der Programmierer Zusatzinformationen für einzelne Listeneinträge verwalten kann.

Zusätzlich stehen über das *View*-Objekt der Listbox folgende Methoden zur Verfügung:

- **`addItem (Item, Pos)`** – fügt die in `Item` angegebene Zeichenfolge an der Position `Pos` in die Liste ein.
- **`addItems (ItemArray, Pos)`** – fügt die in dem Zeichenfolgen-Datenfeld `ItemArray` aufgeführten Einträge in die Liste an der Position `Pos` ein
- **`removeItems (Pos, Count)`** – entfernt `Count` Einträge ab der Position `Pos`.
- **`selectItem (Item, SelectMode)`** – Schaltet die Markierung für das im String `Item` angegebene Element abhängig von der Variable `SelectMode` ein oder aus.
- **`makeVisible (Pos)`** – Scrollt das Listfeld so, dass der über `Pos` angegebene Eintrag sichtbar wird.

Datenbank-Formulare

StarOffice-Formulare können direkt mit einer Datenbank verbunden werden. Die so entstehenden Formulare bieten sämtliche Funktionen eines vollwertigen Datenbank-Frontends, ohne dass dafür eigene Programmierarbeit notwendig wäre.

Der Anwender kann die ausgewählten Tabellen beziehungsweise Abfragen durchblättern und durchsuchen sowie Datensätze verändern und neue Datensätze einfügen. StarOffice sorgt automatisch dafür, dass die jeweils benötigten Daten aus der Datenbank abgerufen und eventuelle Änderungen zurück geschrieben werden.

Ein Datenbank-Formular entspricht im wesentlichen einem gewöhnlichen StarOffice-Formular. Über die Standardeigenschaften hinaus müssen im Formular des weiteren folgende datenbankspezifische Eigenschaften gesetzt werden:

- **`DataSourceName (String)`** – Name der Datenquelle (s. Kapitel 10, *Datenbankzugriff*; die Datenquelle muss in StarOffice global angelegt werden).
- **`Command (String)`** – Name der zu verknüpfenden Tabelle, Abfrage oder des auszuführenden SQL-Select-Befehls.
- **`CommandType (Const)`** – Informationen, ob es sich bei `Command` um eine Tabelle, eine Abfrage oder einen SQL-Befehl handelt (Wert aus Aufzählung `com.sun.star.sdb.CommandType`).

Die Aufzählung `com.sun.star.sdb.CommandType` umfasst folgende Werte:

- **`TABLE`** – Tabelle

- **QUERY** - Abfrage
- **COMMAND** – SQL-Befehl

Die Zuordnung der Datenbankfelder zu den einzelnen Steuerelementen erfolgt über ihre Eigenschaft

- **DataField (String)** – Name des verknüpften Datenbankfeldes.

Tabellen

Für die Arbeit mit Datenbanken steht noch ein weiteres Steuerelement zur Verfügung: Das Tabellen-Steuerelement. Es stellt den Inhalt einer kompletten Datenbanktabelle beziehungsweise -abfrage dar. Im einfachsten Fall erfolgt die Verknüpfung eines Tabellen-Steuerelementes mit einer Datenbank über den Formular-Autopiloten, der sämtliche Spalten gemäß der Benutzervorgaben mit den jeweiligen Datenbankfeldern verbindet. Auf eine komplette Beschreibung des zugehörigen API wird an dieser Stelle aus Gründen der Komplexität verzichtet.

KAPITEL **13**

Anhang

VBA Migrations-Tipps

StarOffice 5.x Migrations-Tipps

Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z