



StarOffice 8 Programming Guide for BASIC

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 819-0439-05
2005

Copyright 2004 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, docs.sun.com, AnswerBook, AnswerBook2, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2004 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, et Solaris sont des marques de fabrique ou des marques déposées, de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



041201@10536



Contents

1	Introduction	13
	How This Book Is Organized	13
	About StarOffice Basic	14
	Intended Users of StarOffice Basic	14
	Use of StarOffice Basic	15
	More Information	15
2	The Language of StarOffice Basic	17
	An Overview of a StarOffice Basic Program	17
	Program Lines	18
	Comments	18
	Markers	19
	Working With Variables	20
	Implicit Variable Declaration	20
	Explicit Variable Declaration	20
	Strings	21
	From a Set of ASCII Characters to Unicode	21
	The ASCII Character Set	22
	The ANSI Character Set	22
	Code Pages	22
	Unicode	22
	String Variables	22
	Specification of Explicit Strings	23
	Numbers	23
	Integer Variables	23
	Long Integer Variables	24

Single Variables	24
Double Variables	24
Currency Variables	25
Specification of Explicit Numbers	25
Whole Numbers	25
Decimal Numbers	25
Exponential Writing Style	26
Hexadecimal Values	26
Octal Values	27
True and False	27
Boolean Variables	27
Date and Time Details	28
Date Variables	28
Data Fields	28
Simple Arrays	28
Specified Value for Start Index	29
Multi-Dimensional Data Fields	30
Dynamic Changes in the Dimensions of Data Fields	30
Scope and Life Span of Variables	31
Local Variables	31
Public Domain Variables	32
Global Variables	33
Private Variables	33
Constants	34
Operators	34
Mathematical Operators	34
Logical Operators	35
Comparison Operators	35
Branching	35
If...Then...Else	36
Select...Case	36
Loops	37
For...Next	37
Do...Loop	39
Programming Example: Sorting With Embedded Loops	40
Procedures and Functions	41
Procedures	41
Functions	41

Terminating Procedures and Functions Prematurely	42
Passing Parameters	43
Optional Parameters	44
Recursion	45
Error Handling	45
The On Error Instruction	45
The Resume Command	46
Queries Regarding Error Information	47
Tips for Structured Error Handling	47
3 The Runtime Library of StarOffice Basic	49
Conversion Functions	49
Implicit and Explicit Type Conversions	49
Checking the Content of Variables	51
Strings	53
Working with Sets of Characters	53
Accessing Parts of a String	53
Search and Replace	54
Formatting Strings	55
Date and Time	56
Specification of Date and Time Details within the Program Code	56
Extracting Date and Time Details	57
Retrieving System Date and Time	58
Files and directories	59
Administering Files	59
Searching Through Directories	59
Creating and Deleting Directories	61
Copying, Renaming, Deleting and Checking the Existence of Files	61
Reading and Changing File Properties	62
Writing and Reading Text Files	63
Writing Text Files	63
Reading Text Files	64
Message and Input Boxes	65
Displaying Messages	65
Input Box For Querying Simple Strings	67
Other functions	67
Beep	67
Shell	67

Wait	68
Environ	68
4 Introduction to the StarOffice API	69
Universal Network Objects (UNO)	69
Properties and Methods	70
Properties	70
Real Properties and Imitated Properties	71
Methods	71
Modules, Services and Interfaces	71
Tools for Working with UNO	72
The supportsService Method	72
Debug Properties	72
API Reference	73
An Overview of a Few Central Interfaces	73
Creating Context-Dependent Objects	73
com.sun.star.lang.XMultiServiceFactory Interface	74
Named Access to Subordinate Objects	74
com.sun.star.container.XNameAccess Interface	74
com.sun.star.container.XNameContainer Interface	75
Index-Based Access to Subordinate Objects	76
com.sun.star.container.XIndexAccess Interface	76
com.sun.star.container.XIndexContainer Interface	76
Iterative Access to Subordinate Objects	76
com.sun.star.container.XEnumeration and XenumerationAccess Interfaces	77
5 Working with StarOffice Documents	79
The StarDesktop	79
Basic Information about Documents in StarOffice	80
File Names in URL Notation	80
XML File Format	81
Compression of Files	81
Creating, Opening and Importing Documents	81
Replacing the Content of the Document Window	82
loadComponentFromURL Method Options	83
Creating New Documents	83
Document Objects	84

Saving and Exporting Documents	84
storeAsURL Method Options	85
Printing Documents	86
Templates	87
Details about various formatting options	89
6 Text Documents	91
The Structure of Text Documents	91
Paragraphs and Paragraph Portions	92
Paragraphs	93
Paragraph Portions	94
Formatting	95
Character Properties	96
Paragraph Properties	96
Example: simple HTML export	97
Default values for character and paragraph properties	98
Editing Text Documents	99
The TextCursor	100
Navigating within a Text	100
Formatting Text with TextCursor	102
Retrieving and Modifying Text Contents	103
Inserting Control Codes	103
Searching for Text Portions	104
Example: Similarity Search	106
Replacing Text Portions	107
Example: searching and replacing text with regular expressions	107
Text Documents: More than Just Text	108
Tables	109
Editing Tables	110
Rows	110
Columns	112
Cells	112
Text Frames	113
Text Fields	116
Number of Pages, Words and Characters	117
Current Page	118
Annotations	118
Date / Time	119

Chapter Name / Number	119
Bookmarks	119
7 Spreadsheet Documents	121
The Structure of Table-Based Documents (Spreadsheets)	121
Spreadsheets	122
Creating, Deleting and Renaming Sheets	122
Rows and Columns	123
Inserting and Deleting Rows and Columns	124
Cells	125
Inserting, Deleting, Copying and Moving Cells	127
Formatting	129
Cell Properties	130
Background Color and Shadows	130
Justification	131
Number, Date and Text Format	132
Page Properties	132
Page Background	133
Page Format	133
Page Margin, Border, and Shadow	134
Headers and Footers	135
Changing the Text of Headers and Footers	137
Centering (Spreadsheets Only)	139
Definition of Elements to be Printed (Spreadsheets Only)	139
Editing Spreadsheet Documents Efficiently	140
Cell Ranges	140
Formatting Cell Ranges	140
Computing With Cell Ranges	141
Deleting Cell Contents	141
Searching and Replacing Cell Contents	142
8 Drawings and Presentations	143
The Structure of Drawings	143
Pages	143
Elementary Properties of Drawing Objects	145
Fill Properties	146
Single Color Fills	146

Color Gradient	147
Hatches	148
Bitmaps	149
Transparency	150
Line Properties	150
Text Properties (Drawing Objects)	151
Shadow Properties	153
An Overview of Various Drawing Objects	154
Rectangle Shapes	154
Circles and Ellipses	155
Lines	156
Polypolygon Shapes	156
Graphics	158
Editing Drawing Objects	160
Grouping Objects	160
Rotating and Shearing Drawing Objects	161
Searching and Replacing	162
Presentations	163
Working With Presentations	163
9 Diagrams (Charts)	165
Using Diagrams in Spreadsheets	165
The Structure of Diagrams	167
The Individual Elements of a Diagram	167
Title, Sub-title and Key	167
Background	168
Diagram Walls and Floors	169
Axes	170
First X, Y and Z-Axis	170
Second X and Y-Axis	170
Properties of the Axes	171
Properties of the axis grid	172
Properties of the axis title	172
Example	172
3D Diagrams	173
Stacked Diagrams	173
Diagram Types	174
Line Diagrams	174

	Area Diagrams	174
	Bar Diagrams	174
	Pie Diagrams	175
10	Database Access	177
	SQL: a Query Language	178
	Types of Database Access	178
	Data Sources	179
	Queries	180
	Links with Database Forms	181
	Database Access	182
	Iteration of Tables	183
	Type-Specific Methods for Retrieving Values	184
	The ResultSet Variants	185
	Methods for Navigation in ResultSets	185
	Modifying Data Records	186
11	Dialogs	187
	Working With Dialogs	187
	Creating Dialogs	187
	Closing Dialogs	190
	Closing With OK or Cancel	190
	Closing With the Close Button in the Title Bar	190
	Closing With an Explicit Program Call	190
	Access to Individual Control Elements	190
	Working With the <i>Model</i> of Dialogs and Control Elements	191
	Properties	191
	Name and Title	191
	Position and Size	192
	Focus and Tabulator Sequence	192
	Multi-Page Dialogs	193
	Events	195
	Parameters	197
	Mouse Events	197
	Keyboard Events	198
	Focus Events	199
	Control Element-Specific Events	200

Dialog Control Elements in Detail	200
Buttons	201
Option Buttons	202
Checkboxes	203
Text Fields	203
List Boxes	204

12 Forms 207

Working with Forms	207
Determining Object Forms	208
The Three Aspects of a Control Element Form	208
Accessing the Model of Control Element Forms	209
Accessing the View of Control Element Forms	209
Accessing the Shape Object of Control Element Forms	210
Determining the Size and Position of Control Elements	211
Control Element Forms in Detail	211
Buttons	211
Option Buttons	212
Checkboxes	214
Text Fields	214
List Boxes	215
Database Forms	216
Tables	217

Index

Introduction

This guide provides an introduction to programming with StarOffice™ 8 Basic and indicates the possible applications provided by using StarOffice Basic in StarOffice. To get the most out of this book, you should be familiar with other programming languages.

Extensive examples are provided to help you quickly develop your own StarOffice Basic programs.

How This Book Is Organized

The first three chapters introduce readers to StarOffice Basic:

- Chapter 2, *The Language of StarOffice Basic*
- Chapter 3, *The Runtime Library of StarOffice Basic*
- Chapter 4, *Introduction to the StarOffice API*

These chapters provide an overview of StarOffice Basic and should be read by anyone who intends to write StarOffice Basic programs.

The remaining chapters describe the individual components of the StarOffice API in more detail and can be read selectively as required:

- Chapter 5, *Working with StarOffice Documents*
- Chapter 6, *Text Documents*
- Chapter 7, *Spreadsheet Documents*
- Chapter 8, *Drawings and Presentations*
- Chapter 9, *Diagrams (Charts)*
- Chapter 10, *Database Access*
- Chapter 11, *Dialogs*
- Chapter 12, *Forms*

About StarOffice Basic

The StarOffice Basic programming language has been developed especially for StarOffice and is firmly integrated in the Office package.

As the name suggests, StarOffice Basic is a programming language from the Basic family. Anyone who has previously worked with other Basic languages — in particular with Visual Basic or Visual Basic for Applications (VBA) from Microsoft — will quickly become accustomed to StarOffice Basic. Large sections of the basic constructs of StarOffice Basic are compatible with Visual Basic.

The StarOffice Basic programming language can be divided into four components:

- **The language of StarOffice Basic:** Defines the elementary linguistic constructs, for example, for variable declarations, loops, and functions.
- **The runtime library:** Provides standard functions which have no direct reference to StarOffice, for example, functions for editing numbers, strings, date values, and files.
- **The StarOffice API (Application Programming Interface):** Permits access to StarOffice documents and allows these to be created, saved, modified, and printed.
- **The Dialog Editor:** Creates personal dialog windows and provides scope for the adding of control elements and event handlers.

Note – Compatibility between StarOffice Basic and VBA relates to the StarOffice Basic language as well as the runtime library. The StarOffice API and the Dialog Editor are *not* compatible with VBA (standardizing these interfaces would have made many of the concepts provided in StarOffice impossible).

Intended Users of StarOffice Basic

The scope of application for StarOffice Basic begins where the standard functions of StarOffice end. Routine tasks can therefore be automated in StarOffice Basic, links can be made to other programs — for example to a database server — and complex activities can be performed at the press of a button by using predefined scripts.

StarOffice Basic offers complete access to all StarOffice functions, supports all functions, modifies document types, and provides options for creating personal dialog windows.

Use of StarOffice Basic

StarOffice Basic can be used by any StarOffice user without any additional programs or aids. Even in the standard installation, StarOffice Basic has all the components needed to create its own Basic macros, including:

- **The integrated development environment (IDE)** which provides an editor for creating and testing macros.
- **The interpreter**, which is needed to run StarOffice Basic macros.
- **The interfaces** to various StarOffice applications, which allow for direct access to Office documents.

More Information

The components of the StarOffice API that are discussed in this guide were selected based on their practical benefits for the StarOffice Basic programmer. In general, only parts of the interfaces are discussed. For a more detailed picture, see the API reference which is available on the Internet at:

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

The Developer's Guide describes the StarOffice API in more detail than this guide, but is primarily intended for Java and C++ programmers. Anyone who is already familiar with StarOffice Basic programming can find additional information in the Developer's Guide on StarOffice Basic and StarOffice programming. You can download the Developer's Guide on the Internet from:

<http://api.openoffice.org/DevelopersGuide/DevelopersGuide.html>

Programmers who want to work directly with Java or C++ rather than StarOffice Basic should consult the StarOffice Developer's Guide instead of this guide. StarOffice programming with Java or C++ is a considerably more complex process than programming with StarOffice Basic.

The Language of StarOffice Basic

StarOffice Basic belongs to the family of Basic languages. Many parts of StarOffice Basic are identical to Microsoft Visual Basic for Applications and Microsoft Visual Basic. Anyone who has already worked with these languages can quickly become accustomed to StarOffice Basic.

Programmers of other languages – such as Java, C++, or Delphi – should also find it easy to familiarize themselves with StarOffice Basic. StarOffice Basic is a fully-developed procedural programming language and no longer uses rudimentary control structures, such as `GoTo` and `GoSub`.

You can also benefit from the advantages of object-oriented programming since an interface in StarOffice Basic enables you to use external object libraries. The entire StarOffice API is based on these interfaces, which are described in more detail in the following chapters of this document.

This chapter provides an overview of the key elements and constructs of the StarOffice Basic language, as well as the framework in which applications and libraries are oriented to StarOffice Basic.

An Overview of a StarOffice Basic Program

StarOffice Basic is an interpreter language. Unlike C++ or Turbo Pascal, the StarOffice compiler does not create executable or self-extracting files that are capable of running automatically. Instead, you can execute a StarOffice Basic program by pressing a button. The code is first checked for obvious errors and then executed line by line.

Program Lines

The Basic interpreter's line-oriented execution produces one of the key differences between Basic and other programming languages. Whereas the position of hard line breaks in the source code of Java, C++, or Delphi programs is irrelevant, each line in a Basic program forms a self-contained unit. Function calls, mathematical expressions, and other linguistic elements, such as function and loop headers, must be completed on the same line that they begin on.

If there is not enough space, or if this results in long lines, then several lines can be linked together by adding underscores `_`. The following example shows how four lines of a mathematical expression can be linked:

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

Note – The underscore must always be the last character in a linked line and cannot be followed by a space or a tab, otherwise the code generates an error.

In addition to linking individual lines, StarOffice Basic, you can use colons to divide one line into several sections so that there is enough space for several expressions. The assignments

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

can be written as follows:

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

Comments

In addition to the program code to be executed, a StarOffice Basic program can also contain comments that explain the individual parts of the program and provide important information that can be helpful at a later point.

StarOffice Basic provides two methods for inserting comments in the program code:

- All characters that follow an apostrophe are treated as comments:

```
Dim A ' This is a comment for variable A
```

- The keyword `Rem`, followed by the comment:

```
Rem This comment is introduced by the keyword Rem.
```

A comment usually includes all characters up to the end of the line. StarOffice Basic then interprets the following line as a regular instruction again. If comments cover several lines, each line must be identified as a comment:

```
Dim B      ' This comment for variable B is relatively long
           ' and stretches over several lines. The
           ' comment character must therefore be repeated
           ' in each line.
```

Markers

A StarOffice Basic program can contain dozens, hundreds, or even thousands of *markers*, which are names for variables, constants, functions, and so on. When you select a name for a marker, the following rules apply:

- Markers can only contain Latin letters, numbers, and underscores (_).
- The first character of a marker must be a letter or an underscore.
- Markers cannot contain special characters, such as ä â î ß.
- The maximum length of a marker is 255 characters.
- No distinction is made between uppercase and lowercase characters. The `OneTestVariable` marker, for example, defines the same variable as `onetestVariable` and `ONETESTVARIABLE`.

There is, however, one exception to this rule: a distinction is made between uppercase and lowercase characters for UNO-API constants. More information about UNO is presented in [Chapter 4](#).

Note – The rules for constructing markers are different in StarOffice Basic than in VBA. For example, StarOffice Basic only allows special characters in markers when using Option Compatible, since they can cause problems in international projects.

Here are a few examples of correct and incorrect markers:

```
Surname      ' Correct
Surname5     ' Correct (number 5 is not the first digit)
First Name   ' Incorrect (spaces are not permitted)
DéjàVu      ' Incorrect (letters such as é, à are not permitted)
5Surnames    ' Incorrect (the first character must not be a number)
First,Name   ' Incorrect (commas and full stops are not permitted)
```

Working With Variables

Implicit Variable Declaration

Basic languages are designed to be easy to use. As a result, StarOffice Basic enables the creation of a variable through simple usage and without an explicit declaration. In other words, a variable exists from the moment that you include it in your code. Depending on the variables that are already present, the following example declares up to three new variables:

```
a = b + c
```

Declaring variables implicitly is not good programming practice because it can result in the inadvertent introduction of a new variable through, for example, a typing error. Instead of producing an error message, the interpreter initializes the typing error as a new variable with a value of 0. It can be very difficult to locate errors of this kind in your code.

Explicit Variable Declaration

To prevent errors caused by an implicit declaration of variables, StarOffice Basic provides a switch called:

```
Option Explicit
```

This must be listed in the first program line of each module and ensures that an error message is issued if one of the variables used is not declared. The `Option Explicit` switch should be included in all Basic modules.

In its simplest form, the command for an explicit declaration of a variable is as follows:

```
Dim MyVar
```

This example declares a variable with the name `MyVar` and the type variant. A variant is a universal variable that can record all conceivable values, including strings, whole numbers, floating point figures, and Boolean values. Here are a few examples of Variant variables:

```
MyVar = "Hello World"      ' Assignment of a string
MyVar = 1                  ' Assignment of a whole number
MyVar = 1.0                ' Assignment of a floating point number
MyVar = True               ' Assignment of a Boolean value
```

The variables declared in the previous example can even be used for different variable types in the same program. Although this provides considerable flexibility, it is best to restrict a variable to one variable type. When StarOffice Basic encounters an incorrectly defined variable type in a particular context, an error message is generated.

Use the following style when you make a type-bound variable declaration:

```
Dim MyVar As Integer ' Declaration of a variable of the integer type
```

The variable is declared as an integer type and can record whole number values. You can also use the following style to declare an integer type variable:

```
Dim MyVar% ' Declaration of a variable of the integer type
```

The Dim instruction can record several variable declarations:

```
Dim MyVar1, MyVar2
```

If you want to assign the variables to a permanent type, you must make separate assignments for each variable:

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

If you do not declare the type for a variable, StarOffice Basic assigns the variable a variant type. For example, in the following variable declaration, MyVar1 becomes a variant and MyVar2 becomes an integer:

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

The following sections list the variable types that are available in StarOffice Basic and describe how they can be used and declared.

Strings

Strings, together with numbers, form the most important basic types of StarOffice Basic. A string consists of a sequence of consecutive individual characters. The computer saves the strings internally as a sequence of numbers where each number represents one specific character.

From a Set of ASCII Characters to Unicode

Character sets match characters in a string with a corresponding code (numbers and characters) in a table that describes how the computer is to display the string.

The ASCII Character Set

The ASCII character set is a set of codes that represent numbers, characters, and special symbols by one byte. The 0 to 127 ASCII codes correspond to the alphabet and to common symbols (such as periods, parentheses, and commas), as well as some special screen and printer control codes. The ASCII character set is commonly used as a standard format for transferring text data between computers.

However, this character set does not include a range of special characters used in Europe, such as â, ä, and î, as well as other character formats, such as the Cyrillic alphabet.

The ANSI Character Set

Microsoft based its Windows product on the American National Standards Institute (ANSI) character set, which was gradually extended to include characters that are missing from the ASCII character set.

Code Pages

The ISO 8859 character sets provide an international standard. The first 128 characters of the ISO character set correspond to the ASCII character set. The ISO standard introduces new character sets (*code pages*) so that more languages can be correctly displayed. However, as a result, the same character value can represent different characters in different languages.

Unicode

Unicode increases the length of a character to four bytes and combines different character sets to create a standard to depict as many of the world's languages as possible. Version 2.0 of Unicode is now supported by many programs — including StarOffice and StarOffice Basic.

String Variables

StarOffice Basic saves strings as string variables in Unicode. A string variable can store up to 65535 characters. Internally, StarOffice Basic saves the associated Unicode value for every character. The working memory needed for a string variable depends on the length of the string.

Example declaration of a string variable:

```
Dim Variable As String
```

You can also write this declaration as:

```
Dim Variable$
```

Note – When porting VBA applications, ensure that the maximum allowed string length in StarOffice Basic is observed (65535 characters).

Specification of Explicit Strings

To assign an explicit string to a string variable, enclose the string in quotation marks ("").

```
Dim MyString As String
MyString = " This is a test"
```

To split a string across two lines, add a plus sign at the end of the first line:

```
Dim MyString As String
MyString = "This string is so long that it" + _
           "has been split over two lines."
```

To include a quotation mark (") in a string, enter it twice at the relevant point:

```
Dim MyString As String
MyString = "a ""-quotation mark." ' produces a "-quotation mark
```

Numbers

StarOffice Basic supports five basic types for processing numbers:

- Integer
- Long Integer
- Float
- Double
- Currency

Integer Variables

Integer variables can store any whole number between -32768 and 32767. An integer variable can take up to two bytes of memory. The type declaration symbol for an integer variable is %. Calculations that use integer variables are very fast and are particularly useful for loop counters. If you assign a floating point number to an integer variable, the number is rounded up or down to the next whole number.

Example declarations for integer variables:

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

Long Integer Variables

Long integer variables can store any whole number between -2147483648 and 2147483647 . A long integer variable can take up to four bytes of memory. The type declaration symbol for a long integer is `&`. Calculations with long integer variables are very fast and are particularly useful for loop counters. If you assign a floating point number to a long integer variable, the number is rounded up or down to the next whole number.

Example declarations for long integer variables:

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

Single Variables

Single variables can store any positive or negative floating point number between 3.402823×10^{38} and 1.401298×10^{-45} . A single variable can take up to four bytes of memory. The type declaration symbol for a single variable is `!`.

Originally, single variables were used to reduce the computing time required for the more precise double variables. However, these speed considerations no longer apply, reducing the need for single variables.

Example declarations for single variables:

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

Double Variables

Double variables can store any positive or negative floating point numbers between $1.79769313486232 \times 10^{308}$ and $4.94065645841247 \times 10^{-324}$. A double variable can take up to eight bytes of memory. Double variables are suitable for precise calculations. The type declaration symbol is `#`.

Example declarations of double variables:

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

Currency Variables

Currency variables differ from the other variable types by the way they handle values. The decimal point is fixed and is followed by four decimal places. The variable can contain up to 15 numbers before the decimal point. A currency variable can store any value between -922337203685477.5808 and $+922337203685477.5807$ and takes up to eight bytes of memory. The type declaration symbol for a currency variable is @.

Currency variables are mostly intended for business calculations that yield unforeseeable rounding errors due to the use of floating point numbers.

Example declarations of currency variables:

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

Specification of Explicit Numbers

Numbers can be presented in several ways, for example, in decimal format or in scientific notation, or even with a different base than the decimal system. The following rules apply to numerical characters in StarOffice Basic:

Whole Numbers

The simplest method is to work with whole numbers. They are listed in the source text without a comma separating the thousand figure:

```
Dim A As Integer
Dim B As Float

A = 1210
B = 2438
```

The numbers can be preceded by both a plus (+) or minus (-) sign (with or without a space in between):

```
Dim A As Integer
Dim B As Float

A = + 121
B = - 243
```

Decimal Numbers

When you type a decimal number, use a period (.) as the decimal point. This rule ensures that source texts can be transferred from one country to another without conversion.

```

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

A = 1223.53      ' is rounded
B = - 23446.46  ' is rounded
C = + 3532.76323

```

You can also use plus (+) or minus (-) signs as prefixes for decimal numbers (again with or without spaces).

If a decimal number is assigned to an integer variable, StarOffice Basic rounds the figure up or down.

Exponential Writing Style

StarOffice Basic allows numbers to be specified in the exponential writing style, for example, you can write 1.5e-10 for the number 1.5×10^{-10} (0.00000000015). The letter "e" can be lowercase or uppercase with or without a plus sign (+) as a prefix.

Here are a few correct and incorrect examples of numbers in exponential format:

```

Dim A As Double

A = 1.43E2      ' Correct
A = + 1.43E2    ' Correct (space between plus and basic number)
A = - 1.43E2    ' Correct (space between minus and basic number)
A = 1.43E-2     ' Correct (negative exponent)

A = 1.43E -2   ' Incorrect (spaces not permitted within the number)
A = 1,43E-2    ' Incorrect (commas not permitted as decimal points)
A = 1.43E2.2   ' Incorrect (exponent must be a whole number)

```

Note that in the first and third incorrect examples that no error message is generated even though the variables return incorrect values. The expression

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

is interpreted as 1.43 minus 2, which corresponds to the value -0.57. However, the value 1.43×10^2 (corresponding to 0.0143) was the intended value. With the value

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

StarOffice Basic ignores the part of the exponent after the decimal point and interprets the expression as

```
A = 1.43E2
```

Hexadecimal Values

In the hexadecimal system (base 16 system), a 2-digit number corresponds to precisely one byte. This allows numbers to be handled in a manner which more closely reflects machine architecture. In the hexadecimal system, the numbers 0 to 9 and the letters A

to F are used as numbers. An A stands for the decimal number 10, while the letter F represents the decimal number 15. StarOffice Basic lets you use whole numbered hexadecimal values, so long as they are preceded by &H.

```
Dim A As Long
```

```
A = &HFF ' Hexadecimal value FF, corresponds to the decimal value 255  
A = &H10 ' Hexadecimal value 10, corresponds to the decimal value 16
```

Octal Values

StarOffice Basic also understands the octal system (base 8 system), which uses the numbers 0 to 7. You must use whole numbers that are preceded by &O.

```
Dim A As Long
```

```
A = &O77 ' Octal value 77, corresponds to the decimal value 63  
A = &O10 ' Octal value 10, corresponds to the decimal value 8
```

True and False

Boolean Variables

Boolean variables can only contain one of two values: True or False. They are suitable for binary specifications that can only adopt one of two statuses. A Boolean value is saved internally as a two-byte integer value, where 0 corresponds to the False and any other value to True. There is no type declaration symbol for Boolean variables. The declaration can only be made using the supplement *As Boolean*.

Example declaration of a Boolean variable:

```
Dim Variable As Boolean
```

Date and Time Details

Date Variables

Date variables can contain date and time values. When saving date values, StarOffice Basic uses an internal format that permits comparisons and mathematical operations on date and time values. There is no type declaration symbol for date variables. The declaration can only be made using the supplement *As Date*.

Example declaration of a date variable:

```
Dim Variable As Date
```

Data Fields

In addition to simple variables (*scalars*), StarOffice Basic also supports data fields (*arrays*). A data field contains several variables, which are addressed through an index.

Simple Arrays

An array declaration is similar to that of a simple variable declaration. However, unlike the variable declaration, the array name is followed by parentheses which contain the specifications for the number of elements. The expression

```
Dim MyArray(3)
```

declares an array that has four variables of the variant data type, namely `MyArray(0)`, `MyArray(1)`, `MyArray(2)`, and `MyArray(3)`.

You can also declare type-specific variables in an array. For example, the following line declares an array with four integer variables:

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

In the previous examples, the index for the array always begins with the standard start value of zero. As an alternative, a validity range with start and end values can be specified for the data field declaration. The following example declares a data field that has six integer values and which can be addressed using the indexes 5 to 10:

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

The indexes do not need to be positive values. The following example also shows a correct declaration, but with negative data field limits:

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

It declares an integer data field with 6 values that can be addressed using the indexes -10 to -5.

There are three limits that you must observe when you define data field indexes:

- The smallest possible index is -32768.
- The largest possible index is 32767.
- The maximum number of elements (within a data field dimension) is 16368.

Note – Other limit values sometimes apply for data field indexes in VBA. The same also applies to the maximum number of elements possible per dimension. The values valid there can be found in the relevant VBA documentation.

Specified Value for Start Index

The start index of a data field usually begins with the value 0. Alternatively, you can change the start index for all data field declarations to the value 1 by using the call:

```
Option Base 1
```

The call must be included in the header of a module if you want it to apply to all array declarations in the module. However, this call does not affect the UNO sequences that are defined through the StarOffice API whose index *always* begins with 0. To improve clarity, you should avoid using Option Base 1.

The number of elements in an array is not affected if you use Option Base 1, only the start index changes. The declaration

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

creates 4 integer variables which can be described with the expressions `MyInteger(1)`, `MyInteger(2)`, `MyInteger(3)`, and `MyInteger(4)`.

Note – In StarOffice Basic, the expression `Option Base 1` does not affect the number of elements in an array as it does in VBA. It is, rather, the start index which moves in StarOffice Basic. While the declaration `MyInteger (3)` creates three integer values in VBA with the indexes 1 to 3, the same declaration in StarOffice Basic creates four integer values with the indexes 1 to 4. By using `Option Compatible`, StarOffice Basic behaves like VBA.

Multi-Dimensional Data Fields

In addition to single dimensional data fields, StarOffice Basic also supports work with multi-dimensional data fields. The corresponding dimensions are separated from one another by commas. The example

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

defines an integer array with two dimensions, each with 6 indexes (can be addressed through the indexes 0 to 5). The entire array can record a total of $6 \times 6 = 36$ integer values.

Although you can define hundreds of dimensions in StarOffice Basic Arrays; however, the amount of available memory limits the number of dimensions you can have.

Dynamic Changes in the Dimensions of Data Fields

The previous examples are based on data fields of a specified dimension. You can also define arrays in which the dimension of the data fields dynamically changes. For example, you can define an array to contain all of the words in a text that begin with the letter A. As the number of these words is initially unknown, you need to be able to subsequently change the field limits. To do this in StarOffice Basic, use the following call:

```
ReDim MyArray(10)
```

Note – Unlike VBA, where you can only dimension dynamic arrays by using `Dim MyArray()`, StarOffice Basic lets you change both static and dynamic arrays using `ReDim`.

The following example changes the dimension of the initial array so that it can record 11 or 21 values:

```
Dim MyArray(4) As Integer           ' Declaration with five elements
```

```
' ...
ReDim MyArray(10) As Integer      ' Increase to 11 elements
' ...

ReDim MyArray(20) As Integer      ' Increase to 21 elements
```

When you reset the dimensions of an array, you can use any of the options outlined in the previous sections. This includes declaring multi-dimensional data fields and specifying explicit start and end values. When the dimensions of the data field are changed, all contents are lost. If you want to keep the original values, use the `Preserve` command:

```
Dim MyArray(10) As Integer        ' Defining the initial
                                  ' dimensions
' ...

ReDim Preserve MyArray(20) As Integer  ' Increase in
                                       ' data field, while
                                       ' retaining content
```

When you use `Preserve`, ensure that the number of dimensions and the type of variables remain the same.

Note – Unlike VBA, where only the upper limit of the last dimension of a data field can be changed through `Preserve`, StarOffice Basic lets you change other dimensions as well.

If you use `ReDim` with `Preserve`, you must use the same data type as specified in the original data field declaration.

Scope and Life Span of Variables

A variable in StarOffice Basic has a limited life span and a limited scope from which it can be read and used in other program fragments. The amount of time that a variable is retained, as well as where it can be accessed from, depends on its specified location and type.

Local Variables

Variables that are declared in a function or a procedure are called local variables:

```

Sub Test
  Dim MyInteger As Integer

  ' ...
End Sub

```

Local variables only remain valid as long as the function or the procedure is executing, and then are reset to zero. Each time the function is called, the values generated previously are not available.

To keep the previous values, you must define the variable as `Static`:

```

Sub Test
  Static MyInteger As Integer

  ' ...
End Sub

```

Note – Unlike VBA, StarOffice Basic ensures that the name of a local variable is not used simultaneously as a global and a private variable in the module header. When you port a VBA application to StarOffice Basic, you must change any duplicate variable names.

Public Domain Variables

Public domain variables are defined in the header section of a module by the keyword `Dim`. These variables are available to all of the modules in their library:

Module A:

```

Dim A As Integer

Sub Test
  Flip
  Flop
End Sub

Sub Flip
  A = A + 1
End Sub

```

Module B:

```

Sub Flop
  A = A - 1
End Sub

```

The value of variable `A` is not changed by the `Test` function, but is increased by one in the `Flip` function and decreased by one in the `Flop` function. Both of these changes to the variable are global.

You can also use the keyword `Public` instead of `Dim` to declare a public domain variable:

```
Public A As Integer
```

A public domain variable is only available so long as the associated macro is executing and then the variable is reset.

Global Variables

In terms of their function, global variables are similar to public domain variables, except that their values are retained even after the associated macro has executed. Global variables are declared in the header section of a module using the keyword `Global`:

```
Global A As Integer
```

Private Variables

`Private` variables are only available in the module in which they are defined. Use the keyword `Private` to define the variable:

```
Private MyInteger As Integer
```

If several modules contain a `Private` variable with the same name, StarOffice Basic creates a different variable for each occurrence of the name. In the following example, both module A and B have a `Private` variable called C. The `Test` function first sets the `Private` variable in module A and then the `Private` variable in module B.

Module A:

```
Private C As Integer
```

```
Sub Test
```

```
SetModuleA      ' Sets the variable C from module A
```

```
SetModuleB      ' Sets the variable C from module B
```

```
ShowVarA        ' Shows the variable C from module A (= 10)
```

```
ShowVarB        ' Shows the variable C from module B (= 20)
```

```
End Sub
```

```
Sub SetmoduleeA
```

```
  A = 10
```

```
End Sub
```

```
Sub ShowVarA
```

```
  MsgBox C      ' Shows the variable C from module A.
```

```
End Sub
```

Module B:

```
Private C As Integer

Sub SetModuleB
    A = 20
End Sub

Sub ShowVarB
    MsgBox C          ' Shows the variable C from module B.
End Sub
```

Constants

In StarOffice Basic, use the keyword `Const` to declare a constant.

```
Const A = 10
```

You can also specify the constant type in the declaration:

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

Operators

StarOffice Basic understands common mathematical, logical, and comparison operators.

Mathematical Operators

Mathematical operators can be applied to all numbers types, whereas the `+` operator can also be used to link strings.

- `+` Addition of numbers and date values, linking of strings
- `-` Subtraction of numbers and date values
- `*` Multiplication of numbers
- `/` Division of numbers
- `\` Division of numbers with a whole number result (rounded)
- `^` Raising the power of numbers
- `MOD` module operation (calculation of the rest of a division)

Logical Operators

Logical operators allow you to link elements according to the rules of Boolean algebra. If the operators are applied to Boolean values, the link provides the result required directly. If used in conjunction with integer and long integer values, the linking is done at the bit level.

AND	And linking
OR	Or linking
XOR	Exclusive or linking
NOT	Negation
EQV	Equivalent test (both parts True or False)
IMP	Implication (if the first expression is true, then the second must also be true)

Comparison Operators

Comparison operators can be applied to all elementary variable types (numbers, date details, strings, and Boolean values).

=	Equality of numbers, date values and strings
<>	Inequality of numbers, date values and strings
>	Greater than check for numbers, date values and strings
>=	Greater than or equal to check for numbers, date values and strings
<	Less than check for numbers, date values and strings
<=	Less than or equal to check for numbers, date values and strings

Note – StarOffice Basic does not support the VBA Like comparison operator.

Branching

Use branching statements to restrict the execution of a code block until a particular condition is satisfied.

If...Then...Else

The most common branching statement is the `If` statement as shown in the following example:

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

The `B = 2` assignment only occurs when value of variable `A` is greater than three. A variation of the `If` statement is the `If/Else` clause:

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

In this example, the variable `B` is assigned the value of 2 when `A` is greater than 3, otherwise `B` is assigned the value of 0.

For more complex statements, you can cascade the `If` statement, for example:

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

If the value of variable `A` equals zero, `B` is assigned the value 0. If `A` is less than 3 (but not equal to zero), then `B` is assigned the value 1. In all other instances (that is, if `A` is greater than or equal to 3), `B` is assigned the value 2.

Select...Case

The `Select...Case` instruction is an alternative to the cascaded `If` statement and is used when you need to check a value against various conditions:

```
Select Case DayOfWeek

Case 1:
    NameOfDay = "Sunday"
Case 2:
    NameOfDay = "Monday"
Case 3:
    NameOfDay = "Tuesday"
Case 4:
    NameOfDay = "Wednesday"
Case 5:
```

```
NameOfWeekday = "Thursday"
Case 6:
  NameOfWeekday = "Friday"
Case 7:
  NameOfWeekday = "Saturday"
End Select
```

In this example, the name of a weekday corresponds to a number, so that the `DayOfWeek` variable is assigned the value of 1 for Sunday, 2 for Monday value, and so on.

The `Select` command is not restricted to simple 1:1 assignments — you can also specify comparison operators or lists of expressions in a `Case` branch. The following example lists the most important syntax variants:

```
Select Case Var

Case 1 To 5
  ' ... Var is between the numbers 1 and 5

Case 6, 7, 8
  ' ... Var is 6, 7 or 8

Case Var > 8 And Var < 11
  ' ... Var is greater than 8 and less than 11

Case Else
  ' ... all other instances

End Select
```

Loops

A loop executes a code block for the number of passes that are specified. You can also have loops with an undefined number of passes.

For...Next

The `For . . . Next` loop has a fixed number of passes. The loop counter defines the number of times that the loop is to be executed. In the following example,

```

Dim I
For I = 1 To 10
    ' ... Inner part of loop
Next I

```

variable `I` is the loop counter, with an initial value of 1. The counter is incremented by 1 at the end of each pass. When variable `I` equals 10, the loop stops.

If you want to increment the loop counter by a value other than 1 at the end of each pass, use the `Step` function:

```

Dim I

For I = 1 To 10 Step 0.5

    ' ... Inner part of loop

Next I

```

In the preceding example, the counter is increased by 0.5 at the end of each pass and the loop is executed 19 times.

You can also use negative step values:

```

Dim I

For I = 10 To 1 Step -1

    ' ... Inner part of loop

Next I

```

In this example, the counter begins at 10 and is reduced by 1 at the end of each pass until the counter is 1.

The `Exit For` instruction allows you to exit a `For` loop prematurely. In the following example, the loop is terminated during the fifth pass:

```

Dim I

For I = 1 To 10

    If I = 5 Then
        Exit For
    End If

    ' ... Inner part of loop

Next I

```

Note – The For Each . . . Next loop variant in VBA is not supported in StarOffice Basic.

Do...Loop

The Do . . . Loop is not linked to a fixed number of passes. Instead, the Do . . . Loop is executed until a certain condition is met. There are four variants of the Do . . . Loop (in the following examples, A > 10 represents any condition):

1. The Do While . . . Loop variant

```
Do While A > 10
  ' ... loop body
Loop
```

checks whether the condition is still satisfied before every pass and only then executes the loop.

2. The Do Until . . . Loop variant

```
Do Until A > 10
  ' ... loop body
Loop
```

executes the loop until the condition is *no longer* satisfied.

3. The Do . . . Loop While variant

```
Do
  ' ... loop body
Loop While A > 10
```

only checks the condition after the first loop pass and terminates if this condition is *satisfied*.

4. The Do . . . Loop Until variant

```
Do
  ' ... loop body
Loop Until A > 10
```

also checks its condition after the first pass, but undertakes the loop until the condition is *no longer* satisfied.

As in the For . . . Next loop, the Do . . . Loop also provides a terminate command. The Exit Do command can exit at loop at any point within the loop.

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

```
End If

' ... loop body
While A > 10
```

Programming Example: Sorting With Embedded Loops

There are many ways to use loops, for example, to search lists, return values, or execute complex mathematical tasks. The following example is an algorithm that uses to loops to sort a list by names.

```
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
```

The values are interchanged as pairs several times until they are finally sorted in ascending order. Like bubbles, the variables gradually migrate to the right position. For this reason, this algorithm is also known as a *Bubble Sort*.

Procedures and Functions

Procedures and functions form pivotal points in the structure of a program. They provide the framework for dividing a complex problem into various sub-tasks.

Procedures

A *procedure* executes an action without providing an explicit value. Its syntax is

```
Sub Test
    ' ... here is the actual code of the procedure
End Sub
```

The example defines a procedure called `Test` that contains code that can be accessed from any point in the program. The call is made by entering the procedure name at the relevant point of the program:

```
Test
```

Functions

A *function*, just like a procedure, combines a block of programs to be executed into one logical unit. However, unlike a procedure, a function provides a return value.

```
Function Test
    ' ... here is the actual code of the function
    Test = 123
End Function
```

The return value is assigned using simple assignment. The assignment does not need to be placed at the end of the function, but can be made anywhere in the function.

The preceding function can be called within a program as follows:

```
Dim A
A = Test
```

The code defines a variable `A` and assigns the result of the `Test` function to it.

The return value can be overwritten several times within the function. As with classic variable assignment, the function in this example returns the value that was last assigned to it.

```
Function Test  
  
    Test = 12  
  
    ' ...  
  
    Test = 123  
  
End Function
```

In this example, the return value of the function is 123.

If an assignment is stopped, the function returns a zero value (number 0 for numerical values and a blank for strings).

The return value of a function can be any type. The type is declared in the same way as a variable declaration:

```
Function Test As Integer  
  
    ' ... here is the actual code of the function  
  
End Function
```

If the specification of an explicit value is stopped, the type of the return value is assigned as variant.

Terminating Procedures and Functions Prematurely

In StarOffice Basic, you can use the `Exit Sub` and `Exit Function` commands to terminate a procedure or function prematurely, for example, for error handling. These commands stop the procedure or function and return the program to the point at which the procedure or function was called up.

The following example shows a procedure which terminates implementation when the `ErrorOccured` variable has the value `True`.

```
Sub Test  
    Dim ErrorOccured As Boolean  
  
    ' ...  
  
    If ErrorOccured Then  
        Exit Sub  
    End If
```

```
' ...  
End Sub
```

Passing Parameters

Functions and procedures can receive one or more parameters. Essential parameters must be enclosed in parentheses after the function or procedure names. The example

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

defines a procedure that expects an integer value A and a string B as parameters.

Parameters are normally passed by *Reference* in StarOffice Basic. Changes made to the variables are retained when the procedure or function is exited:

```
Sub Test  
    Dim A As Integer  
    A = 10  
    ChangeValue(A)  
    ' The parameter A now has the value 20  
End Sub  
Sub ChangeValue(TheValue As Integer)  
    TheValue = 20  
End Sub
```

In this example, the value A that is defined in the Test function is passed as a parameter to the ChangeValue function. The value is then changed to 20 and passed to TheValue, which is retained when the function is exited.

You can also pass a parameter as a *value* if you do not want subsequent changes to the parameter to affect the value that is originally passed. To specify that a parameter is to be passed as a value, ensure that the ByVal keyword precedes the variable declaration in the function header.

In the preceding example, if we replace the ChangeValue function with the

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

function, then the superordinate variable A remains unaffected by this change. After the call for the ChangeValue function, variable A retains the value 10.

Note – The method for passing parameters to procedures and functions in StarOffice Basic is virtually identical to that in VBA. By default, the parameters are passed by reference. To pass parameters as values, use the `ByVal` keyword. In VBA, you can also use the keyword `ByRef` to force a parameter to be passed by reference. StarOffice Basic does not support this keyword because this is already the default procedure in StarOffice Basic.

Optional Parameters

Functions and procedures can only be called up if all the necessary parameters are passed during the call.

StarOffice Basic lets you define parameters as *optional*, that is, if the corresponding values are not included in a call, StarOffice Basic passes an empty parameter. In the example

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

the A parameter is obligatory, whereas the B parameter is optional.

The `IsMissing` function checks whether a parameter has been passed or is left out.

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

  ' Check whether B parameter is actually present
  If Not IsMissing (B) Then
    B_Local = B      ' B parameter present
  Else
    B_Local = 0     ' B parameter missing -> default value 0
  End If

  ' ... Start the actual function

End Sub
```

The example first tests whether the B parameter has been passed and, if necessary, passes the same parameter to the internal `B_Local` variable. If the corresponding parameter is not present, then a default value (in this instance, the value 0) is passed to `B_Local` rather than the passed parameter.

Note – The `ParamArray` keyword present in VBA is not supported in StarOffice Basic.

Recursion

Recursion is now possible in StarOffice Basic. A recursive procedure or function is one that has the ability to call itself until it detects that some base condition has been satisfied. When the function is called with the base condition, a result is returned.

The following example uses a recursive function to calculate the factorial of the numbers 42, -42, and 3.14:

```
Sub Main
  MsgBox CalculateFactorial( 42 )      ' Displays 1,40500611775288E+51
  MsgBox CalculateFactorial( -42 )    ' Displays "Invalid number for factorial!"
  MsgBox CalculateFactorial( 3.14 )   ' Displays "Invalid number for factorial!"
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
```

The example returns the factorial of the number 42 by recursively calling the `CalculateFactorial` function until it reaches the base condition of $0! = 1$.

Note – The recursion levels are set at different levels based on the software platform. For Windows the recursion level is 5800. For Solaris and Linux, an evaluation of the stacksize is performed and the recursion level is calculated.

Error Handling

Correct handling of error situations is one of the most time-consuming tasks of programming. StarOffice Basic provides a range of tools for simplifying error handling.

The On Error Instruction

The `On Error` instruction is the key to any error handling:

```

Sub Test
  On Error Goto ErrorHandler

  ' ... undertake task during which an error may occur

Exit Sub

ErrorHandler:

  ' ... individual code for error handling

End Sub

```

The `On Error Goto ErrorHandler` line defines how StarOffice Basic proceeds in the event of an error. The `Goto ErrorHandler` ensures that StarOffice Basic exits the current program line and then executes the `ErrorHandler:` code.

The Resume Command

The `Resume Next` command continues the program from the line that follows where the error occurred in the program after the code in the error handler has been executed:

```

ErrorHandler:

  ' ... individual code for error handling

  Resume Next

```

Use the `Resume Proceed` command to specify a jump point for continuing the program after error handling:

```

ErrorHandler:

  ' ... individual code for error handling

  Resume Proceed

Proceed:

  ' ... the program continues here after the error

```

To continue a program without an error message when an error occurs, use the following format:

```

Sub Test
  On Error Resume Next

  ' ... perform task during which an error may occur

End Sub

```

Use the `On Error Resume Next` command with caution as its effect is global. For more information, see [“Tips for Structured Error Handling”](#) on page 47.

Queries Regarding Error Information

In error handling, it is useful to have a description of the error and to know where and why the error occurred:

- The `Err` variable contains the number of errors that has occurred.
- The `Error$` variable contains a description of the error.
- The `Erl` variable contains the line number where the error occurred.

The call

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

shows how the error information can be displayed in a message window.

Note – Whereas VBA summarizes the error messages in a statistical object called `Err`, StarOffice Basic provides the `Err`, `Error$`, and `Erl` variables.

The status information remains valid until the program encounters a `Resume` or `On Error` command, whereupon the information is reset.

Note – In VBA, the `Err.Clear` method of the `Err` object resets the error status after an error occurs. In StarOffice Basic, this is accomplished with the `On Error` or `Resume` commands.

Tips for Structured Error Handling

Both the definition command, `On Error`, and the return command, `Resume`, are variants of the `Goto` construct.

If you want to cleanly structure your code to prevent generating errors when you use this construct, you should not use jump commands without monitoring them.

Care should be taken when you use the `On Error Resume Next` command as this dismisses all open error messages.

The best solution is to use only one approach for error handling within a program - keep error handling separate from the actual program code and do not jump back to the original code after the error occurs.

The following is an example of an error handling procedure:

```
Sub Example

    ' Define error handler at the start of the function
    On Error Goto ErrorHandler

    ' ... Here is the actual program code

    ' Deactivate error handling
    On Error Goto 0

    ' End of regular program implementation
Exit Sub

' Start point of error handling
ErrorHandler:

    ' Check whether error was expected
    If Err = ExpectedErrorNo Then
        ' ... Process error
    Else
        ' ... Warning of unexpected error
    End If

    On Error Goto 0          ' Deactivate error handling
End Sub
```

This procedure begins with the definition of an error handler, followed by the actual program code. At the end of the program code, the error handling is deactivated by the `On Error Goto 0` call and the procedure implementation is ended by the `Exit Sub` command (not to be confused with `End Sub`).

The example first checks if the error number corresponds to the expected number (as stored in the imaginary `ExpectedErrorNo` constant) and then handles the error accordingly. If another error occurs, the system outputs a warning. It is important to check the error number so that unanticipated errors can be detected.

The `On Error Goto 0` call at the end of the code resets the status information of the error (the error code in the `Err` system variables) so that an error occurring at a later date can be clearly recognized.

The Runtime Library of StarOffice Basic

The following sections present the central functions of the runtime library.

Conversion Functions

In many situations, circumstances arise in which a variable of one type has to be changed into a variable of another type.

Implicit and Explicit Type Conversions

The easiest way to change a variable from one type to another is to use an assignment.

```
Dim A As String
Dim B As Integer

B = 101
A = B
```

In this example, variable A is a string, and variable B is an integer. StarOffice Basic ensures that variable B is converted to a string during assignment to variable A. This conversion is much more elaborate than it appears: the integer B remains in the working memory in the form of a two-byte long number. A, on the other hand, is a string, and the computer saves a one- or two-byte long value for each character (each number). Therefore, before copying the content from B to A, B has to be converted into A's internal format.

Unlike most other programming languages, Basic performs type conversion automatically. However, this may have fatal consequences. Upon closer inspection, the following code sequence

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

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

which at first glance seems straightforward, ultimately proves to be something of a trap. The Basic interpreter first calculates the result of the addition process and then converts this into a string, which, as its result, produces the string 2.

If, on the other hand, the Basic interpreter first converts the start values B and C into a string and applies the plus operator to the result, it produces the string 11.

The same applies when using variant variables:

```
Dim A
Dim B
Dim C
```

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

Since variant variables may contain both numbers and strings, it is unclear whether variable A is assigned the number 2 or the string 11.

The error sources noted for implicit type conversions can only be avoided by careful programming; for example, by not using the variant data type.

To avoid other errors resulting from implicit type conversions, StarOffice Basic offers a range of conversion functions, which you can use to define when the data type of an operation should be converted:

- **CStr(Var)** - converts any data type into a string.
- **CInt(Var)** - converts any data types into an integer value.
- **CLng(Var)** - converts any data types into a long value.
- **CSng(Var)** - converts any data types into a single value.
- **CDbl(Var)** - converts any data types into a double value.
- **CBool(Var)** - converts any data types into a Boolean value.
- **CDate(Var)** - converts any data types into a date value.

You can use these conversion functions to define how StarOffice Basic should perform these type conversion operations:

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

```
B = 1
C = 1
```

```

A = CStr(B + C)      ' B and C are added together first, then
                    ' converted (produces the number 2)
A = CStr(B) + CStr(C) ' B and C are converted into a string, then
                    ' combined (produces string "11")

```

During the first addition in the example, StarOffice Basic first adds the integer variables and then converts the result into a chain of characters. A is assigned the string 2. In the second instance, the integer variables are first converted into two strings and then linked with one another by means of the assignment. A is therefore assigned the string 11.

The numerical CSng and Cdbl conversion functions also accept decimal numbers. The symbol defined in the corresponding country-specific settings must be used as the decimal point symbol. Conversely, the CStr methods use the currently selected country-specific settings when formatting numbers, dates and time details.

The Val function is different from the CSng, Cdbl and CStr methods. It converts a string into a number; however it always expects a period to be used as the decimal point symbol.

```

Dim A As String
Dim B As Double

A = "2.22"
B = Val(A)      ' Is converted correctly regardless of the
                ' country-specific settings

```

Checking the Content of Variables

In some instances, the date cannot be converted:

```

Dim A As String
Dim B As Date

A = "test"
B = A      ' Creates error message

```

In the example shown, the assignment of the test string to a date variable makes no sense, so the Basic interpreter reports an error. The same applies when attempting to assign a string to a Boolean variable:

```

Dim A As String
Dim B As Boolean

A = "test"
B = A      ' Creates error message

```

Again, the basic interpreter reports an error.

These error messages can be avoided by checking the program before an assignment, in order to establish whether the content of the variable to be assigned matches the type of the target variable. StarOffice Basic provides the following test functions for this purpose:

- **IsNumeric(Value)** - checks whether a value is a number.
- **IsDate(Value)** - checks whether a value is a date.
- **IsArray(Value)** - checks whether a value is an array.

These functions are especially useful when querying user input. For example, you can check whether a user has typed a valid number or date.

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

In the previous example, if the `UserInput` variable contains a valid numerical value, then this is assigned to the `ValidInput` variable. If `UserInput` does not contain a valid number, `ValidInput` is assigned the value 0 and an error message is returned.

While test functions exist for checking numbers, date details and arrays in StarOffice Basic, a corresponding function for checking Boolean values does not exist. The functionality can, however, be imitated by using the `IsBoolean` function:

```
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
```

The `IsBoolean` function defines an internal `Dummy` help variable of the Boolean type and tries to assign this to the transferred value. If assignment is successful, the function returns `True`. If it fails, a runtime error is produced, which intercepts the test function to return an error.

Note – If a string in StarOffice Basic contains a non-numerical value and if this is assigned to a number, StarOffice Basic does not produce an error message, but transfers the value 0 to the variable. This procedure differs from VBA. There, an error is triggered and program implementation terminated if a corresponding assignment is executed.

Strings

Working with Sets of Characters

When administering strings, StarOffice Basic uses the set of Unicode characters. The `Asc` and `Chr` functions allow the Unicode value belonging to a character to be established and/or the corresponding character to be found for a Unicode value. The following expressions assign the various Unicode values to the code variable:

```
Code = Asc("A")           ' Latin letter A (Unicode-value 65)
Code = Asc("-")           ' Euro character (Unicode-value 8364)
Code = Asc("??")          ' Cyrillic letter ?? (Unicode-value 1083)
```

Conversely, the expression

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

ensures that the `MyString` string is initialized with the value of the number 13, which stands for a hard line break.

The `Chr` command is often used in Basic languages to insert control characters in a string. The assignment

```
MyString = Chr(9) + "This is a test" + Chr(13)
```

therefore ensures that the text is preceded by a tab character (Unicode-value 9) and that a hard line break (Unicode-value 13) is added after the text.

Accessing Parts of a String

StarOffice Basic provides four functions that return partial strings:

- **Left(MyString, Length)** - returns the first `Length` characters of `MyString`.
- **Right(MyString, Length)** - returns the last `Length` characters of `MyString`.
- **Mid(MyString, Start, Length)**- returns first `Length` characters of `MyString` as of the `Start` position.
- **Len(MyString)**- returns the number of characters in `MyString`.

Here are a few example calls for the named functions:

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

```

MyString = "This is a small test"

MyResult = Left(MyString,5)      ' Provides the string "This "
MyResult = Right(MyString, 5)   ' Provides the string " test"
MyResult = Mid(MyString, 8, 5)  ' Provides the string " a sm"
MyLen = Len(MyString)           ' Provides the value 21

```

Search and Replace

StarOffice Basic provides the `InStr` function for searching for a partial string within another string:

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

The `SearchString` parameter specifies the string to be searched for within `MyString`. The function returns a number that contains the position at which the `SearchString` first appears within `MyString`. If you want to find other matches for the string, the function also provides the opportunity to specify an optional start position from which StarOffice Basic begins the search. In this case, the syntax of the function is:

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

In the previous examples, `InStr` ignores uppercase and lowercase characters. To change the search so that `InStr` is case sensitive, add the parameter 0, as shown in the following example:

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

Using the previous functions for editing strings, programmers can search for and replace one string in another string:

```

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, Search, Source)
            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

```

The function searches through the transferred `Search` string in a loop by means of `InStr` in the original term `Source`. If it finds the search term, it takes the part before the expression and writes it to the `Result` return buffer. It adds the new `Part` section at the point of the search term `Search`. If no more matches are found for the search term, the function establishes the part of the string still remaining and adds this to the return buffer. It returns the string produced in this way as the result of the replacement process.

Since replacing parts of character sequences is one of the most frequently used functions, the `Mid` function in StarOffice Basic has been extended so that this task is performed automatically. The following example

```

Dim MyString As String

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

```

replaces three characters with the string `is` from the sixth position of the `MyString` string.

Formatting Strings

The `Format` function formats numbers as a string. To do this, the function expects a `Format` expression to be specified, which is then used as the template for formatting the numbers. Each place holder within the template ensures that this item is formatted correspondingly in the output value. The five most important place holders within a template are the *zero* (0), *pound sign* (#), *period* (.), *comma* (,) and *dollar sign* (\$) characters.

The *zero* character within the template ensures that a number is always placed at the corresponding point. If a number is not provided, 0 is displayed in its place.

A *period* stands for the decimal point symbol defined by the operating system in the country-specific settings.

The example below shows how the *zero* and *period* characters can define the digits after the decimal point in an expression:

```

MyFormat = "0.00"

MyString = Format(-1579.8, MyFormat) ' Provides "-1579,80"
MyString = Format(1579.8, MyFormat) ' Provides "1579,80"
MyString = Format(0.4, MyFormat) ' Provides "0,40"

```

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

In the same way, zeros can be added in front of a number to achieve the desired length:

```
MyFormat = "0000.00"
```

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

A *comma* represents the character that the operating system uses for a thousands separator, and the *pound sign* stands for a digit or place that is only displayed if it is required by the input string.

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

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

In place of the *dollar sign* place holder, the `Format` function displays the relevant currency symbol defined by the system:

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

```
MyString = Format(-1579.8, MyFormat) ' Provides "-1.579,80 —"  
MyString = Format(1579.8, MyFormat) ' Provides "1.579,80 —"  
MyString = Format(0.4, MyFormat) ' Provides "0,40 —"  
MyString = Format(0.434, MyFormat) ' Provides "0,43 —"
```

Note – The format instructions used in VBA for formatting date and time details are not supported in StarOffice Basic.

Date and Time

StarOffice Basic provides the `Date` data type, which saves the date and time details in binary format.

Specification of Date and Time Details within the Program Code

You can assign a date to a date variable through the assignment of a simple string:

```
Dim MyDate As Date
```

```
MyDate = "1.1.2002"
```

This assignment can function properly because StarOffice Basic automatically converts the date value defined as a string into a date variable. This type of assignment, however, can cause errors, date and time values are defined and displayed differently in different countries.

Since StarOffice Basic uses the country-specific settings of the operating system when converting a string into a date value, the expression shown previously only functions correctly if the country-specific settings match the string expression.

To avoid this problem, the `DateSerial` function should be used to assign a fixed value to a date variable:

```
Dim MyVar As Date
```

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

The function parameter must be in the sequence: year, month, day. The function ensures that the variable is actually assigned the correct value regardless of the country-specific settings

The `TimeSerial` function formats time details in the same way that the `DateSerial` function formats dates:

```
Dim MyVar As Date
```

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

Their parameters should be specified in the sequence: hours, minutes, seconds.

Extracting Date and Time Details

The following functions form the counterpart to the `DateSerial` and `TimeSerial` functions:

- **Day(MyDate)** - returns the day of the month from `MyDate`
- **Month(MyDate)** - returns the month from `MyDate`
- **Year(MyDate)** - returns the year from `MyDate`
- **Weekday(MyDate)** - returns the number of the weekday from `MyDate`
- **Hour(MyTime)** - returns the hours from `MyTime`
- **Minute(MyTime)** - returns the minutes from `MyTime`
- **Second(MyTime)** - returns the seconds from `MyTime`

These functions extract the date or time sections from a specified `Date` variable. The example

```
Dim MyDate As Date
```

```
' ... Initialization of MyDate
```

```

If Year(MyDate) = 2003 Then
    ' ... Specified date is in the year 2003
End If

```

checks whether the date saved in `MyDate` is in the year 2003. In the same way, the example

```

Dim MyTime As Date
' ... Initialization of MyTime
If Hour(MyTime) >= 12 And Hour(MyTime) < 14 Then
    ' ... Specified time is between 12 and 14 hours
End If

```

checks whether `MyTime` is between 12 and 14 hours.

The `Weekday` function returns the number of the weekday for the transferred date:

```

Dim MyDate As Date
Dim MyWeekday As String
' ... initialize MyDate
Select Case WeekDay(MyDate)
case 1
    MyWeekday = "Sunday"
case 2
    MyWeekday = "Monday"
case 3
    MyWeekday = "Tuesday"
case 4
    MyWeekday = "Wednesday"
case 5
    MyWeekday = "Thursday"
case 6
    MyWeekday = "Friday"
case 7
    MyWeekday = "Saturday"
End Select

```

Note – Sunday is considered the first day of the week.

Retrieving System Date and Time

The following functions are available in StarOffice Basic to retrieve the system time and system date:

- **Date** - returns the present date
- **Time** - returns the present time
- **Now** - returns the present point in time (date and time as combined value)

Files and directories

Working with files is one of the basic tasks of an application. The StarOffice API provides you with a whole range of objects with which you can create, open and modify Office documents. These are presented in detail in [Chapter 4](#). Regardless of this, in some instances you will have to directly access the file system, search through directories or edit text files. The runtime library from StarOffice Basic provides several fundamental functions for these tasks.

Note – Some DOS-specific file and directory functions are no longer provided in StarOffice 8, or their function is only limited. For example, support for the `ChDir`, `ChDrive` and `CurDir` functions is not provided. Some DOS-specific properties are no longer used in functions that expect file properties as parameters (for example, to differentiate from concealed files and system files). This change became necessary to ensure the greatest possible level of platform independence for StarOffice.

Administering Files

Searching Through Directories

The `Dir` function in StarOffice Basic is responsible for searching through directories for files and sub-directories. When first requested, a string containing the path of the directories to be searched must be assigned to `Dir` as its first parameter. The second parameter of `Dir` specifies the file or directory to be searched for. StarOffice Basic returns the name of the first directory entry found. To retrieve the next entry, the `Dir` function should be requested without parameters. If the `Dir` function finds no more entries, it returns an empty string.

The following example shows how the `Dir` function can be used to request all files located in one directory. The procedure saves the individual file names in the `AllFiles` variable and then displays this in a message box.

```
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

```

The 0 used as the second parameter in the `Dir` function ensures that `Dir` only returns the names of files and directories are ignored. The following parameters can be specified here:

- 0 : returns normal files
- 16 : sub-directories

The following example is virtually the same as the preceding example, but the `Dir` function transfers the value 16 as a parameter, which returns the sub-directories of a folder rather than the file names.

```

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

```

Note – When requested in StarOffice Basic, the `Dir` function, using the parameter 16, only returns the sub-directories of a folder. In VBA, the function also returns the names of the standard files so that further checking is needed to retrieve the directories only. When using the `CompatibilityMode (true)` function, StarOffice Basic behaves like VBA and the `Dir` function, using parameter 16, returns sub-directories and standard files.

Note – The options provided in VBA for searching through directories specifically for files with the *concealed*, *system file*, *archived*, and *volume name* properties does not exist in StarOffice Basic because the corresponding file system functions are not available on all operating systems.

Note – The path specifications listed in `Dir` may use the `*` and `?` place holders in both VBA and StarOffice Basic. In StarOffice Basic, the `*` place holder may however only be the last character of a file name and/or file extension, which is not the case in VBA.

Creating and Deleting Directories

StarOffice Basic provides the `MkDir` function for creating directories.

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

This function creates directories and sub-directories. All directories needed within a hierarchy are also created, if required. For example, if only the `C:\SubDir1` directory exists, then a call

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

creates both the `C:\SubDir1\SubDir2` directory and the `C:\SubDir1\SubDir2\SubDir3` directory.

The `RmDir` function deletes directories.

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

If the directory contains sub-directories or files, these are *also deleted*. You should therefore be careful when using `RmDir`.

Note – In VBA, the `MkDir` and `RmDir` functions only relate to the current directory. In StarOffice Basic on the other hand, `MkDir` and `RmDir` can be used to create or delete levels of directories.

Note – In VBA, `RmDir` produces an error message if a directory contains a file. In StarOffice Basic, the directory *and all its files* are deleted. If you use the `CompatibilityMode (true)` function, StarOffice Basic will behave like VBA.

Copying, Renaming, Deleting and Checking the Existence of Files

The call

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

creates a copy of the `Source` file under the name of `Destination`.

With the help of the function

```
Name OldName As NewName
```

you can rename the OldName file with NewName. The As keyword syntax, and the fact that a comma is not used, goes back to the roots of the Basic language.

The call

```
Kill (Filename)
```

deletes the Filename file. If you want to delete directory (including its files) use the Rmdir function.

The FileExists function can be used to check whether a file exists:

```
If FileExists (Filename) Then  
    MsgBox "file exists."  
End If
```

Reading and Changing File Properties

When working with files, it is sometimes important to be able to establish the file properties, the time the file was last changed and the length of the file.

The call

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

returns some properties about a file. The return value is provided as a bit mask in which the following values are possible:

- 1 : read-only file
- 16 : name of a directory

The example

```
Dim FileMask As Integer  
Dim FileDescription As String  
  
FileMask = GetAttr("test.txt")  
If (FileMask AND 1) > 0 Then  
    FileDescription = FileDescription & " read-only "  
End IF  
  
If (FileMask AND 16) > 0 Then  
    FileDescription = FileDescription & " directory "  
End IF  
  
If FileDescription = "" Then  
    FileDescription = " normal "  
End IF
```

MsgBox FileDescription

determines the bit mask of the `test.txt` file and checks whether this is read-only whether it is a directory. If neither of these apply, `FileDescription` is assigned the "normal" string.

Note – The flags used in VBA for querying the *concealed*, *system file*, *archived* and *volume name* file properties are not supported in StarOffice Basic because these are Windows-specific and are not or are only partially available on other operating systems.

The `SetAttr` function permits the properties of a file to be changed. The call

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

can therefore be used to provide a file with read-only status. An existing read-only status can be deleted with the following call:

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

The date and time of the last amendment to a file are provided by the `FileDateTime` function. The date is formatted here in accordance with the country-specific settings used on the system.

```
FileDateTime("test.txt") ' Provides date and time of the last file amendment.
```

The `FileLen` function determines the length of a file in bytes (as long integer value).

```
FileLen("test.txt") ' Provides the length of the file in bytes
```

Writing and Reading Text Files

StarOffice Basic provides a whole range of methods for reading and writing files. The following explanations relate to working with text files (*not* text documents).

Writing Text Files

Before a text file is accessed, it must first be opened. To do this, a free *file handle* is needed, which clearly identifies the file for subsequent file access.

The `FreeFile` function is used to create a free file handle. The handle is used as a parameter for the `Open` instruction, which opens the file. To open a file so that it can be specified as a text file, the `Open` call is:

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

Filename is a string containing the name of the file. FileNo is the handle created by the FreeFile function.

Once the file is opened, the Print instruction can be described line by line:

```
Print #FileNo, "This is a test line."
```

FileNo also stands for the file handle here. The second parameter specifies the text that is to be saved as a line of the text file.

Once the writing process has been completed, the file must be closed using a Close call:

```
Close #FileNo
```

Again here, the file handle should be specified.

The following example shows how a text file is opened, described and closed:

```
Dim FileNo As Integer
Dim CurrentLine As String
Dim Filename As String

Filename = "c:\data.txt"           ' Define file name
FileNo = Freefile                  ' Establish free file handle

Open Filename For Output As #FileNo ' Open file (writing mode)
Print #FileNo, "This is a line of text" ' Save line
Print #FileNo, "This is another line of text" ' Save line
Close #FileNo                       ' Close file
```

Reading Text Files

Text files are read in the same way that they are written. The Open instruction used to open the file contains the For Input expression in place of the For Output expression and, rather than the Print command for writing data, the Line Input instruction should be used to read the data.

Finally, when calling up a text file, the instruction

```
eof(FileNo)
```

is used to check whether the end of the file has been reached.

The following example shows how a text file can be read in:

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

' Define filename
Filename = "c:\data.txt"
```

```

' Establish free file handle
FileNo = Freefile

' Open file (reading mode)
Open Filename For Input As FileNo

' Check whether file end has been reached

Do While not eof(FileNo)
  ' Read line
  Line Input #FileNo, CurrentLine
  If CurrentLine <>" " then
    Msg = Msg & CurrentLine & Chr(13)
  end if
Loop

' Close file
Close #FileNo

Msgbox Msg

```

The individual lines are retrieved in a Do While loop, saved in the `Msg` variable, and displayed at the end in a message box.

Message and Input Boxes

StarOffice Basic provides the `MsgBox` and `InputBox` functions for basic user communication.

Displaying Messages

`MsgBox` displays a basic information box, which can have one or more buttons. In its simplest variant

```
MsgBox "This is a piece of information!"
```

the `MsgBox` only contains text and an OK button.

The appearance of the information box can be changed using a parameter. The parameter provides the option of adding additional buttons, defining the pre-assigned button, and adding an information symbol. The values for selecting the buttons are:

- 0 - OK button
- 1 - OK and Cancel button
- 2 - Cancel and Retry buttons

- 3 - Yes, No, and Cancel buttons
- 4 - Yes and No buttons
- 5 - Retry and Cancel buttons

To set a button as the default button, add one of the following values to the parameter value from the list of button selections. For example, to create Yes, No and Cancel buttons (value 3) where Cancel is the default (value 512), the parameter value is $3 + 512 = 515$.

- 0 - First button is default value
- 256 - Second button is default value
- 512 - Third button is default value

Finally, the following information symbols are available and can also be displayed by adding the relevant parameter values:

- 16 - Stop sign
- 32 - Question mark
- 48 - Exclamation point
- 64 - Tip icon

The call

```
MsgBox "Do you want to continue?", 292
```

displays an information box with the Yes and No buttons (value 4), of which the second button (No) is set as the default value (value 256) and which also receives a question mark (value 32), $4+256+32=292$

If an information box contains several buttons, then a return value should be queried to determine which button has been pressed. The following return values are available in this instance:

- 1 - Ok
- 2 - Cancel
- 4 - Retry
- 5 - Ignore
- 6 - Yes
- 7 - No

In the previous example, checking the return values could be as follows:

```
If MsgBox ("Do you want to continue?", 292) = 6 Then
    ' Yes button pressed
Else
    ' No button pressed
End IF
```

In addition to the information text and the parameter for arranging the information box, MsgBox also permits a third parameter, which defines the text for the box title:

```
MsgBox "Do you want to continue?", 292, "Box Title"
```

If no box title is specified, the default is "soffice".

Input Box For Querying Simple Strings

The `InputBox` function queries simple strings from the user. It is therefore a simple alternative to configuring dialogs. `InputBox` receives three standard parameters:

- an information text
- a box title
- a default value which can be added within the input area

```
InputVal = InputBox("Please enter value:", "Test", "default value")
```

As a return value, the `InputBox` provides the string typed by the user.

Other functions

Beep

The `Beep` function causes the system to play a sound that can be used to warn the user of an incorrect action. `Beep` does not have any parameters:

```
Beep ' creates an informative tone
```

Shell

External programs can be started using the `Shell` function.

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

`Pathname` defines the path of the program to be executed. `Windowstyle` defines the window in which the program is started. The following values are possible:

- 0 - The program receives the focus and is started in a concealed window.
- 1 - The program receives the focus and is started in a normal-sized window.
- 2 - The program receives the focus and is started in a minimized window.
- 3 - The program receives the focus and is started in a maximized window.
- 4 - The program is started in a normal-sized window, without receiving the focus.
- 6 - The program is started in a minimized window, the focus remains in the current window.
- 10 - The program is started in full screen mode.

The third parameter, `Param`, permits command line parameters to be transferred to the program to be started.

Wait

The `Wait` function terminates program execution for a specified time. The waiting period is specified in milliseconds. The command

```
wait 2000
```

specifies an interrupt of 2 seconds (2000 milliseconds).

Environ

The `Environ` function returns the environmental variables of the operating system. Depending on the system and configuration, various types of data are saved here. The call

```
Dim TempDir
```

```
TempDir=Environ ("TEMP")
```

determines the environment variables of temporary directory of the operating system.

Introduction to the StarOffice API

The StarOffice API is a universal programming interface for access to StarOffice. You can use the StarOffice API to create, open, modify and print out StarOffice documents. It provides the option of extending the functional scope of StarOffice through personal macros and allows personal dialogs to be written.

The StarOffice API may not only be used with StarOffice Basic, but also with other programming languages such as Java and C++. A technique called *Universal Network Objects* (UNO) which provides an interface to various programming languages makes this possible.

This chapter centers on how StarOffice can be used in StarOffice Basic with the aid of UNO. It describes the main concepts of UNO from the standpoint of a StarOffice Basic programmer. Details on how to work with the various parts of the StarOffice API can be found in the following chapters.

Universal Network Objects (UNO)

StarOffice provides a programming interface in the form of the Universal Network Objects (UNO). This is an object-oriented programming interface which StarOffice sub-divides into various objects which for their part ensure program-controlled access to the Office package.

Since StarOffice Basic is a procedural programming language, several linguistic constructs have had to be added to it which enable the use of UNO.

To use a Universal Network Object in StarOffice Basic, you will need a variable declaration for the associated object. The declaration is made using the `Dim` instruction (see [Chapter 2](#)). The `Object` type designation should be used to declare an object variable:

```
Dim Obj As Object
```

The call declares an object variable named `Obj`.

The object variable created must then be initialized so that it can be used. This can be done using the `createUnoService` function:

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

This call assigns to the `Obj` variable a reference to the newly created object. `com.sun.star.frame.Desktop` resembles an object type; however in UNO terminology it is called a “*service*” rather than a type. In accordance with UNO philosophy, an `Obj` is described as a reference to an object which supports the `com.sun.star.frame.Desktop` service. The “*service*” term used in StarOffice Basic therefore corresponds to the *type* and *class* terms used in other programming languages.

There is, however, one main difference: a Universal Network Object may support several services at the same time. Some UNO services in turn support other services so that, through one object, you are provided with a whole range of services. For example, that the aforementioned object, which is based on the `com.sun.star.frame.Desktop` service, can also include other services for loading documents and for ending the program.

Note – Whereas the structure of an object in VBA is defined by the class to which it belongs, in StarOffice Basic the structure is defined through the services which it supports. A VBA object is always assigned to precisely one single class. A StarOffice Basic object can, however, support several services.

Properties and Methods

An object in StarOffice Basic provides a range of properties and methods which can be called by means of the object.

Properties

Properties are like the properties of an object; for example, `Filename` and `Title` for a Document object.

The properties are set by means of a simple assignment:

```
Document.Title = "StarOffice 8 Basic Programmer's Guide"  
Document.Filename = "progman.sxv"
```

A property, just like a normal variable, has a type that defines which values it can record. The preceding `Filename` and `Title` properties are of the string type.

Real Properties and Imitated Properties

Most of the properties of an object in StarOffice Basic are defined as such in the UNO description of the service. In addition to these "real" properties, there are also properties in StarOffice Basic which consist of two methods at the UNO level. One of these is used to query the value of the property and the other is issued to set it (`get` and `set` methods). The property has been virtually imitated from two methods. Character objects in UNO, for example, provide the `getPosition` and `setPosition` methods through which the associated key point can be called up and changed. The StarOffice Basic programmer can access the values through the `Position` property. Regardless of this, the original methods are also available (in our example, `getPosition` and `setPosition`).

Methods

Methods can be understood as functions that relate directly to an object and through which this object is called. The preceding `Document` object could, for example, provide a `Save` method, which can be called as follows:

```
Document . Save ()
```

Methods, just like functions, may contain parameters and return values. The syntax of such method calls is oriented towards classic functions. The call

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

also specifies the `True` parameter for the document object when requesting the `Save` method. Once the method has been completed, `Save` saves a return value in the `Ok` variable.

Modules, Services and Interfaces

StarOffice provides hundreds of services. To provide an overview of these services, they have been combined into modules. The modules are of no other functional importance for StarOffice Basic programmers. When specifying a service name, it is only the module name which is of any importance because this must be also listed in the name. The complete name of a service consists of the `com.sun.star` expression, which specifies that it is a StarOffice service, followed by the module name, such as `frame`, and finally the actual service name, such as `Desktop`. The complete name in the named example would be:

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

In addition to the module and service terms, UNO introduces the term "*interface*". While this term may be familiar to Java programmers, it is not used in Basic.

An interface combines several methods. In the strictest sense of the word, a service in UNO does not support methods, but rather interfaces, which in turn provide different methods. In other words, the methods are assigned (as combinations) to the service in interfaces. This detail may be of interest in particular to Java- or C++ programmers, since in these languages, the interface is needed to request a method. In StarOffice Basic, this is irrelevant. Here, the methods are called directly by means of the relevant object.

For an understanding of the API, it is, however, useful to have the assignment of methods to various interfaces handy, since many interfaces are used in the different services. If you are familiar with an interface, then you can transfer your knowledge from one service to another.

Some central interfaces are used so frequently that they are shown again at the end of this chapter, triggered by different services.

Tools for Working with UNO

The question remains as to which objects — or services if we are going to remain with UNO terminology — support which properties, methods and interfaces and how these can be determined. In addition to this guide, you can get more information about objects from the following sources: the `supportsService` method, the debug methods as well as the Developer's Guide, and the API reference.

The `supportsService` Method

A number of UNO objects support the `supportsService` method, with which you can establish whether an object supports a particular service. The call

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

for example, determines whether the `TextElement` object supports the `com.sun.star.text.Paragraph` service.

Debug Properties

Every UNO object in StarOffice Basic knows what properties, methods and interfaces it already contains. It provides properties that return these in the form of a list. The corresponding properties are:

DBG_properties - returns a string containing all properties of an object

DBG_methods - returns a string containing all methods of an object

DBG_supportedInterfaces - returns a string containing all interfaces which support an object.

The following program code shows how `DBG_properties` and `DBG_methods` can be used in real-life applications. It first creates the `com.sun.star.frame.Desktop` service and then displays the supported properties and methods in message boxes.

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

MsgBox Obj.DBG_Propierties
MsgBox Obj.DBG_methods
```

When using `DBG_properties`, note that the function returns all properties that one particular service can theoretically support. No assurances are, however, provided for whether these can also be used by the object in question. Before calling up properties, you must therefore use the `IsEmpty` function to check whether this is actually available.

API Reference

More information about the available services, and their interfaces, methods and properties can be found in the API reference for the StarOffice API. This can be found at www.openoffice.org:

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

An Overview of a Few Central Interfaces

Some interfaces of StarOffice can be found in many parts of the StarOffice API. They define sets of methods for abstract tasks which can be applied to various problems. Here, you will find an overview of the most common of these interfaces.

The origin of the objects is explained at a later point in this guide. At this point, only some of the abstract aspects of objects, for which the StarOffice API provides some central interfaces, are discussed.

Creating Context-Dependent Objects

The StarOffice API provides two options for creating objects. One can be found in the `createUnoService` function mentioned at the start of this chapter. `createUnoService` creates an object which can be used universally. Such objects and services are also known as *context-independent services*.

In addition to context-independent services, there are also *context-dependent services* whose objects are only useful when used in conjunction with another object. A drawing object for a spreadsheet document, for example, can therefore only exist in conjunction with this one document.

com.sun.star.lang.XMultiServiceFactory Interface

Context-dependent objects are usually created by means of an object method, on which the object depends. The `createInstance` method, which is defined in the `XMultiServiceFactory` interface, is used in particular in the document objects.

The aforementioned drawing object can, for example, be created as follows using a spreadsheet object:

```
Dim RectangleShape As Object

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

A paragraph template in a text document is created in the same way:

```
Dim Style As Object
Style = Textdocument.CreateInstance("com.sun.star.style.ParagraphStyle")
```

Named Access to Subordinate Objects

The `XNameAccess` and `XNameContainer` interfaces are used in objects that contain subordinate objects, which can be addressed using a natural language name.

While `XNamedAccess` permits access to the individual objects, `XNameContainer` takes on the insertion, modification and deletion of elements.

com.sun.star.container.XNameAccess Interface

An example of the use of `XNameAccess` is provided by the `sheet` object of a spreadsheet. It combines all the pages within the spreadsheet. The individual pages are accessed using the `getByName` method from `XNameAccess`:

```
Dim Sheets As Object
Dim Sheet As Object

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

The `getElementNames` method provides an overview of the names of all elements. As a result, it returns a data field containing the names. The following example shows how all element names of a spreadsheet can thereby be determined and displayed in a loop:

```

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

```

The `hasByName` method of the `XNameAccess` interface reveals whether a subordinate object with a particular name exists within the basic object. The following example therefore displays a message that informs the user whether the `Spreadsheet` object contains a page of the name `Sheet1`.

```

Dim Sheets As Object

Sheets = Spreadsheet.Sheets
If Sheets.HasByName("Sheet1") Then
    MsgBox " Sheet1 available"
Else
    MsgBox "Sheet1 not available"
End If

```

`com.sun.star.container.XNameContainer` Interface

The `XNameContainer` interface takes on the insertion, deletion and modification of subordinate elements in a basic object. The functions responsible are `insertByName`, `removeByName` and `replaceByName`.

The following is a practical example of this. It calls a text document, which contains a `StyleFamilies` object and uses this to in turn make the paragraph templates (`ParagraphStyles`) of the document available.

```

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")

```

The `insertByName` line inserts the `NewStyle` style under the name of the same name in the `ParagraphStyles` object. The `replaceByName` line changes the object behind `ChangingStyle` into `NewStyle`. Finally, the `removeByName` call removes the object behind `OldStyle` from `ParagraphStyles`.

Index-Based Access to Subordinate Objects

The `XIndexAccess` and `XIndexContainer` interfaces are used in objects which contain subordinate objects and which can be addressed using an index.

`XIndexAccess` provides the methods for accessing individual objects.

`XIndexContainer` provides methods for inserting and removing elements.

`com.sun.star.container.XIndexAccess` Interface

`XIndexAccess` provides the `getByIndex` and `getCount` methods for calling the subordinate objects. `getByIndex` provides an object with a particular index. `getCount` returns how many objects are available.

```
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)
    ' Editing sheet
Next I
```

The example shows a loop that runs through all sheet elements one after another and saves a reference to each in the `Sheet` object variable. When working with the indexes, note that `getCount` returns the number of elements. The elements in `getByIndex` however are numbered beginning with 0. The counting variable of the loop therefore runs from 0 to `getCount() - 1`.

`com.sun.star.container.XIndexContainer` Interface

The `XIndexContainer` interface provides the `insertByIndex` and `removeByIndex` functions. The parameters are structured in the same way as the corresponding functions in `XNameContainer`.

Iterative Access to Subordinate Objects

In some instances, an object may contain a list of subordinate objects that cannot be addressed by either a name or an index. In these situations, the `XEnumeration` and `XEnumerationAccess` interfaces are appropriate. They provide a mechanism through which all subordinate elements of an objects can be passed, step by step, without having to use direct addressing.

com.sun.star.container.XEnumeration and XEnumerationAccess Interfaces

The basic object must provide the `XEnumerationAccess` interface, which contains only a `createEnumeration` method. This returns an auxiliary object, which in turn provides the `XEnumeration` interface with the `hasMoreElements` and `nextElement` methods. Through these, you then have access to the subordinate objects.

The following example steps through all the paragraphs of a text:

```
Dim ParagraphEnumeration As Object
Dim Paragraph As Object

ParagraphEnumeration = Textdoc.Text.createEnumeration

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

The example first creates a `ParagraphEnumeration` auxiliary object. This gradually returns the individual paragraphs of the text in a loop. The loop is terminated as soon as the `hasMoreElements` method returns the `False` value, signaling that the end of the text has been reached.

Working with StarOffice Documents

The StarOffice API has been structured so that as many of its parts as possible can be used universally for different tasks. This includes the interfaces and services for creating, opening, saving, converting, and printing documents and for template administration. Since these function areas are available in all types of documents, they are explained first in this chapter.

The StarDesktop

When working with documents, there are two services which are used most frequently:

- The `com.sun.star.frame.Desktop` service, which is similar to the core service of StarOffice. It provides the functions for the frame object of StarOffice, under which all document windows are classified. Documents can also be created, opened and imported using this service.
- The basic functionality for the individual document objects is provided by the `com.sun.star.document.OfficeDocument` service. This provides the methods for saving, exporting and printing documents.

The `com.sun.star.frame.Desktop` service opens automatically when StarOffice is started. To do this, StarOffice creates an object which can be reached by means of the global name `StarDesktop`.

The most important interface of the `StarDesktop` is `com.sun.star.frame.XComponentLoader`. This basically covers the `loadComponentFromURL` method, which is responsible for creating, importing, and opening documents.

The name of the `StarDesktop` object dates back to StarOffice 5, in which all document windows were embedded in one common application called `StarDesktop`. In the present version of StarOffice, a visible `StarDesktop` is no longer used. The name `StarDesktop` was, however, retained for the frame object of StarOffice because it clearly indicates that this is a basic object for the entire application.

The `StarDesktop` object assumes the position of successor to the `Application` object of StarOffice 5 which previously applied as a root object. Unlike the old `Application` object however it is primarily responsible for opening new documents. The functions resident in the old `Application` object for controlling the on-screen depiction of StarOffice (for example, `FullScreen`, `FunctionBarVisible`, `Height`, `Width`, `Top`, `Visible`) are no longer used.

Note – Whereas the active document in Word is accessed through `Application.ActiveDocument` and in Excel through `Application.ActiveWorkbook`, in StarOffice, the `StarDesktop` is responsible for this task. The active document object is accessed in StarOffice 7 through the `StarDesktop.CurrentComponent` property.

Basic Information about Documents in StarOffice

When working with StarOffice documents, it is useful to deal with some of the basic issues of document administration in StarOffice. This includes the way in which file names are structured for StarOffice documents, as well as the format in which files are saved.

File Names in URL Notation

Since StarOffice is a platform-independent application, it uses URL notation (which is independent of any operating system), as defined in the Internet Standard RFC 1738 for file names. Standard file names using this system begin with the prefix

```
file:///
```

followed by the local path. If the file name contains sub-directories, then these are separated by a single *forward* slash, not with a backslash usually used under Windows. The following path references the `test.sxw` file in the `doc` directory on the C: drive.

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

To convert local file names into an URL, StarOffice provides the `ConvertToUrl` function. To convert a URL into a local file name, StarOffice provides the `ConvertFromUrl` function:

```

MsgBox ConvertToUrl ("C:\doc\test.sxw")
    ' supplies file:///C:/doc/test.sxw
MsgBox ConvertFromUrl ("file:///C:/doc/test.sxw")
    ' supplies (under Windows) c:\doc\test.sxw

```

The example converts a local file name into a URL and displays it in a message box. It then converts a URL into a local file name and also displays this.

The Internet Standard RFC 1738, upon which this is based, permits use of the 0-9, a-z, and A-Z characters. All other characters are inserted as escape coding in the URLs. To do this, they are converted into their hexadecimal value in the ISO 8859-1 (ISO-Latin) set of characters and are preceded by a percent sign. A space in a local file name therefore, for example, becomes a %20 in the URL.

XML File Format

Since Version 6.0, StarOffice provides an XML-based file format. Through the use of XML, the user has the option of also opening and editing files in other programs.

Compression of Files

Since XML is based on standard text files, the resultant files are usually very large. StarOffice therefore compresses the files and saves them as a ZIP file. By means of a `storeAsURL` method option, the user can save the original XML files directly. See [“storeAsURL Method Options” on page 85](#).

Creating, Opening and Importing Documents

Documents are opened, imported and created using the method

```

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

```

The first parameter of `loadComponentFromURL` specifies the URL of the associated file.

As the second parameter, `loadComponentFromURL` expects a name for the frame object of the window that StarOffice creates internally for its administration. The predefined `_blank` name is usually specified here, and this ensures that StarOffice creates a new window. Alternatively, `_hidden` can also be specified, and this ensures that the corresponding document is loaded but remains invisible.

Using these parameters, the user can open a StarOffice document, since place holders (dummy values) can be assigned to the last two parameters:

```

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

```

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

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

The preceding call opens the `test.sxw` file and displays this in a new window.

Any number of documents can be opened in this way in StarOffice Basic and then edited using the returned document objects.

Note – `StarDesktop.loadComponentFromURL` supersedes the `Documents.Add` and `Documents.Open` methods from the old StarOffice API.

Replacing the Content of the Document Window

The named `_blank` and `_hidden` values for the `Frame` parameter ensure that StarOffice creates a new window for every call from `loadComponentFromURL`. In some situations, it is useful to replace the content of an existing window. In this case, the frame object of the window should contain an explicit name. Note that this name must not begin with an underscore. Furthermore, the `SearchFlags` parameter must be set so that the corresponding framework is created, if it does not already exist. The corresponding constant for `SearchFlags` is:

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

The following example shows how the content of an opened window can be replaced with the help of the frame parameter and `SearchFlags`:

```
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 "Press OK to display the second document."  
  
Url = "file:///C:/test2.sxw"  
Doc = StarDesktop.loadComponentFromURL(Url, "MyFrame", _  
              SearchFlags, Dummy)
```

The example first opens the `test.sxw` file in a new window with the frame name of `MyFrame`. Once the message box has been confirmed, it replaces the content of the window with the `test2.sxw` file.

loadComponentFromURL Method Options

The fourth parameter of the `loadComponentFromURL` function is a `PropertyValue` data field, which provides StarOffice with various options for opening and creating documents. The data field must provide a `PropertyValue` structure for each option in which the name of the option is saved as a string as well as the associated value.

`loadComponentFromURL` supports the following options:

- **AsTemplate (Boolean)** – if true, loads a new, untitled document from the given URL. If is false, template files are loaded for editing.
- **CharacterSet (String)** – defines which set of characters a document is based on.
- **FilterName (String)** – specifies a special filter for the `loadComponentFromURL` function. The filter names available are defined in the `\share\config\registry\instance\org\openoffice\office\TypeDetection.xml` file.
- **FilterOptions (String)** – defines additional options for filters.
- **JumpMark (String)** – once a document has been opened, jumps to the position defined in `JumpMark`.
- **Password (String)** – transfers a password for a protected file.
- **ReadOnly (Boolean)** – loads a read-only document.

The following example shows how a text file separated by a comma in StarOffice Calc can be opened using the `FilterName` option.

```
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())
```

The `FileProperties` data field covers precisely one value because it records one option. The `Filtername` property defines whether StarOffice uses a StarOffice Calc text filter to open files.

Creating New Documents

StarOffice automatically creates a new document if the document specified in the URL is a template.

Alternatively, if only an empty document without any adaptation is needed, a `private:factory` URL can be specified:

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

```
Dim Doc As Object

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

The call creates an empty StarOffice writer document.

Document Objects

The `loadComponentFromURL` function introduced in the previous section returns a document object. This supports the `com.sun.star.document.OfficeDocument` service, which in turn provides two central interfaces:

- the `com.sun.star.frame.XStorable` interface, which is responsible for saving documents and
- the `com.sun.star.view.XPrintable` interface, which contains the methods for printing documents.

Note – When changing over to StarOffice 8, you will find that the functional scope of the document objects has remained the same for the most part. The document objects, for example, still provide methods for saving and printing documents. The names and parameters of the methods have, however, changed.

Saving and Exporting Documents

StarOffice documents are saved directly through the document object. The `store` method of the `com.sun.star.frame.XStorable` interface is available for this purpose:

```
Doc.store()
```

This call functions provided that the document has already been assigned a memory space. This is not the case for new documents. In this instance, the `storeAsURL` method is used. This method is also defined in `com.sun.star.frame.XStorable` and can be used to define the location of the document:

```
Dim URL As String
Dim Dummy()

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

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

In addition to the preceding methods, `com.sun.star.frame.XStorable` also provides some help methods which are useful when saving documents. These are:

- **hasLocation()** – specifies whether the document has already been assigned a URL.

- **isReadOnly()** – specifies whether a document has read-only protection.
- **isModified()** – specifies whether a document has been modified since it was last saved.

The code for saving a document can be extended by these options so that the document is only saved if the object has actually been modified and the file name is only queried if it is actually needed:

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

The example first checks whether the relevant document has been modified since it was last saved. It only continues with the saving process if this is the case. If the document has already been assigned a URL and is not a read-only document, it is saved under the existing URL. If it does not have a URL or was opened in its read-only status, it is saved under a new URL.

storeAsURL Method Options

As with the `loadComponentFromURL` method, some options can also be specified in the form of a `PropertyValue` data field using the `storeAsURL` method. These determine the procedure StarOffice uses when saving a document. `storeAsURL` provides the following options:

- **CharacterSet (String)** – defines which set of characters a document is based on.
- **FilterName (String)** – specifies a special filter for the `loadComponentFromURL` function. The filter names available are defined in the `\share\config\registry\instance\org\openoffice\office\TypeDetection.xml` file.
- **FilterOptions (String)** – defines additional options for filters.
- **Overwrite (Boolean)** – allows a file which already exists to be overwritten without a query.
- **Password (String)** – transfers the password for a protected file.
- **Unpacked (Boolean)** – saves the document (not compressed) in sub-directories.

The following example shows how the `Overwrite` option can be used in conjunction with `storeAsURL`:

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

' ... Initialize Doc
```

```

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

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

Doc.storeAsURL(sUrl, mFileProperties())

```

The example then saves Doc under the specified file name if a file already exists under the name.

Printing Documents

Similar to saving, documents are printed out directly by means of the document object. The Print method of the `com.sun.star.view.Xprintable` interface is provided for this purpose. In its simplest form, the print call is:

```

Dim Dummy()

Doc.print(Dummy())

```

As in the case of the `loadComponentFromURL` method, the Dummy parameter is a `PropertyValue` data field through which StarOffice can specify several options for printing.

The options of the print method

The print method expects a `PropertyValue` data field as a parameter, which reflects the settings of the print dialog of StarOffice:

- **CopyCount (Integer)** – specifies the number of copies to be printed.
- **FileName (String)** – prints the document in the specified file.
- **Collate (Boolean)** – advises the printer to collate the pages of the copies.
- **Sort (Boolean)** – sorts the pages when printing out several copies (`CopyCount > 1`).
- **Pages (String)** – contains the list of the pages to be printed (syntax as specified in print dialog).

The following example shows how several pages of a document can be printed out using the Pages option:

```

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())

```

Printer selection and settings

The `com.sun.star.view.XPrintable` interface provides the `Printer` property, which selects the printer. This property receives a `PropertyValue` data field with the following settings:

- **Name (String)** – specifies the name of printer.
- **PaperOrientation (Enum)** – specifies the paper orientation (`com.sun.star.view.PaperOrientation.PORTRAIT` value for portrait format, `com.sun.star.view.PaperOrientation.LANDSCAPE` for landscape format).
- **PaperFormat (Enum)** – specifies the paper format (for example, `com.sun.star.view.PaperFormat.A4` for DIN A4 or `com.sun.star.view.PaperFormat.Letter` for US letters).
- **PaperSize (Size)** – specifies the paper size in hundredths of a millimeter.

The following example shows how a printer can be changed and the paper size set with the help of the `Printer` property.

```
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 ' corresponds to 20 cm
PaperSize.Height = 20000 ' corresponds to 20 cm

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

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

Doc.Printer = PrinterProperties()
```

The example defines an object named `PaperSize` with the `com.sun.star.awt.Size` type. This is needed to specify the paper size. Furthermore, it creates a data field for two `PropertyValue` entries named `PrinterProperties`. This data field is then initialized with the values to be set and assigned the `Printer` property. From the standpoint of UNO, the printer is not a real property but an imitated one.

Templates

Templates are named lists containing formatting attributes. They move through all applications of StarOffice and help to significantly simplify formatting. If the user changes one of the attributes of a template, then StarOffice automatically adjusts all

document sections depending on the attribute. The user can therefore, for example, change the font type of all level one headers by means of a central modification in the document. Depending on the relevant document types, StarOffice recognizes a whole range of different types of template.

StarOffice Writer supports

- character templates,
- paragraph templates,
- frame templates,
- page templates
- numbering templates

StarOffice Calc supports

- cell template
- page templates

StarOffice Impress supports

- character element templates
- presentation templates

In StarOffice terminology, the different types of templates are called `StyleFamilies` in accordance with the `com.sun.star.style.StyleFamily` service on which they are based. The `StyleFamilies` are accessed by means of the document object:

```
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")
```

The example uses the `StyleFamilies` property of a spreadsheet document to establish a list containing all available cell templates.

The individual templates can be accessed directly by means of an index:

```
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
```

The loop added since the previous example displays the names of all cell templates one after another in a message box.

Details about various formatting options

Each type of template provides a whole range of individual formatting properties. Here is an overview of the most important formatting properties and the points at which they are explained:

- Character properties, Chapter 6, Text Documents, `com.sun.star.style.CharacterProperties` service
- Paragraph properties, Chapter 6, Text Documents, `com.sun.star.text.Paragraph` service
- Cell properties, Chapter 7, Spreadsheet Documents, `com.sun.star.table.CellProperties` service
- Page properties, Chapter 7, Spreadsheet Documents, `com.sun.star.style.PageStyle` service
- Character element properties, Chapter 7, Spreadsheet Documents, Various services

The format properties are by no means restricted to the applications in which these are explained, but instead can be used universally. For example, most of the page properties described in [Chapter 7](#) can therefore be used not only in StarOffice Calc, but also in StarOffice Writer.

More information about working with templates can be found in the *Default values for character and paragraph properties* section in [Chapter 6](#).

Text Documents

In addition to pure strings, text documents also contain formatting information. These may appear at any point in the text. The structure is further complicated by tables. These include not only single-dimensional strings, but also two-dimensional fields. Most word processing programs now finally provide the option of placing drawing objects, text frames and other objects within a text. These may be outside the flow of text and can be positioned anywhere on the page.

This chapter presents the central interfaces and services of text documents. The first section deals with the anatomy of text documents and concentrates on how a StarOffice Basic program can be used to take iterative steps through a StarOffice document. It focuses on paragraphs, paragraph portions and their formatting.

The second section focuses on efficiently working with text documents. For this purpose, StarOffice provides several help objects, such as the `TextCursor` object, which extend beyond those specified in the first section.

The third section moves beyond work with texts. It concentrates on tables, text frames, text fields, bookmarks, content directories and more.

Information about how to create, open, save and print documents is described in [Chapter 5](#), because it can be used not only for text documents, but also for other types of document.

The Structure of Text Documents

A text document can essentially contain four types of information:

- the actual text
- templates for formatting characters, paragraphs, and pages
- non-text elements such as tables, graphics and drawing objects

- global settings for the text document

This section concentrates especially on the text and associated formatting options.

Note – The design of the StarOffice 8 API for StarOffice Writer differs from that of the previous version. The old API version concentrated on work with the Selection object, which was heavily oriented towards the idea of the user interface for end users, which focused on mouse-controlled highlighting work.

The StarOffice 8 API has replaced these connections between user interface and programmer interface. The programmer therefore has parallel access to all parts of an application and can work with objects from different sub-sections of a document at the same time. The old Selection object is no longer available.

Paragraphs and Paragraph Portions

The core of a text document consists of a sequence of paragraphs. These are neither named nor indexed and there is therefore no possible way of directly accessing individual paragraphs. The paragraphs can however be sequentially traversed with the help of the Enumeration object described in [Chapter 4](#). This allows the paragraphs to be edited.

When working with the Enumeration object, one special scenario should, however, be noted: it not only returns paragraphs, but also tables (strictly speaking, in StarOffice Writer, a table is a special type of paragraph). Before accessing a returned object, you should therefore check whether the returned object supports the `com.sun.star.text.Paragraph` service for paragraphs or the `com.sun.star.text.TextTable` service for tables.

The following example traverses the contents of a text document in a loop and uses a message in each instance to inform the user whether the object in question is a paragraph or table.

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

' Create document object
Doc = StarDesktop.CurrentComponent

' Create enumeration object
Enum = Doc.Text.createEnumeration

' loop over all text elements
While Enum.hasMoreElements
    TextElement = Enum.nextElement

    If TextElement.supportsService("com.sun.star.text.TextTable") Then
```

```

        MsgBox "The current block contains a table."
    End If

    If TextElement.supportsService("com.sun.star.text.Paragraph") Then
        MsgBox "The current block contains a paragraph."
    End If
Wend

```

The example creates a `Doc` document object which references the current StarOffice document. With the aid of `Doc`, the example then creates an `Enumeration` object that traverses through the individual parts of the text (paragraphs and tables) and assigns the current element to `TextElement` object. The example uses the `supportsService` method to check whether the `TextElement` is a paragraph or a table.

Paragraphs

The `com.sun.star.text.Paragraph` service grants access to the content of a paragraph. The text in the paragraph can be retrieved and modified using the `String` property:

```

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

```

The example opens the current text document and passes through it with the help of the `Enumeration` object. It uses the `TextElement.String` property in all paragraphs to access the relevant paragraphs and replaces the `you`, `too` and `for` strings with the `U`, `2` and `4` characters. The `Replace` function used for replacing does not fall within the standard linguistic scope of StarOffice Basic. This is an instance of the example function described in [Chapter 3](#) in the “Search and Replace” on page 54 section.

Note – The content of the procedure described here for accessing the paragraphs of a text is comparable with the Paragraphs listing used in VBA, which is provided in the Range and Document objects available there. Whereas in VBA the paragraphs are accessed by their number (for example, by the Paragraph(1) call), in StarOffice Basic, the Enumeration object described previously should be used.

There is no direct counterpart in StarOffice Basic for the Characters, Sentences and Words lists provided in VBA. You do, however, have the option of switching to a TextCursor which allows for navigation at the level of characters, sentences and words (refer to *The TextCursor*).

Paragraph Portions

The previous example may change the text as requested, but it may sometimes also destroy the formatting.

This is because a paragraph in turn consists of individual sub-objects. Each of these sub-objects contains its own formatting information. If the center of a paragraph, for example, contains a word printed in bold, then it will be represented in StarOffice by three paragraph portions: the portion before the bold type, then the word in bold, and finally the portion after the bold type, which is again depicted as normal.

If the text of the paragraph is now changed using the paragraph's String property, then StarOffice first deletes the old paragraph portions and inserts a new paragraph portion. The formatting of the previous sections is then lost.

To prevent this effect, the user can access the associated paragraph portions rather than the entire paragraph. Paragraphs provide their own Enumeration object for this purpose. The following example shows a double loop which passes over all paragraphs of a text document and the paragraph portions they contain and applies the replacement processes from the previous example:

```
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

' loop over all paragraphs
While Enum1.hasMoreElements
    TextElement = Enum1.nextElement
    If TextElement.supportsService("com.sun.star.text.Paragraph") Then
        Enum2 = TextElement.createEnumeration

        ' loop over all sub-paragraphs
```

```

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

```

The example runs through a text document in a double loop. The outer loop refers to the paragraphs of the text. The inner loop processes the paragraph portions in these paragraphs. The example code modifies the content in each of these paragraph portions using the `String` property of the string, as is the case in the previous example for paragraphs. Since however, the paragraph portions are edited directly, their formatting information is retained when replacing the string.

Formatting

There are various ways of formatting text. The easiest way is to assign the format properties directly to the text sequence. This is called direct formatting. Direct formatting is used in particular with short documents because the formats can be assigned by the user with the mouse. You can, for example, highlight a certain word within a text using bold type or center a line.

In addition to direct formatting, you can also format text using templates. This is called indirect formatting. With indirect formatting, the user assigns a pre-defined template to the relevant text portion. If the layout of the text is changed at a later date, the user only needs to change the template. StarOffice then changes the way in which all text portions which use this template are depicted.

Note – In VBA, the formatting properties of an object are usually spread over a range of sub-objects (for example, `Range.Font`, `Range.Borders`, `Range.Shading`, `Range.ParagraphFormat`). The properties are accessed by means of cascading expressions (for example, `Range.Font.AllCaps`). In StarOffice Basic, the formatting properties on the other hand are available directly, using the relevant objects (`TextCursor`, `Paragraph`, and so on). You will find an overview of the character and paragraph properties available in StarOffice in the following two sections.

Note – In the old StarOffice API, a text was essentially formatted using the `Selection` object and its subordinate objects (for example, `Selection.Font`, `Selection.Paragraph`, and `Selection.Border`). In the new API, the formatting properties can be found in each object (`Paragraph`, `TextCursor`, and so on) and can be applied directly. A list of the character and paragraph properties available can be found in the following paragraphs.

Character Properties

Those format properties that refer to individual characters are described as character properties. These include bold type and the font type. Objects that allow character properties to be set have to support the `com.sun.star.style.CharacterProperties` service. StarOffice recognizes a whole range of services that support this service. These include the previously described `com.sun.star.text.Paragraph` services for paragraphs as well as the `com.sun.star.text.TextPortion` services for paragraph portions.

The `com.sun.star.style.CharacterProperties` service does not provide any interfaces, but instead offers a range of properties through which character properties can be defined and called. A complete list of all character properties can be found in the StarOffice API reference. The following list describes the most important properties:

- **CharFontName (String)** - name of font type selected.
- **CharColor (Long)** - text color.
- **CharHeight (Float)** - character height in points (pt).
- **CharUnderline (Constant group)** - type of underscore (constants in accordance with `com.sun.star.awt.FontUnderline`).
- **CharWeight (Constant group)** - font weight (constants in accordance with `com.sun.star.awt.FontWeight`).
- **CharBackColor (Long)** - background color.
- **CharKeepTogether (Boolean)** - suppression of automatic line break.
- **CharStyleName (String)** - name of character template.

Paragraph Properties

Formatting information that does not refer to individual characters, but to the entire paragraph is considered to be a paragraph property. This includes the distance of the paragraph from the edge of the page as well as line spacing. The paragraph properties are available through the `com.sun.star.style.ParagraphProperties` service.

Even the paragraph properties are available in various objects. All objects that support the `com.sun.star.text.Paragraph` service also provide support for the paragraph properties in `com.sun.star.style.ParagraphProperties`.

A complete list of the paragraph properties can be found in the StarOffice API reference. The most common paragraph properties are:

- **ParaAdjust (enum)** - vertical text orientation (constants in accordance with `com.sun.star.style.ParagraphAdjust`).
- **ParaLineSpacing (struct)** - line spacing (structure in accordance with `com.sun.star.style.LineSpacing`).
- **ParaBackColor (Long)** - background color.
- **ParaLeftMargin (Long)** - left margin in 100ths of a millimeter.
- **ParaRightMargin (Long)** - right margin in 100ths of a millimeter.
- **ParaTopMargin (Long)** - top margin in 100ths of a millimeter.
- **ParaBottomMargin (Long)** - bottom margin in 100ths of a millimeter.
- **ParaTabStops (Array of struct)** - type and position of tabs (array with structures of the Typs `com.sun.star.style.TabStop`).
- **ParaStyleName (String)** - name of the paragraph template.

Example: simple HTML export

The following example demonstrates how to work with formatting information. It iterates through a text document and creates a simple HTML file. Each paragraph is recorded in its own HTML element `<P>` for this purpose. Paragraph portions displayed in bold type are marked using a `` HTML element when exporting.

```
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

' loop over all paragraphs
While Enum1.hasMoreElements
    TextElement = Enum1.nextElement
    If TextElement.supportsService("com.sun.star.text.Paragraph") Then
        Enum2 = TextElement.createEnumeration
        CurLine = "<P>"

        ' loop over all paragraph portions
        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

' output the line
CurLine = CurLine & "</P>"
Print #FileNo, CurLine

End If
Wend

' write HTML footer
Print #FileNo, "</BODY></HTML>"
Close #FileNo

```

The basic structure of the example is oriented towards the examples for running through the paragraph portions of a text already discussed previously. The functions for writing the HTML file, as well as a test code that checks the font weight of the corresponding text portions and provides paragraph portions in bold type with a corresponding HTML tag, have been added.

Default values for character and paragraph properties

Direct formatting always takes priority over *indirect* formatting. In other words, formatting using templates is assigned a lower priority than direct formatting in a text.

Establishing whether a section of a document has been directly or indirectly formatted is not easy. The symbol bars provided by StarOffice show the common text properties such as font type, weight and size. However, whether the corresponding settings are based on template or direct formatting in the text is still unclear.

StarOffice Basic provides the `getPropertyState` method, with which programmers can check how a certain property was formatted. As a parameter, this takes the name of the property and returns a constant that provides information about the origin of the formatting. The following responses, which are defined in the `com.sun.star.beans.PropertyState` enumeration, are possible:

- **com.sun.star.beans.PropertyState.DIRECT_VALUE** - the property is defined directly in the text (direct formatting),
- **com.sun.star.beans.PropertyState.DEFAULT_VALUE** - the property is defined by a template (indirect formatting)
- **com.sun.star.beans.PropertyState.AMBIGUOUS_VALUE** - the property is unclear. This status arises, for example, when querying the bold type property of a paragraph, which includes both words depicted in bold and words depicted in normal font.

The following example shows how format properties can be edited in StarOffice. It searches through a text for paragraph portions which have been depicted as bold type using direct formatting. If it encounters a corresponding paragraph portion, it deletes the direct formatting using the `setPropertyToDefault` method and assigns a `MyBold` character template to the corresponding paragraph portion.

```
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

' loop over all paragraphs
While Enum1.hasMoreElements
  TextElement = Enum1.nextElement
  If TextElement.supportsService("com.sun.star.text.Paragraph") Then
    Enum2 = TextElement.createEnumeration

    ' loop over all paragraph portions
    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

End If

Wend
```

Editing Text Documents

The previous section has already discussed a whole range of options for editing text documents, focusing on the `com.sun.star.text.TextPortion` and `com.sun.star.text.Paragraph` services, which grant access to paragraph portions as well as paragraphs. These services are appropriate for applications in

which the content of a text is to be edited in one pass through a loop. However, this is not sufficient for many problems. StarOffice provides the `com.sun.star.text.TextCursor` service for more complicated tasks, including navigating backward within a document or navigating based on sentences and words rather than `TextPortions`.

The TextCursor

A `TextCursor` in the StarOffice API is comparable with the visible cursor used in a StarOffice document. It marks a certain point within a text document and can be navigated in various directions through the use of commands. The `TextCursor` objects available in StarOffice Basic should not, however, be confused with the visible cursor. These are two very different things.

Note – Terminology differs from that used in VBA: In terms of scope of function, the Range object from VBA can be compared with the `TextCursor` object in StarOffice and not — as the name possibly suggests — with the Range object in StarOffice.

The `TextCursor` object in StarOffice, for example, provides methods for navigating and changing text which are included in the Range object in VBA (for example, `MoveStart`, `MoveEnd`, `InsertBefore`, `InsertAfter`). The corresponding counterparts of the `TextCursor` object in StarOffice are described in the following sections.

Navigating within a Text

The `TextCursor` object in StarOffice Basic acts independently from the visible cursor in a text document. A program-controlled position change of a `TextCursor` object has no impact whatsoever on the visible cursor. Several `TextCursor` objects can even be opened for the same document and used in various positions, which are independent of one another.

A `TextCursor` object is created using the `createTextCursor` call:

```
Dim Doc As Object
Dim Cursor As Object

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

The `Cursor` object created in this way supports the `com.sun.star.text.TextCursor` service, which in turn provides a whole range of methods for navigating within text documents. The following example first moves the `TextCursor` ten characters to the left and then three characters to the right:

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

A `TextCursor` can highlight a complete area. This can be compared with highlighting a point in the text using the mouse. The `False` parameter in the previous function call specifies whether the area passed over with the cursor movement is highlighted. For example, the `TextCursor` in the following example

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

first moves ten characters to the right without highlighting, and then moves back three characters and highlights this. The area highlighted by the `TextCursor` therefore begins after the seventh character in the text and ends after the tenth character.

Here are the central methods that the `com.sun.star.text.TextCursor` service provides for navigation:

- **goLeft (Count, Expand)** – jumps Count characters to the left.
- **goRight (Count, Expand)** – jumps Count characters to the right.
- **gotoStart (Expand)** – jumps to the start of the text document.
- **gotoEnd (Expand)** – jumps to the end of the text document.
- **gotoRange (TextRange, Expand)** – jumps to the specified `TextRange`-Object.
- **gotoStartOfWord (Expand)** – jumps to the start of the current word.
- **gotoEndOfWord (Expand)** – jumps to the end of the current word.
- **gotoNextWord (Expand)** – jumps to the start of the next word.
- **gotoPreviousWord (Expand)** – jumps to the start of the previous word.
- **isStartOfWord ()** – returns `True` if the `TextCursor` is at the start of a word.
- **isEndOfWord ()** – returns `True` if the `TextCursor` is at the end of a word.
- **gotoStartOfSentence (Expand)** – jumps to the start of the current sentence.
- **gotoEndOfSentence (Expand)** – jumps to the end of the current sentence.
- **gotoNextSentence (Expand)** – jumps to the start of the next sentence.
- **gotoPreviousSentence (Expand)** – jumps to the start of the previous sentence.
- **isStartOfSentence ()** – returns `True` if the `TextCursor` is at the start of a sentence.
- **isEndOfSentence ()** – returns `True` if the `TextCursor` is at the end of a sentence.
- **gotoStartOfParagraph (Expand)** – jumps to the start of the current paragraph.
- **gotoEndOfParagraph (Expand)** – jumps to the end of the current paragraph.
- **gotoNextParagraph (Expand)** – jumps to the start of the next paragraph.
- **gotoPreviousParagraph (Expand)** – jumps to the start of the previous paragraph.
- **isStartOfParagraph ()** – returns `True` if the `TextCursor` is at the start of a paragraph.
- **isEndOfParagraph ()** – returns `True` if the `TextCursor` is at the end of a paragraph.

The text is divided into sentences on the basis of sentence symbols. Periods are, for example, interpreted as symbols indicating the end of sentences.

The `Expand` parameter is a Boolean value which specifies whether the area passed over during navigation is to be highlighted. All navigation methods furthermore return a parameter which specifies whether the navigation was successful or whether the action was terminated for lack of text.

The following is a list of several methods for editing highlighted areas using a `TextCursor` and which also support the `com.sun.star.text.TextCursor` service:

- **`collapseToStart ()`** – resets the highlighting and positions the `TextCursor` at the start of the previously highlighted area.
- **`collapseToEnd ()`** – resets the highlighting and positions the `TextCursor` at the end of the previously highlighted area.
- **`isCollapsed ()`** – returns `True` if the `TextCursor` does not cover any highlighting at present.

Formatting Text with TextCursor

The `com.sun.star.text.TextCursor` service supports all the character and paragraph properties that were presented at the start of this chapter.

The following example shows how these can be used in conjunction with a `TextCursor`. It passes through a complete document and formats the first word of every sentence in bold type.

```
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
```

The example first creates a document object for the text that has just been opened. Then it iterates through the entire text, sentence by sentence, and highlights each of the first words and formats this in bold.

Retrieving and Modifying Text Contents

If a `TextCursor` contains a highlighted area, then this text is available by means of the `String` property of the `TextCursor` object. The following example uses the `String` property to display the first words of a sentence in a message box:

```
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
```

The first word of each sentence can be modified in the same way using the `String` property:

```
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
```

If the `TextCursor` contains a highlighted area, an assignment to the `String` property replaces this with the new text. If there is no highlighted area, the text is inserted at the present `TextCursor` position.

Inserting Control Codes

In some situations, it is not the actual text of a document, but rather its structure that needs modifying. StarOffice provides control codes for this purpose. These are inserted in the text and influence its structure. The control codes are defined in the `com.sun.star.text.ControlCharacter` group of constants. The following control codes are available in StarOffice:

- **PARAGRAPH_BREAK** – paragraph break.

- **LINE_BREAK** – line break within a paragraph.
- **SOFT_HYPHEN** – possible point for syllabification.
- **HARD_HYPHEN** – obligatory point for syllabification.
- **HARD_SPACE** – protected space that is not spread out or compressed in justified text.

To insert the control codes, you need not only the cursor but also the associated text document objects. The following example inserts a paragraph after the 20th character of a text:

```
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)
```

The `False` parameter in the call of the `insertControlCharacter` method ensures that the area currently highlighted by the `TextCursor` remains after the insert operation. If the `True` parameter is passed here, then `insertControlCharacter` replaces the current text.

Searching for Text Portions

In many instances, it is the case that a text is to be searched for a particular term and the corresponding point needs to be edited. All StarOffice documents provide a special interface for this purpose, and this interface always functions in accordance with the same principle: Before a search process, what is commonly referred to as a `SearchDescriptor` must first be created. This defines what StarOffice searches for in a document. A `SearchDescriptor` is an object which supports the `com.sun.star.util.SearchDescriptor` service and can be created by means of the `createSearchDescriptor` method of a document:

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

Once the `SearchDescriptor` has been created, it receives the text to be searched for:

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

In terms of its function, the `SearchDescriptor` is best compared with the search dialog from StarOffice. In a similar way to the search window, the settings needed for a search can be set in the `SearchDescriptor` object.

The properties are provided by the `com.sun.star.util.SearchDescriptor` service:

- **SearchBackwards (Boolean)** - searches through the text backward rather than forward.
- **SearchCaseSensitive (Boolean)** - takes uppercase and lowercase characters into consideration during the search.
- **SearchRegularExpression (Boolean)** - treats the search expression like a regular expression.
- **SearchStyles (Boolean)** - searches through the text for the specified paragraph template.
- **SearchWords (Boolean)** - only searches for complete words.

The StarOffice `SearchSimilarity` (or “fuzzy match”) function is also available in StarOffice Basic. With this function, StarOffice searches for an expression that may be similar to but not exactly the same as the search expression. The number of additional, deleted and modified characters for these expressions can be defined individually. Here are the associated properties of the `com.sun.star.util.SearchDescriptor` service:

- **SearchSimilarity (Boolean)** - performs a similarity search.
- **SearchSimilarityAdd (Short)** - number of characters which may be added for a similarity search.
- **SearchSimilarityExchange (Short)** - number of characters which may be replaced as part of a similarity search.
- **SearchSimilarityRemove (Short)** - number of characters which may be removed as part of a similarity search.
- **SearchSimilarityRelax (Boolean)** - takes all deviation rules into consideration at the same time for the search expression.

Once the `SearchDescriptor` has been prepared as requested, it can be applied to the text document. The StarOffice documents provide the `findFirst` and `findNext` methods for this purpose:

```
Found = Doc.findFirst (SearchDesc)

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

Loop
```

The example finds all matches in a loop and returns a `TextRange` object, which refers to the found text passage.

Example: Similarity Search

This example shows how a text can be searched for the word "turnover" and the results formatted in bold type. A similarity search is used so that not only the word "turnover", but also the plural form "turnovers" and declinations such as "turnover's" are found. The found expressions differ by up to two letters from the search expression:

```
Dim SearchDesc As Object
Dim Doc As Object

Doc = StarDesktop.CurrentComponent

SearchDesc = Doc.createSearchDescriptor
SearchDesc.SearchString="turnover"
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
```

Note – The basic idea of search and replace in StarOffice is comparable to that used in VBA. Both interfaces provide you with an object, through which the properties for searching and replacing can be defined. This object is then applied to the required text area in order to perform the action. Whereas the responsible auxiliary object in VBA can be reached through the Find property of the Range object, in StarOffice Basic it is created by the `createSearchDescriptor` or `createReplaceDescriptor` call of the document object. Even the search properties and methods available differ.

As in the old API from StarOffice, searching and replacing text in the new API is also performed using the document object. Whereas previously there was an object called `SearchSettings` especially for defining the search options, in the new object searches are now performed using a `SearchDescriptor` or `ReplaceDescriptor` object for automatically replacing text. These objects cover not only the options, but also the current search text and, if necessary, the associated text replacement. The descriptor objects are created using the document object, completed in accordance with the relevant requests, and then transferred back to the document object as parameters for the search methods.

Replacing Text Portions

Just as with the search function, the replacement function from StarOffice is also available in StarOffice Basic. The two functions are handled identically. A special object which records the parameters for the process is also first needed for a replacement process. It is called a `ReplaceDescriptor` and supports the `com.sun.star.util.ReplaceDescriptor` service. All the properties of the `SearchDescriptor` described in the previous paragraph are also supported by `ReplaceDescriptor`. For example, during a replacement process, case sensitivity can also be activated and deactivated, and similarity searches can be performed.

The following example demonstrates the use of `ReplaceDescriptors` for a search within a StarOffice document.

```
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
```

The expressions for searching and replacing are set using the `SearchString` and `ReplaceString` properties of the `ReplaceDescriptors`. The actual replacement process is finally implemented using the `replaceAll` method of the document object, which replaces all occurrences of the search expression.

Example: searching and replacing text with regular expressions

The replacement function of StarOffice is particularly effective when used in conjunction with regular expressions. These provide the option of defining a variable search expression with place holders and special characters rather than a fixed value.

The regular expressions supported by StarOffice are described in detail in the online help section for StarOffice. Here are a few examples:

- A period within a search expression stands for any character. The search expression `sh.rt` therefore can stand for both for `shirt` and for `short`.

- The character `^` marks the start of a paragraph. All occurrences of the name `Peter` that are at the start of a paragraph can therefore be found using the search expression `^Peter`.
- The character `$` marks a paragraph end. All occurrences of the name `Peter` that are at the end of a paragraph can therefore be found using the search expression `Peter$`.
- A `*` indicates that the preceding character may be repeated any number of times. It can be combined with the period as a place holder for any character. The `temper.*e` expression, for example, can stand for the expressions `temperance` and `temperature`.

The following example shows how all empty lines in a text document can be removed with the help of the regular expression `^$`:

```
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)
```

Text Documents: More than Just Text

So far, this chapter has only dealt with text paragraphs and their portions. But text documents may also contain other objects. These include tables, drawings, text fields and directories. All of these objects can be anchored to any point within a text.

Thanks to these common features, all of these objects in StarOffice support a common basic service called `com.sun.star.text.TextContent`. This provides the following properties:

- **AnchorType (Enum)** – determines the anchor type of a `TextContent` object (default values in accordance with `com.sun.star.text.TextContentAnchorTypeenumeration`).
- **AnchorTypes (sequence of Enum)** – enumeration of all `AnchorTypes` which support a special `TextContent` object.
- **TextWrap (Enum)** – determines the text wrap type around a `TextContent` object (default values in accordance with `com.sun.star.text.WrapTextModeenumeration`).

The `TextContent` objects also share some methods – in particular, those for creating, inserting and deleting objects.

- A new `TextContent` object is **created** using the `createInstance` method of the document object.
- An object is **inserted** using the `insertTextContent` method of the text object.
- `TextContent` objects are **deleted** using the `removeTextContent` method.

You will find a range of examples which use these methods in the following sections.

Tables

The following example creates a table with the help of the `createInstance` method described previously.

```
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)
```

Once created, the table is set to the number of rows and columns requested using an `initialize` call and then inserted in the text document using `insertTextContent`.

As can be seen in the example, the `insertTextContent` method expects not only the `Content` object to be inserted, but two other parameters:

- a `Cursor` object which determines the insert position
- a Boolean variable which specifies whether the `Content` object is to replace the current selection of the cursor (`True` value) or is to be inserted before the current selection in the text (`False`)

Note – When creating and inserting tables in a text document, objects similar to those available in VBA are used in StarOffice Basic: The document object and a `TextCursor` object in StarOffice Basic or the `Range` object as the VBA counterpart. Whereas the `Document.Tables.Add` method takes on the task of creating and setting the table in VBA, this is created in StarOffice Basic in accordance with the previous example using `createInstance`, initialized and inserted in the document through `insertTextContent`.

The tables inserted in a text document can be determined using a simple loop. The method of the `getTextTables()` of the text document object is used for this purpose:

```
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)
    ' Editing table
Next I
```

Note – Text tables are available in StarOffice 8 through the `TextTables` list of the document object. This takes the place of the former `tables` list provided in the `Selection` object. The previous example shows how a text table can be created. The options for accessing text tables are described in the following section.

Editing Tables

A table consists of individual rows. These in turn contain the various cells. Strictly speaking, there are no table columns in StarOffice. These are produced implicitly by arranging the rows (one under another) next to one another. To simplify access to the tables, StarOffice, however, provides some methods which operate using columns. These are useful if no cells have been merged in the table.

Let us first take the properties of the table itself. These are defined in the `com.sun.star.text.TextTable` service. Here is an list of the most important properties of the table object:

- **BackColor (Long)** – background color of table.
- **BottomMargin (Long)** – bottom margin in 100ths of a millimeter.
- **LeftMargin (Long)** – left margin in 100ths of a millimeter.
- **RightMargin (Long)** – right margin in 100ths of a millimeter.
- **TopMargin (Long)** – top margin in 100ths of a millimeter.
- **RepeatHeadline (Boolean)** – table header is repeated on every page.
- **Width (Long)** – absolute width of the table in 100ths of a millimeter.

Rows

A table consists of a list containing rows. The following example shows how the rows of a table can be retrieved and formatted.

```
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.BackgroundColor = &HFF00FF
Next

```

The example first creates a list containing all rows using a `Table.getRows` call. The `getCount` and `getByIndex` methods allow the list to be further processed and belongs to the `com.sun.star.table.XtableRows` interface. The `getByIndex` method returns a row object, which supports the `com.sun.star.text.TextTableRow` service.

Here are the central methods of the `com.sun.star.table.XtableRows` interface:

- **getByIndex(Integer)** – returns a row object for the specified index.
- **getCount()** – returns the number of row objects.
- **insertByIndex(Index, Count)** – inserts `Count` rows in the table as of the `Index` position.
- **removeByIndex(Index, Count)** – deletes `Count` rows from the table as of the `Index` position.

Whereas the `getByIndex` and `getCount` methods are available in all tables, the `insertByIndex` and `removeByIndex` methods can only be used in tables that do not contain merged cells.

The `com.sun.star.text.TextTableRow` service provides the following properties:

- **BackColor (Long)** – background color of row.
- **Height (Long)** – height of line in 100ths of a millimeter.
- **IsAutoHeight (Boolean)** – table height is dynamically adapted to the content.
- **VertOrient (const)** – vertical orientation of the text frame — details on vertical orientation of the text within the table (values in accordance with `com.sun.star.text.VertOrientation`)

Columns

Columns are accessed in the same way as rows, using the `getByIndex`, `getCount`, `insertByIndex`, and `removeByIndex` methods on the `Column` object, which is reached through `getColumns`. They can, however, only be used in tables that do not contain merged table cells. Cells cannot be formatted by column in StarOffice Basic. To do so, the method of formatting individual table cells must be used.

Cells

Each cell of a StarOffice-document has a unique name. If the cursor of StarOffice is in a cell, then the name of that cell can be seen in the status bar. The top left cell is usually called A1 and the bottom right row is usually called Xn, where X stands for the letters of the top column and n for the numbers of the last row. The cell objects are available through the `getCellByName()` method of the table object. The following example shows a loop that passes through all the cells of a table and enters the corresponding row and column numbers into the cells.

```
Dim Doc As Object
Dim Table As Object
Dim Cursor As Object
Dim Rows As Object
DimRowIndex 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

ForRowIndex = 1 To Rows.getCount()
  ForColIndex = 1 To Cols.getCount()
    CellName = Chr(64 + ColIndex) & RowIndex
    Cell = Table.getCellByName(CellName)
    Cell.String = "row: " & CStr(RowIndex) + ", column: " & CStr(ColIndex)
  Next
Next
```

A table cell is comparable with a standard text. It supports the `createTextCursor` interface for creating an associated `TextCursor` object.

```
CellCursor = Cell.createTextCursor()
```

All formatting options for individual characters and paragraphs are therefore automatically available.

The following example searches through all tables of a text document and applies the right-align format to all cells with numerical values by means of the corresponding paragraph property.

```
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
```

The example creates a `TextTables` list containing all tables of a text that are traversed in a loop. `StarOffice` then creates a list of the associated cell names for each of these tables. There are passed through in turn in a loop. If a cell contains a numerical value, then the example changes the formatting correspondingly. To do this, it first creates a `TextCursor` object which makes reference to the content of the table cell and then adapts the paragraph properties of the table cell.

Text Frames

Text frames are considered to be `TextContent` objects, just like tables and graphs. They may essentially consist of standard text, but can be placed at any position on a page and are not included in the text flow.

As with all `TextContent` objects, a distinction is also made with text frames between the actual creation and insertion in the document.

```
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)

```

The text frame is created using the `createInstance` method of the document object. The text frame created in this way can then be inserted in the document using the `insertTextContent` method of the `Text` object. In so doing, the name of the proper `com.sun.star.text.TextFrame` service should be specified.

The text frame's insert position is determined by a `Cursor` object, which is also executed when inserted.

Note – Text frames are StarOffice's counterpart to the position frame used in Word. Whereas VBA uses the `Document.Frames.Add` method for this purpose, creation in VBA is performed using the previous procedure with the aid of a `TextCursor` as well as the `createInstance` method of the document object.

Text frame objects provide a range of properties with which the position and behavior of the frame can be influenced. The majority of these properties are defined in the `com.sun.star.text.BaseFrameProperties` service, which is also supported by each `TextFrame` service. The central properties are:

- **BackColor (Long)** – background color of the text frame.
- **BottomMargin (Long)** – bottom margin in 100ths of a millimeter.
- **LeftMargin (Long)** – left margin in 100ths of a millimeter.
- **RightMargin (Long)** – right margin in 100ths of a millimeter.
- **TopMargin (Long)** – top margin in 100ths of a millimeter.
- **Height (Long)** – height of text frame in 100ths of a millimeter.
- **Width (Long)** – width of text frame in 100ths of a millimeter.
- **HoriOrient (const)** – horizontal orientation of text frame (in accordance with `com.sun.star.text.HoriOrientation`).
- **VertOrient (const)** – vertical orientation of text frame (in accordance with `com.sun.star.text.VertOrientation`).

The following example creates a text frame using the properties described previously:

```

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)

```

The example creates a `TextCursor` as the insertion mark for the text frame. This is positioned between the first and second word of the text. The text frame is created using `Doc.createInstance`. The properties of the text frame objects are set to the starting values required.

The interaction between the `AnchorType` (from the `TextContent` Service) and `VertOrient` (from the `BaseFrameProperties` Service) properties should be noted here. `AnchorType` receives the `AS_CHARACTER` value. The text frame is therefore inserted directly in the text flow and behaves like a character. It can, for example, be moved into the next line if a line break occurs. The `LINE_TOP` value of the `VertOrient` property ensures that the upper edge of the text frame is at the same height as the upper edge of the character.

Once initialization is complete, the text frame is finally inserted in the text document using a call from `insertTextContent`.

To edit the content of a text frame, the user uses the `TextCursor`, which has already been mentioned numerous times and is also available for text frames.

```

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 = "This is a small Test!"
```

The example creates a text frame, inserts this in the current document and opens a `TextCursor` for the text frame. This cursor is used to set the frame font to bold type and to set the paragraph orientation to centered. The text frame is finally assigned the "This is a small test!" string.

Text Fields

Text fields are `TextContent` objects because they provide additional logic extending beyond pure text. Text fields can be inserted in a text document using the same methods as those used for other `TextContent` objects:

```
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)
```

The example inserts a text field with the current date at the start of the current text document. The `True` value of the `IsDate` property results in only the date and not time being displayed. The `False` value for `IsFixed` ensures that the date is automatically updated when the document is opened.

Note – While the type of a field in VBA is specified by a parameter of the `Document.Fields.Add` method, the name of the service that is responsible for the field type in question defines it in StarOffice Basic.

In the past, text fields were accessed using a whole range of methods that StarOffice made available in the old `Selection` object (for example `InsertField`, `DeleteUserField`, `SetCurField`).

In StarOffice 8, the fields are administered using an object-oriented concept. To create a text field, a text field of the type required should first be created and initialized using the properties required. The text field is then inserted in the document using the `insertTextContent` method. A corresponding source text can be seen in the previous example. The most important field types and their properties are described in the following sections.

In addition to inserting text fields, searching a document for the fields can also be an important task. The following example shows how all text fields of a text document can be traversed in a loop and checked for their relevant type.

```

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 "Date/time"
    ElseIf TextField.supportsService("com.sun.star.text.TextField.Annotation") Then
        MsgBox "Annotation"
    Else
        MsgBox "unknown"
    End If

Wend

```

The starting point for establishing the text fields present is the `TextFields` list of the document object. The example creates an `Enumeration` object on the basis of this list, with which all text fields can be queried in turn in a loop. The text fields found are checked for the service supported using the `supportsService` method. If the field proves to be a date/time field or an annotation, then the corresponding field type is displayed in an information box. If on the other hand, the example encounters another field, then it displays the information “unknown”.

Below, you will find a list of the most important text fields and their associated properties. A complete list of all text fields is provided in the API reference in the `com.sun.star.text.TextField` module. (When listing the service name of a text field, uppercase and lowercase characters should be used in StarOffice Basic, as in the previous example.)

Number of Pages, Words and Characters

The text fields

- `com.sun.star.text.TextField.PageCount`
- `com.sun.star.text.TextField.WordCount`
- `com.sun.star.text.TextField.CharacterCount`

return the number of pages, words, or characters of a text. They support the following property:

- **NumberingType (const)** – numbering format (guidelines in accordance with constants from `com.sun.star.style.NumberingType`).

Current Page

The number of the current page can be inserted in a document using the `com.sun.star.text.TextField.PageNumber` text field. The following properties can be specified:

- **NumberingType (const)** - number format (guidelines in accordance with constants from `com.sun.star.style.NumberingType`).
- **Offset (short)** – offset added to the number of pages (negative specification also possible).

The following example shows how the number of pages can be inserted into the footer of a document.

```
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)
```

The example first creates a text field which supports the `com.sun.star.text.TextField.PageNumber` service. Since the header and footer lines are defined as part of the page templates of StarOffice, this is initially established using the list of all `PageStyles`.

To ensure that the footer line is visible, the `FooterIsOn` property is set to `True`. The text field is then inserted in the document using the associated text object of the left-hand footer line.

Annotations

Annotation fields (`com.sun.star.text.TextField.Annotation`) can be seen by means of a small yellow symbol in the text. Clicking on this symbol opens a text field, in which a comment on the current point in the text can be recorded. An annotation field has the following properties.

- **Author (String)** - name of author.

- **Content (String)** - comment text.
- **Date (Date)** - date on which annotation is written.

Date / Time

A date / time field (`com.sun.star.text.TextField.DateTime`) represents the current date or the current time. It supports the following properties:

- **IsFixed (Boolean)** – if `True`, the time details of the insertion remain unchanged, if `False`, these are updated each time the document is opened.
- **IsDate (Boolean)** – if `True`, the field displays the current date, otherwise the current time.
- **DateTimeValue (struct)** – current content of field (`com.sun.star.util.DateTime` structure)
- **NumberFormat (const)** – format in which the time or date is depicted.

Chapter Name / Number

The name of the current chapter is available through a text field of the `com.sun.star.text.TextField.Chapter` type. The form can be defined using two properties.

- **ChapterFormat (const)** – determines whether the chapter name or the chapter number is depicted (in accordance with `com.sun.star.text.ChapterFormat`)
- **Level (Integer)** – determines the chapter level whose name and/or chapter number is to be displayed. The value 0 stands for highest level available.

Bookmarks

Bookmarks (Service `com.sun.star.text.Bookmark`) are `TextContent` objects. Bookmarks are created and inserted using the concept already described previously:

```
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 = "My bookmarks"
Doc.Text.insertTextContent(Cursor, Bookmark, True)
```

The example creates a `Cursor`, which marks the insert position of the bookmark and then the actual bookmark object (`Bookmark`). The bookmark is then assigned a name and is inserted in the document through `insertTextContent` at the cursor position.

The bookmarks of a text are accessed through a list called `Bookmarks`. The bookmarks can either be accessed by their number or their name.

The following example shows how a bookmark can be found within a text, and a text inserted at its position.

```
Dim Doc As Object
Dim Bookmark As Object
Dim Cursor As Object

Doc = StarDesktop.CurrentComponent

Bookmark = Doc.Bookmarks.getByName("My bookmarks")

Cursor = Doc.Text.createTextCursorByRange(Bookmark.Anchor)
Cursor.String = "Here is the bookmark"
```

In this example, the `getByName` method is used to find the bookmark required by means of its name. The `createTextCursorByRange` call then creates a `Cursor`, which is positioned at the anchor position of the bookmark. The cursor then inserts the text required at this point.

Spreadsheet Documents

StarOffice Basic provides an extensive interface for program-controlled creation and editing of spreadsheets. This chapter describes how to control the relevant services, methods and properties of spreadsheet documents.

The first section addresses the basic structure of spreadsheet documents and shows you how to access and to edit the contents of individual cells.

The second section concentrates on how to edit spreadsheets efficiently by focusing on cell areas and the options for searching and replacing cell contents.

Note – The Range object allows you to address any table area and has been extended in the new API.

The Structure of Table-Based Documents (Spreadsheets)

The document object of a spreadsheet is based on the `com.sun.star.sheet.SpreadsheetDocument` service. Each of these documents may contain several spreadsheets. In this guide, a *table-based document* or *spreadsheet document* is the entire document, whereas a *spreadsheet* (or *sheet* for short) is a sheet (table) in the document.

Note – Different terminology for spreadsheets and their content is used in VBA and StarOffice Basic. Whereas the document object in VBA is called a *Workbook* and its individual pages *Worksheets*, they are called *SpreadsheetDocument* and *Sheet* in StarOffice Basic.

Spreadsheets

You can access the individual sheets of a spreadsheet document through the `Sheets` list.

The following examples show you how to access a sheet either through its number or its name.

Example 1: access by means of the number (numbering begins with 0)

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets (0)
```

Example 2: access by means of the name

```
Dim Doc As Object
Dim Sheet As Object

Doc = StarDesktop.CurrentComponent
Sheet = Doc.Sheets.getByName("Sheet 1")
```

In the first example, the sheet is accessed by its number (counting begins at 0). In the second example, the sheet is accessed by its name and the `getByName` method.

The `Sheet` object that is obtained by the `getByName` method supports the `com.sun.star.sheet.Spreadsheet` service. In addition to providing several interfaces for editing the content, this service provides the following properties:

- **IsVisible (Boolean)** – the spreadsheet is visible.
- **PageStyle (String)** – name of the page template for the spreadsheet.

Creating, Deleting and Renaming Sheets

The `Sheets` list for a spreadsheet document is also used to create, delete, and rename individual sheets. The following example uses the `hasByName` method to check if a sheet called *MySheet* exists. If it does, the method determines a corresponding object reference by using the `getByName` method and then saves the reference in a variable in `Sheet`. If the corresponding sheet does not exist, it is created by the `createInstance` call and inserted in the spreadsheet document by the `insertByName` method.

```

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

```

The `GetByName` and `InsertByName` methods are from the `com.sun.star.container.XNameContainer` interface as described in [Chapter 4](#).

Rows and Columns

Each sheet contains a list of its rows and columns. These are available through the `Rows` and `Columns` properties of the spreadsheet object and support the `com.sun.star.table.TableColumns` and/or `com.sun.star.table.TableRows` services.

The following example creates two objects that reference the first row and the first column of a sheet and stores the references in the `FirstCol` and `FirstRow` object variables.

```

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)

```

The column objects support the `com.sun.star.table.TableColumn` service that has the following properties:

- **Width (long)** – width of a column in hundredths of a millimeter.
- **OptimalWidth (Boolean)** – sets a column to its optimum width.
- **IsVisible (Boolean)** – displays a column.
- **IsStartOfNewPage (Boolean)** – when printing, creates a page break before a column.

The width of a column is only optimized when the `OptimalWidth` property is set to `True`. If the width of an individual cell is changed, the width of the column that contains the cell is not changed. In terms of functionality, `OptimalWidth` is more of a method than a property.

The row objects are based on the `com.sun.star.table.RowColumn` service that has the following properties:

- **Height (long)** - height of the row in 100ths of a millimeter.
- **OptimalHeight (Boolean)** - sets the row to its optimum height.
- **IsVisible (Boolean)** - displays the row.
- **IsStartOfNewPage (Boolean)** - when printing, creates a page break before the row.

If the `OptimalHeight` property of a row is set to the `True`, the row height changes automatically when the height of a cell in the row is changed. Automatic optimization continues until the row is assigned an absolute height through the `Height` property.

The following example activates the automatic height optimization for the first five rows in the sheet and makes the second column invisible.

```
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
```

Note – The `Rows` and `Columns` lists can be accessed through an index in StarOffice Basic. Unlike in VBA, the first column has the index 0 and not the index 1.

Inserting and Deleting Rows and Columns

The `Rows` and `Columns` objects of a sheet can access existing rows and columns as well as insert and delete them.

```
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)
```

This example uses the `insertByIndex` method to insert a new column into the fourth column position in the sheet (index 3 - numbering starts at 0). The second parameter specifies the number of columns to be inserted (in this example: one).

The `removeByIndex` method deletes the sixth column (index 5). Again, the second parameter specifies the number of columns that you want to delete.

The methods for inserting and deleting rows use the `Rows` object function in the same way as the methods shown for editing columns using the `Columns` object.

Cells

A spreadsheet consists of a two-dimensional list containing cells. Each cell is defined by its X and Y-position with respect to the top left cell which has the position (0,0).

The following example creates an object that references the top left cell and inserts a text in the cell:

```
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"
```

In addition to numerical coordinates, each cell in a sheet has a name, for example, the top left cell (0,0) of a spreadsheet is called A1. The letter A stands for the column and the number 1 for the row. It is important that the *name* and *position* of a cell are not confused because row counting for names begins with 1 but the counting for position begins with 0.

In StarOffice, a table cell can be empty or contain text, numbers, or formulas. The cell type is not determined by the content that is saved in the cell, but rather the object property which was used for its entry. Numbers can be inserted and called up with the `Value` property, text with the `String` property, and formulas with the `Formula` property.

```
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"
```

The example inserts one number, one text, and one formula in the fields A1 to A3.

Note – The `Value`, `String`, and `Formula` properties supersede the `PutCell` method for setting the values of a table cell.

StarOffice treats cell content that is entered using the `String` property as text, even if the content is a number. Numbers are left-aligned in the cell instead of right-aligned. You should also note the difference between text and numbers when you use formulas:

```
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
```

Although cell A1 contains the value 100 and cell A2 contains the value 1000, the A1+A2 formula returns the value 100. This is because the contents of cell A2 were entered as a string and not as a number.

To check if the contents of a cell contains a number or a string, use the `Type` property:

```
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

```

The `Cell.Type` property returns a value for the `com.sun.star.table.CellContentType` enumeration which identifies the contents type of a cell. The possible values are:

- **EMPTY** - no value
- **VALUE** - number
- **TEXT** - strings
- **FORMULA** - formula

Inserting, Deleting, Copying and Moving Cells

In addition to directly modifying cell content, StarOffice Calc also provides an interface that allows you to insert, delete, copy, or merge cells. The interface (`com.sun.star.sheet.XRangeMovement`) is available through the spreadsheet object and provides four methods for modifying cell content.

The `insertCell` method is used to insert cells into a sheet.

```

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)

```

This example inserts a cells range that is two rows by two columns in size into the second column and row (each bear the number 1) of the first sheet (number 0) in the spreadsheet. Any existing values in the specified cell range are moved below the range.

To define the cell range that you want to insert, use the `com.sun.star.table.CellRangeAddress` structure. The following values are included in this structure:

- **Sheet (short)** - number of the sheet (numbering begins with 0).
- **StartColumn (long)** – first column in the cell range (numbering begins with 0).
- **StartRow (long)** – first row in the cell range (numbering begins with 0).
- **EndColumn (long)** – final column in the cell range (numbering begins with 0).
- **EndRow (long)** – final row in the cell range (numbering begins with 0).

The completed `CellRangeAddress` structure must be passed as the first parameter to the `insertCells` method. The second parameter of `insertCells` contains a value of the `com.sun.star.sheet.CellInsertMode` enumeration and defines what is to be done with the values that are located in front of the insert position. The `CellInsertMode` enumeration recognizes the following values:

- **NONE** – the current values remain in their present position.
- **DOWN** – the cells at and under the insert position are moved downwards.
- **RIGHT** – the cells at and to the right of the insert position are moved to the right.
- **ROWS** – the rows after the insert position are moved downwards.
- **COLUMNS** – the columns after the insert position are moved to the right.

The `removeRange` method is the counterpart to the `insertCells` method. This method deletes the range that is defined in the `CellRangeAddress` structure from the sheet.

```
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)
```

This example removes the B2:C3 cell range from the sheet and then shifts the underlying cells up by two rows. The type of removal is defined by one of the following values from the `com.sun.star.sheet.CellDeleteMode` enumeration:

- **NONE** – the current values remain in their current position.
- **UP** – the cells at and below the insert position are moved upwards.
- **LEFT** – the cells at and to the right of the insert position are moved to the left.
- **ROWS** – the rows after the insert position are moved upwards.
- **COLUMNS** – the columns after the insert position are moved to the left.

The `XRangeMovement` interface provides two additional methods for moving (`moveRange`) or copying (`copyRange`) cell ranges. The following example moves the `B2 : C3` range so that the range starts at 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)
```

In addition to the `CellRangeAddress` structure, the `moveRange` method expects a `com.sun.star.table.CellAddress` structure to define the origin of the move's target region. The `CellAddress` method provides the following values:

- **Sheet (short)** – number of the spreadsheet (numbering begins with 0).
- **Column (long)** – number of the addressed column (numbering begins with 0).
- **Row (long)** – number of the addressed row (numbering begins with 0).

The cell contents in the target range are always overwritten by the `moveRange` method. Unlike in the `InsertCells` method, a parameter for performing automatic moves is not provided in the `removeRange` method.

The `copyRange` method functions in the same way as the `moveRange` method, except that `copyRange` inserts a copy of the cell range instead of moving it.

Note – In terms of their function, the StarOffice Basic `insertCell`, `removeRange`, and `copyRange` methods are comparable with the VBA `Range.Insert`, `Range.Delete`, and `Range.Copy` methods. Whereas in VBA, the methods are applied to the corresponding `Range` object, in StarOffice Basic they are applied to the associated `Sheet` object.

Formatting

A spreadsheet document provides properties and methods for formatting cells and pages.

Cell Properties

There are numerous options for formatting cells, such as specifying the font type and size for text. Each cell supports the `com.sun.star.style.CharacterProperties` and `com.sun.star.style.ParagraphProperties` services, the main properties of which are described in [Chapter 6](#). Special cell formatting is handled by the `com.sun.star.table.CellProperties` service. The main properties of this service are described in the following sections.

You can apply all of the named properties to individual cells and to cell ranges.

Note – The `CellProperties` object in the StarOffice API is comparable with the `Interior` object from VBA which also defines cell-specific properties.

Background Color and Shadows

The `com.sun.star.table.CellProperties` service provides the following properties for defining background colors and shadows:

- **CellBackColor (Long)** – background color of the table cell.
- **IsCellBackgroundTransparent (Boolean)** – sets the background color to transparent.
- **ShadowFormat (struct)** – specifies the shadow for cells (structure in accordance with `com.sun.star.table.ShadowFormat`).

The `com.sun.star.table.ShadowFormat` structure and the detailed specifications for cell shadows have the following structure:

- **Location (enum)** – position of shadow (value from the `com.sun.star.table.ShadowLocation` structure).
- **ShadowWidth (Short)** – size of shadow in hundredths of a millimeter.
- **IsTransparent (Boolean)** – sets the shadow to transparent.
- **Color (Long)** – color of shadow.

The following example writes the number 1000 to the B2 cell, changes the background color to red using the `CellBackColor` property, and then creates a light gray shadow for the cell that is moved 1 mm to the left and down.

```
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

```

Justification

StarOffice provides various functions that allow you to change the justification of a text in a table cell.

The following properties define the horizontal and vertical justification of a text:

- **HoriJustify (enum)** – horizontal justification of the text (value from `com.sun.star.table.CellHoriJustify`)
- **VertJustify (enum)** - vertical justification of the text (value from `com.sun.star.table.CellVertJustify`)
- **Orientation (enum)** – orientation of text (value in accordance with `com.sun.star.table.CellOrientation`)
- **IsTextWrapped (Boolean)** – permits automatic line breaks within the cell
- **RotateAngle (Long)** – angle of rotation of text in hundredths of a degree

The following example shows how you can "stack" the contents of a cell so that the individual characters are printed one under another in the top left corner of the cell. The characters are not rotated.

```

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

```

Number, Date and Text Format

StarOffice provides a whole range of predefined date and time formats. Each of these formats has an internal number that is used to assign the format to cells using the `NumberFormat` property. StarOffice provides the `queryKey` and `addNew` methods so that you can access existing number formats as well as create your own number formats. The methods are accessed through the following object call:

```
NumberFormats = Doc.NumberFormats
```

A format is specified using a format string that is structured in a similar way to the `format` function of StarOffice Basic. However there is one major difference: whereas the `command format` expects English abbreviations and decimal points or characters as thousands separators, the country-specified abbreviations must be used for the structure of a `command format` for the `NumberFormats` object.

The following example formats the B2 cell so that numbers are displayed with three decimal places and use commas as a thousands separator.

```
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
```

The **Format Cells** dialog in StarOffice Calc provides an overview of the different formatting options for cells.

Page Properties

Page properties are the formatting options that position document content on a page as well as visual elements that are repeated page after page. These include

- Paper formats
- Page margins
- Headers and footers.

The procedure for defining page formats differs from other forms of formatting. Whereas cell, paragraph, and character element can be directly, page formats can also be defined and indirectly applied using page styles. For example, headers or footers are added to the page style.

The following sections describe the main formatting options for spreadsheet pages. Many of the styles that are described are also available for text documents. The page properties that are valid for both types of documents are defined in the `com.sun.star.style.PageProperties` service. The page properties that only apply to spreadsheet documents are defined in the `com.sun.star.sheet.TablePageStyle` service.

Note – The page properties (page margins, borders, and so on) for a Microsoft Office document are defined by means of a `PageSetup` object at the `Worksheet` object (Excel) or `Document` object (Word) level. In StarOffice, these properties are defined using a page style which in turn is linked to the associated document.

Page Background

The `com.sun.star.style.PageProperties` service defines the following properties of a pages background:

- **BackColor (long)** – color of background
- **BackGraphicURL (String)** – URL of the background graphics that you want to use
- **BackGraphicFilter (String)** – name of the filter for interpreting the background graphics
- **BackGraphicLocation (Enum)** – position of the background graphics (value according to enumeration)
- **BackTransparent (Boolean)** - makes the background transparent

Page Format

The page format is defined using the following properties of the `com.sun.star.style.PageProperties` service:

- **IsLandscape (Boolean)** – landscape format
- **Width (long)** – width of page in hundredths of a millimeter
- **Height (long)** – height of page in hundredths of a millimeter
- **PrinterPaperTray (String)** – name of the printer paper tray that you want to use

The following example sets the page size of the "Default" page style to the DIN A5 landscape format (height 14.8 cm, width 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
```

Page Margin, Border, and Shadow

The `com.sun.star.style.PageProperties` service provides the following properties for adjusting page margins as well as borders and shadows:

- **LeftMargin (long)** – width of the left hand page margin in hundredths of a millimeter
- **RightMargin (long)** – width of the right hand page margin in hundredths of a millimeter.
- **TopMargin (long)** – width of the top page margin in hundredths of a millimeter
- **BottomMargin (long)** – width of the bottom page margin in hundredths of a millimeter
- **LeftBorder (struct)** – specifications for left-hand line of page border
(`com.sun.star.table.BorderLine` structure)
- **RightBorder (struct)** – specifications for right-hand line of page border
(`com.sun.star.table.BorderLine` structure)
- **TopBorder (struct)** – specifications for top line of page border
(`com.sun.star.table.BorderLine` structure)
- **BottomBorder (struct)** – specifications for bottom line of page border
(`com.sun.star.table.BorderLine` structure)
- **LeftBorderDistance (long)** – distance between left-hand page border and page content in hundredths of a millimeter
- **RightBorderDistance (long)** – distance between right-hand page border and page content in hundredths of a millimeter
- **TopBorderDistance (long)** – distance between top page border and page content in hundredths of a millimeter
- **BottomBorderDistance (long)** – distance between bottom page border and page content in hundredths of a millimeter

- **ShadowFormat (struct)** – specifications for shadow of content area of page (`com.sun.star.table.ShadowFormat` structure)

The following example sets the left and right-hand borders of the "Default" page style to 1 centimeter.

```
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
```

Headers and Footers

The headers and footers of a document form part of the page properties and are defined using the `com.sun.star.style.PageProperties` service. The properties for formatting headers are:

- **HeaderIsOn (Boolean)** – header is activated
- **HeaderLeftMargin (long)** – distance between header and left-hand page margin in hundredths of a millimeter.
- **HeaderRightMargin (long)** – distance between header and right-hand page margin in hundredths of a millimeter
- **HeaderBodyDistance (long)** – distance between header and main body of document in hundredths of a millimeter
- **HeaderHeight (long)** – height of header in hundredths of a millimeter
- **HeaderIsDynamicHeight (Boolean)** – height of header is automatically adapted to content
- **HeaderLeftBorder (struct)** – details of the left-hand border of frame around header (`com.sun.star.table.BorderLine` structure)
- **HeaderRightBorder (struct)** – details of the right-hand border of frame around header (`com.sun.star.table.BorderLine` structure)
- **HeaderTopBorder (struct)** – details of the top line of the border around header (`com.sun.star.table.BorderLine` structure)
- **HeaderBottomBorder (struct)** – details of the bottom line of the border around header (`com.sun.star.table.BorderLine` structure)
- **HeaderLeftBorderDistance (long)** – distance between left-hand border and content of header in hundredths of a millimeter

- **HeaderRightBorderDistance (long)** – distance between right-hand border and content of header in hundredths of a millimeter
- **HeaderTopBorderDistance (long)** – distance between top border and content of header in hundredths of a millimeter
- **HeaderBottomBorderDistance (long)** – distance between bottom border and content of header in hundredths of a millimeter
- **HeaderIsShared (Boolean)** – headers on even and odd pages have the same content (refer to `HeaderText`, `HeaderTextLeft`, and `HeaderTextRight`)
- **HeaderBackColor (long)** – background color of header
- **HeaderBackGraphicURL (String)** – URL of the background graphics that you want to use
- **HeaderBackGraphicFilter (String)** – name of the filter for interpreting the background graphics for the header
- **HeaderBackGraphicLocation (Enum)** – position of the background graphics for the header (value according to `com.sun.star.style.GraphicLocation` enumeration)
- **HeaderBackTransparent (Boolean)** – shows the background of the header as transparent
- **HeaderShadowFormat (struct)** – details of shadow of header (`com.sun.star.table.ShadowFormat` structure)

The properties for formatting footers are:

- **FooterIsOn (Boolean)** – footer is activated
- **FooterLeftMargin (long)** – distance between footer and left-hand page margin in hundredths of a millimeter
- **FooterRightMargin (long)** – distance between footer and right-hand page margin in hundredths of a millimeter
- **FooterBodyDistance (long)** – distance between footer and main body of document in hundredths of a millimeter
- **FooterHeight (long)** – height of footer in hundredths of a millimeter
- **FooterIsDynamicHeight (Boolean)** – height of footer is adapted automatically to the content
- **FooterLeftBorder (struct)** – details of left-hand line of border around footer (`com.sun.star.table.BorderLine` structure)
- **FooterRightBorder (struct)** – details of right-hand line of border around footer (`com.sun.star.table.BorderLine` structure)
- **FooterTopBorder (struct)** – details of top line of border around footer (`com.sun.star.table.BorderLine` structure)
- **FooterBottomBorder (struct)** – details of bottom line of border around footer (`com.sun.star.table.BorderLine` structure)
- **FooterLeftBorderDistance (long)** – distance between left-hand border and content of footer in hundredths of a millimeter

- **FooterRightBorderDistance (long)** – distance between right-hand border and content of footer in hundredths of a millimeter
- **FooterTopBorderDistance (long)** – distance between top border and content of footer in hundredths of a millimeter
- **FooterBottomBorderDistance (long)** – distance between bottom border and content of footer in hundredths of a millimeter
- **FooterIsShared (Boolean)** – the footers on the even and odd pages have the same content (refer to `FooterText`, `FooterTextLeft`, and `FooterTextRight`).
- **FooterBackColor (long)** – background color of footer
- **FooterBackGraphicURL (String)** – URL of the background graphics that you want to use
- **FooterBackGraphicFilter (String)** – name of the filter for interpreting the background graphics for the footer
- **FooterBackGraphicLocation (Enum)** – position of background graphics for the footer (value according to `com.sun.star.style.GraphicLocation` enumeration)
- **FooterBackTransparent (Boolean)** – shows the background of the footer as transparent
- **FooterShadowFormat (struct)** – details of shadow of footer (`com.sun.star.table.ShadowFormat` structure)

Changing the Text of Headers and Footers

The content of headers and footers in a spreadsheet is accessed through the following properties:

- **LeftPageHeaderContent (Object)** - content of headers for even pages (`com.sun.star.sheet.HeaderFooterContent` service)
- **RightPageHeaderContent (Object)** - content of headers for odd pages (`com.sun.star.sheet.HeaderFooterContent` service)
- **LeftPageFooterContent (Object)**- content of footers for even pages (`com.sun.star.sheet.HeaderFooterContent` service)
- **RightPageFooterContent (Object)** - content of footers for odd pages (`com.sun.star.sheet.HeaderFooterContent` service)

If you do not need to distinguish between headers or footers for odd and even pages (the `FooterIsShared` property is `False`), then set the properties for headers and footers on odd pages.

All the named objects return an object that supports the `com.sun.star.sheet.HeaderFooterContent` service. By means of the (non-genuine) properties `LeftText`, `CenterText`, and `RightText`, this service provides three text elements for the headers and footers of StarOffice Calc.

The following example writes the "Just a Test." value in the left-hand text field of the header from the "Default" template.

```
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
```

Note the last line in the example: Once the text is changed, the `TextContent` object must be assigned to the header again so that the change is effective.

Another mechanism for changing the text of headers and footers is available for text documents (StarOffice Writer) because these consist of a single block of text. The following properties are defined in the `com.sun.star.style.PageProperties` service:

- **HeaderText (Object)** – text object with content of the header
(`com.sun.star.text.XText` service)
- **HeaderTextLeft (Object)** – text object with content of headers on left-hand pages
(`com.sun.star.text.XText` service)
- **HeaderTextRight (Object)** – text object with content of headers on right-hand pages
(`com.sun.star.text.XText` service)
- **FooterText (Object)** – text object with content of the footer
(`com.sun.star.text.XText` service)
- **FooterTextLeft (Object)** – text object with content of footers on left-hand pages
(`com.sun.star.text.XText` service)
- **FooterTextRight (Object)** – text object with content of footers on right-hand pages
(`com.sun.star.text.XText` service)

The following example creates a header in the "Default" page style for text documents and adds the text "Just a Test" to the header.

```
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."

```

In this instance, access is provided directly through the `HeaderText` property of the page style rather than the `HeaderFooterContent` object.

Centering (Spreadsheets Only)

The `com.sun.star.sheet.TablePageStyle` service is only used in StarOffice Calc page styles and allows cell ranges that you want to printed to be centered on the page. This service provides the following properties:

- **CenterHorizontally (Boolean)** - table content is centered horizontally
- **CenterVertically (Boolean)** - table content is centered vertically

Definition of Elements to be Printed (Spreadsheets Only)

When you format sheets, you can define whether page elements are visible. For this purpose, the `com.sun.star.sheet.TablePageStyle` service provides the following properties:

- **PrintAnnotations (Boolean)** - prints cell comments
- **PrintGrid (Boolean)** - prints the cell gridlines
- **PrintHeaders (Boolean)** - prints the row and column headings
- **PrintCharts (Boolean)** - prints charts contained in a sheet
- **PrintObjects (Boolean)** - prints embedded objects
- **PrintDrawing (Boolean)** - prints draw objects
- **PrintDownFirst (Boolean)** - if the contents of a sheet extend across several pages, they are first printed in vertically descending order, and then down the right-hand side.
- **PrintFormulas (Boolean)** - prints the formulas instead of the calculated values
- **PrintZeroValues (Boolean)** - prints the zero values

Editing Spreadsheet Documents Efficiently

Whereas the previous section described the main structure of spreadsheet documents, this section describes the services that allow you to easily access individual cells or cell ranges.

Cell Ranges

In addition to an object for individual cells (`com.sun.star.table.Cell` service), StarOffice also provides objects that represent cell ranges. Such `CellRange` objects are created using the `getCellRangeByName` call of the spreadsheet object:

```
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")
```

A colon (:) is used to specify a cell range in a spreadsheet document. For example, A1:C15 represents all the cells in rows 1 to 15 in columns A, B, and C.

The location of individual cells in a cell range can be determined using the `getCellByPosition` method, where the coordinates of the top left cell in the cell range is (0, 0). The following example uses this method to create an object of cell 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)
```

Formatting Cell Ranges

Just like individual cells, you can apply formatting to cell ranges using the `com.sun.star.table.CellProperties` service. For more information and examples of this service, see the *Formatting* section.

Computing With Cell Ranges

You can use the `computeFunction` method to perform mathematical operations on cell ranges. The `computeFunction` expects a constant as the parameter that describes the mathematical function that you want to use. The associated constants are defined in the `com.sun.star.sheet.GeneralFunction` enumeration. The following values are available:

- **SUM** - sum of all numerical values
- **COUNT** - total number of all values (including non-numerical values)
- **COUNTNUMS** - total number of all numerical values
- **AVERAGE** - average of all numerical values
- **MAX** - largest numerical value
- **MIN** - smallest numerical value
- **PRODUCT** - product of all numerical values
- **STDEV** - standard deviation
- **VAR** - variance
- **STDEVP** - standard deviation based on the total population
- **VARP** - variance based on the total population

The following example computes the average value of the A1 : C3 range and prints the result in a message box:

```
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)
```

Deleting Cell Contents

The `clearContents` method simplifies the process of deleting cell contents and cell ranges in that it deletes one specific type of content from a cell range.

The following example removes all the strings and the direct formatting information from the B2 : C3 range.

```
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)
```

The flags specified in `clearContents` come from the `com.sun.star.sheet.CellFlags` constants list. This list provides the following elements:

- **VALUE** - numerical values that are not formatted as date or time
- **DATETIME** - numerical values that are formatted as date or time
- **STRING** - strings
- **ANNOTATION** - comments that are linked to cells
- **FORMULA** - formulas
- **HARDATTR** - direct formatting of cells
- **STYLES** - indirect formatting
- **OBJECTS** - drawing objects that are connected to cells
- **EDITATTR** - character formatting that only applies to parts of the cells

You can also add the constants together to delete different information using a call from `clearContents`.

Searching and Replacing Cell Contents

Spreadsheet documents, like text documents, provide a function for searching and replacing.

The descriptor objects for searching and replacing in spreadsheet documents are not created directly through the document object, but rather through the `Sheets` list. The following is an example of a search and replace process:

```
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
```

This example uses the first page of the document to create a `ReplaceDescriptor` and then applies this to all pages in a loop.

Drawings and Presentations

This chapter provides an introduction to the macro-controlled creation and editing of drawings. The first section describes the structure of drawings, including the basic elements that contain drawings. The second section addresses more complex editing functions, such as grouping, rotating, and scaling objects.

Information about creating, opening, and saving drawings can be found in Chapter 5, *Working with StarOffice Documents*.

The Structure of Drawings

StarOffice does not limit the number of pages in a drawing document. You can design each page separately. There is also no limit to the number of drawing elements that you can add to a page.

This picture is slightly complicated by the presence of *layers*. By default, each drawing document contains the *Layout*, *Controls*, and *Dimension Lines* layers and all drawing elements are added to the *Layout* layer. You also have the option to add new layers. See the StarOffice Developer's Guide for more information about drawing layers.

Pages

The pages of a drawing document are available through the `DrawPages` list. You can access individual pages either through their number or their name. If a document has one page and this is called *Slide 1*, then the following examples are identical.

Example 1:

```
Dim Doc As Object  
Dim Page As Object
```

```
Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages(0)
```

Example 2:

```
Dim Doc As Object
Dim Page As Object
```

```
Doc = StarDesktop.CurrentComponent
Page = Doc.drawPages.getByName("Slide 1")
```

In example 1, the page is addressed by its number (counting begins at 0). In the second example, the page is accessed by its name and the `getByName` method.

```
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
```

The preceding call returns a page object that supports the `com.sun.star.drawing.DrawPage` service. The service recognizes the following properties:

- **BorderLeft (Long)** – left-hand border in hundredths of a millimeter
- **BorderRight (Long)** – right-hand border in hundredths of a millimeter
- **BorderTop (Long)** – top border in hundredths of a millimeter
- **BorderBottom (Long)** – bottom border in hundredths of a millimeter
- **Width (Long)** – page width in hundredths of a millimeter
- **Height (Long)** – page height in hundredths of a millimeter
- **Number (Short)** – number of pages (numbering begins at 1), read-only
- **Orientation (Enum)** – page orientation (in accordance with `com.sun.star.view.PaperOrientation` enumeration)

If these settings are changed, then *all* of the pages in the document are affected.

The following example sets the page size of a drawing document which has just been opened to 20 x 20 centimeters with a page margin of 0.5 centimeters:

```
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

```

Elementary Properties of Drawing Objects

Drawing objects include shapes (rectangles, circles, and so on), lines, and text objects. All of these share a number of common features and support the `com.sun.star.drawing.Shape` service. This service defines the `Size` and `Position` properties of a drawing object.

StarOffice Basic also offers several other services through which you can modify such properties, as formatting or apply fills. The formatting options that are available depend on the type of drawing object.

The following example creates and inserts a rectangle in a drawing document:

```

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)

```

This example uses the `StarDesktop.CurrentComponent` call to determine which document is open. The document object determined this way returns the first page of the drawing through the `drawPages(0)` call.

The `Point` and `Size` structures with the point of origin (left hand corner) and the size of the drawing object are then initialized. The lengths are specified in hundredths of a millimeter.

The program code then uses the `Doc.CreateInstance` call to create the rectangle drawing object as specified by the `com.sun.star.drawing.RectangleShape` service. At the end, the drawing object is assigned to a page using a `Page.add` call.

Fill Properties

This section describes four services and in each instance the sample program code uses a rectangle shape element that combines several types of formatting. Fill properties are combined in the `com.sun.star.drawing.FillProperties` service.

StarOffice recognizes four main types of formatting for a fill area. The simplest variant is a single-color fill. The options for defining color gradients and hatches let you create other colors into play. The fourth variant is the option of projecting existing graphics into the fill area.

The fill mode of a drawing object is defined using the `FillStyle` property. The permissible values are defined in `com.sun.star.drawing.FillStyle`.

Single Color Fills

The main property for single-color fills is

- **FillColor (Long)** – fill color of area.

To use the fill mode, you must the `FillStyle` property to the `SOLID` fill mode.

The following example creates a rectangle shape and fills it with red (RGB value 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)
```

Color Gradient

If you set the `FillStyle` property to `GRADIENT`, you can apply a color gradient to any fill area of a StarOffice document.

If you want to apply a predefined color gradient, you can assign the associated name of the `FillTransparenceGradientName` property. To define your own color gradient, you need to complete a `com.sun.star.awt.Gradient` structure to assign the `FillGradient` property. This property provides the following options:

- **Style (Enum)** - type of gradient, for example, linear or radial (default values in accordance with `com.sun.star.awt.GradientStyle`)
- **StartColor (Long)** - start color of color gradient
- **EndColor (Long)** - end color of color gradient
- **Angle (Short)** - angle of color gradient in tenths of a degree
- **XOffset (Short)** - X-coordinate at which the color gradient starts, specified in hundredths of a millimeter
- **YOffset (Short)** - Y-coordinate at which the color gradient begins, specified in hundredths of a millimeter
- **StartIntensity (Short)** - intensity of `StartColor` as a percentage (in StarOffice Basic, you can also specify values higher than 100 percent)
- **EndIntensity (Short)** - intensity of `EndColor` as a percentage (in StarOffice Basic, you can also specify values higher than 100 percent)
- **StepCount (Short)** - number of color graduations which StarOffice is to calculate for the gradients

The following example demonstrates the use of color gradients with the aid of the `com.sun.star.awt.Gradient` structure:

```
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.GRAIENT
RectangleShape.FillGradient = Gradient

Page.add(RectangleShape)

```

This example creates a linear color gradient (`Style = LINEAR`). The gradient starts with red (`StartColor`) in the top left corner, and extends at a 45 degree angle (`Angle`) to green (`EndColor`) in the bottom right corner. The color intensity of the start and end colors is 150 percent (`StartIntensity` and `EndIntensity`) which results in the colors seeming brighter than the values specified in the `StartColor` and `EndColor` properties. The color gradient is depicted using a hundred graduated individual colors (`StepCount`).

Hatches

To create a hatch fill, the `FillStyle` property must be set to `HATCH`. The program code for defining the hatch is very similar to the code for color gradients. Again an auxiliary structure, in this case `com.sun.star.drawing.Hatch`, is used to define the appearance of hatches. The structure for hatches has the following properties:

- **Style (Enum)** - type of hatch: simple, squared, or squared with diagonals (default values in accordance with `com.sun.star.awt.HatchStyle`)
- **Color (Long)** - color of lines
- **Distance (Long)** - distance between lines in hundredths of a millimeter
- **Angle (Short)** - angle of hatch in tenths of a degree

The following example demonstrates the use of a hatch structure:

```

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)

```

This code creates a simple hatch structure (`HatchStyle = SINGLE`) whose lines are rotated 45 degrees (`Angle`). The lines are dark gray (`Color`) and are spaced is 0.2 millimeters (`Distance`) apart.

Bitmaps

To use bitmap projection as a fill, you must set the `FillStyle` property to `BITMAP`. If the bitmap is already available in StarOffice, you just need to specify its name in the `FillBitmapName` property and its display style (simple, tiled, or elongated) in the `FillBitmapMode` property (default values in accordance with `com.sun.star.drawing.BitmapMode`).

If you want to use an external bitmap file, you can specify its URL in the `FillBitmapURL` property.

The following example creates a rectangle and tiles the Sky bitmap that is available in StarOffice to fill the area of the rectangle:

```

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)

```

Transparency

You can adjust the transparency of any fill that you apply. The simplest way to change the transparency of a drawing element is to use the `FillTransparency` property.

The following example creates a red rectangle with a transparency of 50 percent.

```

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.FillTransparency = 50
RectangleShape.FillColor = RGB(255,0,0)

Page.add(RectangleShape)

```

To make the fill transparent, set the `FillTransparency` property to 100.

In addition to the `FillTransparency` property, the `com.sun.star.drawing.FillProperties` service also provides the `FillTransparencyGradient` property. This is used to define a gradient that specifies the transparency of a fill area.

Line Properties

All drawing objects that can have a border line support the `com.sun.star.drawing.LineStyle` service. Some of the properties that this service provides are:

- **LineStyle (Enum)** - line type (default values in accordance with `com.sun.star.drawing.LineStyle`)
- **LineColor (Long)** - line color
- **LineTransparence (Short)** - line transparency
- **LineWidth (Long)** - line thickness in hundredths of a millimeter
- **LineJoint (Enum)** - transitions to connection points (default values in accordance with `com.sun.star.drawing.LineJoint`)

The following example creates a rectangle with a solid border (`LineStyle = SOLID`) that is 5 millimeters thick (`LineWidth`) and 50 percent transparent. The right and left-hand edges of the line extend to their points of intersect with each other (`LineJoint = MITER`) to form a right-angle.

```
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)
```

In addition to the listed properties, the `com.sun.star.drawing.LineStyle` service provides options for drawing dotted and dashed lines. For more information, see the StarOffice API reference.

Text Properties (Drawing Objects)

The `com.sun.star.style.CharacterProperties` and `com.sun.star.style.ParagraphProperties` services can format text in drawing objects. These services relate to individual characters and paragraphs and are described in detail in Chapter 6 (*Text Documents*).

The following example inserts text in a rectangle and formats the font `com.sun.star.style.CharacterProperties` service.

```
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 = "This is a test"
RectangleShape.CharWeight = com.sun.star.awt.FontWeight.BOLD
RectangleShape.CharFontName = "Arial"
```

This code uses the `String`-property of the rectangle to insert the text and the `CharWeight` and `CharFontName` properties from the `com.sun.star.style.CharacterProperties` service to format the text font.

The text can only be inserted after the drawing object has been added to the drawing page. You can also use the `com.sun.star.drawing.Text` service to position and format text in drawing object. The following are some of the important properties of this service:

- **TextAutoGrowHeight (Boolean)** – adapts the height of the drawing element to the text it contains
- **TextAutoGrowWidth (Boolean)** - adapts the width of the drawing element to the text it contains
- **TextHorizontalAdjust (Enum)** - horizontal position of text within the drawing element (default values in accordance with `com.sun.star.drawing.TextHorizontalAdjust`)
- **TextVerticalAdjust (Enum)** - vertical position of text within the drawing element (default values in accordance with `com.sun.star.drawing.TextVerticalAdjust`)
- **TextLeftDistance (Long)** - left-hand distance between drawing element and text in hundredths of a millimeter
- **TextRightDistance (Long)** - right-hand distance between drawing element and text in hundredths of a millimeter
- **TextUpperDistance (Long)** - upper distance between drawing element and text in hundredths of a millimeter

- **TextLowerDistance (Long)** - lower distance between drawing element and text in hundredths of a millimeter

The following example demonstrates use of the named properties.

```
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 = "This is a test" ' May only take place after Page.add!

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
```

This code inserts a drawing element in a page and then adds text to the top left corner of the drawing object using the `TextVerticalAdjust` and `TextHorizontalAdjust` properties. The minimum distance between the text edge of the drawing object is set to three millimeters.

Shadow Properties

You can add a shadow to most drawing objects with the `com.sun.star.drawing.ShadowProperties` service. The properties of this service are:

- **Shadow (Boolean)** – activates the shadow
- **ShadowColor (Long)** – shadow color
- **ShadowTransparence (Short)** – transparency of the shadow
- **ShadowXDistance (Long)** – vertical distance of the shadow from the drawing object in hundredths of a millimeter

- **ShadowYDistance (Long)** – horizontal distance of the shadow from the drawing object in hundredths of a millimeter

The following example creates a rectangle with a shadow that is vertically and horizontally offset from the rectangle by 2 millimeters. The shadow is rendered in dark gray with 50 percent transparency.

```
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.ShadowTransparency = 50
RectangleShape.ShadowXDistance = 200
RectangleShape.ShadowYDistance = 200

Page.add(RectangleShape)
```

An Overview of Various Drawing Objects

Rectangle Shapes

Rectangle shape objects (`com.sun.star.drawing.RectangleShape`) support the following services for formatting objects:

- **Fill properties** – `com.sun.star.drawing.FillProperties`
- **Line properties** – `com.sun.star.drawing.LineProperties`
- **Text properties** – `com.sun.star.drawing.Text` (with `com.sun.star.style.CharacterProperties` and `com.sun.star.style.ParagraphProperties`)
- **Shadow properties** – `com.sun.star.drawing.ShadowProperties`
- **CornerRadius (Long)** – radius for rounding corners in hundredths of a millimeter

Circles and Ellipses

The Service `com.sun.star.drawing.EllipseShape` service is responsible for circles and ellipses and supports the following services:

- **Fill properties** – `com.sun.star.drawing.FillProperties`
- **Line properties** – `com.sun.star.drawing.LineProperties`
- **Text properties** – `com.sun.star.drawing.Text` (with `com.sun.star.style.CharacterProperties` and `com.sun.star.style.ParagraphProperties`)
- **Shadow properties** – `com.sun.star.drawing.ShadowProperties`

In addition to these services, circles and ellipses also provide these properties:

- **CircleKind (Enum)** – type of circle or ellipse (default values in accordance with `com.sun.star.drawing.CircleKind`)
- **CircleStartAngle (Long)** – start angle in tenths of a degree (only for circle or ellipse segments)
- **CircleEndAngle (Long)** – end angle in tenths of a degree (only for circle or ellipse segments)

The `CircleKind` property determines if an object is a complete circle, a circular slice, or a section of a circle. The following values are available:

- **`com.sun.star.drawing.CircleKind.FULL`** – full circle or full ellipse
- **`com.sun.star.drawing.CircleKind.CUT`** – section of circle (partial circle whose interfaces are linked directly to one another)
- **`com.sun.star.drawing.CircleKind.SECTION`** – circle slice
- **`com.sun.star.drawing.CircleKind.ARC`** – angle (not including circle line)

The following example creates a circular slice with a 70 degree angle (produced from difference between start angle of 20 degrees and end angle of 90 degrees)

```
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)

```

Lines

StarOffice provides the `com.sun.star.drawing.LineShape` service for line objects. Line objects support all of the general formatting services with the exception of areas. The following are all of the properties that are associated with the `LineShape` service:

- **Line properties** – `com.sun.star.drawing.LineProperties`
- **Text properties** – `com.sun.star.drawing.Text` (with `com.sun.star.style.CharacterProperties` and `com.sun.star.style.ParagraphProperties`)
- **Shadow properties** – `com.sun.star.drawing.ShadowProperties`

The following example creates and formats a line with the help of the named properties. The origin of the line is specified in the `Location` property, whereas the coordinates listed in the `Size` property specify the end point of the line.

```

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)

```

Polypolygon Shapes

StarOffice also supports complex polygonal shapes through the `com.sun.star.drawing.PolyPolygonShape` service. Strictly speaking, a *PolyPolygon* is not a simple polygon but a multiple polygon. Several independent lists containing corner points can therefore be specified and combined to form a complete object.

As with rectangle shapes, all the formatting properties of drawing objects are also provided for polypolygons:

- **Fill properties** – `com.sun.star.drawing.FillProperties`
- **Line properties** – `com.sun.star.drawing.LineProperties`
- **Text properties** – `com.sun.star.drawing.Text` (with `com.sun.star.style.CharacterProperties` and `com.sun.star.style.ParagraphProperties`)
- **Shadow properties** – `com.sun.star.drawing.ShadowProperties`

The `PolyPolygonShape` service also has a property that lets you define the coordinates of a polygon:

- `PolyPolygon (Array)` – field containing the coordinates of the polygon (double array with points of the `com.sun.star.awt.Point` type)

The following example shows how you can define a triangle with the `PolyPolygonShape` service.

```
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 must take place before the coordinates are set

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())
```

Since the points of a polygon are defined as absolute values, you do not need to specify the size or the start position of a polygon. Instead, you need to create an array of the points, package this array in a second array (using the `Array(Coordinates())` call), and then assign this array to the polygon. Before the corresponding call can be made, the polygon must be inserted into the document.

The double array in the definition allows you to create complex shapes by merging several polygons. For example, you can create a rectangle and then insert another rectangle inside it to create a hole in the original rectangle:

```
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 must take place before the coordinates are set

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())

```

With respect as to which areas are filled and which areas are holes, StarOffice applies a simple rule: the edge of the outer shape is always the outer border of the polypolygon. The next line inwards is the inner border of the shape and marks the transition to the first hole. If there is another line inwards, it marks the transition to a filled area.

Graphics

The last of the drawing elements presented here are graphic objects that are based on the `com.sun.star.drawing.GraphicObjectShape` service. These can be used with any graphic within StarOffice whose appearance can be adapted using a whole range of properties.

Graphic objects support two of the general formatting properties:

- **Text properties** – `com.sun.star.drawing.Text` (with `com.sun.star.style.CharacterProperties` and `com.sun.star.style.ParagraphProperties`)
- **Shadow properties** – `com.sun.star.drawing.ShadowProperties`

Additional properties that are supported by graphic objects are:

- **GraphicURL (String)** – URL of the graphic
- **AdjustLuminance (Short)** – luminance of the colors, as a percentage (negative values are also permitted)
- **AdjustContrast (Short)** – contrast as a percentage (negative values are also permitted)
- **AdjustRed (Short)** – red value as a percentage (negative values are also permitted)
- **AdjustGreen (Short)** – green value as a percentage (negative values are also permitted)
- **AdjustBlue (Short)** – blue value as a percentage (negative values are also permitted)
- **Gamma (Short)** – gamma value of a graphic
- **Transparency (Short)** – transparency of a graphic as a percentage
- **GraphicColorMode (enum)** – color mode, for example, standard, gray stages, black and white (default value in accordance with `com.sun.star.drawing.ColorMode`)

The following example shows how to insert a page into a graphics object. 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          ' specifications, insignificant because latter
                        ' coordinates are binding
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)

```

This code inserts the `test.jpg` graphic and adapts its appearance using the `Adjust` properties. In this example, the graphics are depicted as 40 percent transparent with no other color conversions do not take place (`GraphicColorMode = STANDARD`).

Editing Drawing Objects

Grouping Objects

In many situations, it is useful to group several individual drawing objects together so that they behave as a single large object.

The following example combines two drawing objects:

```

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
' create square drawing element
Square = Doc.createInstance("com.sun.star.drawing.RectangleShape")
Square.Size = Size
Square.Position = Point
Square.FillColor = RGB(255,128,128)
Page.add(Square)

```

```

' create circle drawing element
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)
' combine square and circle drawing elements
Shapes = createUnoService("com.sun.star.drawing.ShapeCollection")
Shapes.add(Square)
Shapes.add(Circle)
Group = Page.group(Shapes)
' centre combined drawing elements
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

```

This code creates a rectangle and a circle and inserts them into a page. It then creates an object that supports the `com.sun.star.drawing.ShapeCollection` service and uses the `Add` method to add the rectangle and the circle to this object. The `ShapeCollection` is added to the page using the `Group` method and returns the actual `Group` object that can be edited like an individual `Shape`.

If you want to format the individual objects of a group, apply the formatting before you add them to the group. You cannot modify the objects once they are in the group.

Rotating and Shearing Drawing Objects

All of the drawing objects that are described in the previous sections can also be rotated and sheared using the `com.sun.star.drawing.RotationDescriptor` service.

The service provides the following properties:

- **RotateAngle (Long)** – rotary angle in hundredths of a degree
- **ShearAngle (Long)** – shear angle in hundredths of a degree

The following example creates a rectangle and rotates it by 30 degrees using the `RotateAngle` property:

```

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)

```

The next example creates the same rectangle as in the previous example, but instead shears it through 30 degrees using the `ShearAngle` property.

```

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)

```

Searching and Replacing

As in text documents, drawing documents provide a function for searching and replace. This function is similar to the one that is used in text documents as described in Chapter 6, *Text Documents*. However, in drawing documents the descriptor objects for searching and replacing are not created directly through the document object, but rather through the associated character level. The following example outlines the replacement process within a drawing:

```

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

```

This code uses the first `DrawPage` of the document to create a `ReplaceDescriptor` and then applies this descriptor in a loop to all of the pages in the drawing document.

Presentations

StarOffice presentations are based on drawing documents. Each page in the presentation is a slide. You can access slides in the same way as a standard drawing is accessed through the `DrawPages` list of the document object. The `com.sun.star.presentation.PresentationDocument` service, responsible for presentation documents, also provides the complete `com.sun.star.drawing.DrawingDocument` service.

Working With Presentations

In addition to the drawing functions that are provided by the `Presentation` property, the presentation document has a presentation object that provides access to the main properties and control mechanisms for presentations. For example, this object provides a `start` method that can start presentations.

```

Dim Doc As Object
Dim Presentation As Object

Doc = StarDesktop.CurrentComponent
Presentation = Doc.Presentation
Presentation.start()

```

The code used in this example creates a `Doc` object that references the current presentation document and establishes the associated presentation object. The `start()` method of the object is used to start the example and run the screen presentation.

The following methods are provided as presentation objects:

- **start** - starts the presentation
- **end** - ends the presentation
- **rehearseTimings** - starts the presentation from the beginning and establishes its runtime

The following properties are also available:

- **AllowAnimations (Boolean)**– runs animations in the presentation
- **CustomShow (String)** – allows you to specify the name of the presentation so that you can reference the name in the presentation
- **FirstPage (String)** – name of slide that you want to start the presentation with
- **IsAlwaysOnTop (Boolean)** – always displays the presentation window as the first window on the screen
- **IsAutomatic (Boolean)** – automatically runs through the presentation
- **IsEndless (Boolean)** – restarts the presentation from the beginning once it ends
- **IsFullScreen (Boolean)** – automatically starts the presentation in full screen mode
- **IsMouseVisible (Boolean)** – displays the mouse during the presentation
- **Pause (long)** – the amount of time that a blank screen is displayed at the end of the presentation
- **StartWithNavigator (Boolean)** – displays the navigator window when the presentation starts
- **UsePn (Boolean)** – displays the pointer during the presentation

Diagrams (Charts)

StarOffice can display data as a diagram, which creates graphical links between data in the form of bars, pie charts, lines or other elements. Data can either be displayed as 2D or 3D graphics, and the appearance of the diagram elements can be individually adapted in a similar way to the process used for drawing elements.

If the data is available in the form of a spreadsheet, then this can be dynamically linked to the diagram. Any modifications made to the basic data can in this instance be seen immediately in the assigned diagram. This chapter provides an overview of the programming interface for diagram modules of StarOffice and focuses on the use of diagrams within spreadsheet documents.

Using Diagrams in Spreadsheets

Diagrams are not treated as independent documents in StarOffice, but as objects that are embedded in an existing document.

While diagrams in text and drawing documents remain isolated from the content of the document, when used in spreadsheet documents, a mechanism is provided which allows a link to be established between the document data and embedded diagrams. The following example explains the interaction between spreadsheet document and diagram:

```
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)

```

Although the code used in the example may appear to be complex, the central processes are limited to three lines: the first central line creates the `Doc` document variable, which references the current spreadsheet document (`Doc line = StarDesktop.CurrentComponent`). The code used in the example then creates a list containing all charts of the first spreadsheet (`Charts line = Doc.Sheets(0).Charts`). Finally, a new chart is added to the last line of this list using the `addNewByName` method. This new chart is then visible to the user.

The last line initializes the `Rect` and `RangeAddress` auxiliary structures, which the `addNewByName` method also provides as a parameter. `Rect` determines the position of the chart within the spreadsheet. `RangeAddress` determines the range whose data is to be linked to the chart.

The previous example creates a bar diagram. If a different type of graphic is needed, then the bar diagram must be explicitly replaced:

```

Chart = Charts.getByName("MyChart").embeddedObject
Chart.Diagram = Chart.createInstance("com.sun.star.chart.LineDiagram")

```

The first lines defines the corresponding chart object. The second line replaces the current diagram with a new one — in this example, a line diagram.

Note – In Excel, a distinction is made between charts which have been inserted as a separate page in an Excel document and charts which are embedded in a table page. Correspondingly, two different access methods are defined there for charts. This distinction is not made in StarOffice Basic, because charts in StarOffice Calc are always created as embedded objects of a table page. The charts are always accessed using the `Charts` list of the associated `Sheet` object.

The Structure of Diagrams

The structure of a diagram — and therefore the list of services and interfaces supported by it — depends on its type. The methods and properties of the Z-axis, are, for example, only available in 3D diagrams, but not in 2D diagrams. In pie charts, there are no interfaces for working with axes.

The Individual Elements of a Diagram

Title, Sub-title and Key

A title, sub-title and key form part of the basic elements of every diagram. Diagrams provide their own objects for each of these elements. The `Chart` object provides the following properties for administrating these elements:

- **HasMainTitle (Boolean)** – activates the title.
- **Title (Object)** – object with detailed information about the diagram title (supports the `com.sun.star.chart.ChartTitle` service).
- **HasSubTitle(Boolean)** – activates the sub-title.
- **Subtitle (Object)** – object with detailed information about the diagram sub-title (supports the `com.sun.star.chart.ChartTitle` service).
- **HasLegend (Boolean)** – activates the key.
- **Legend (Object)** – object with detailed information about the key to the diagram (supports the `com.sun.star.chart.ChartLegendPosition` service).

In many respects, the elements specified correspond to a drawing element. This is due to the fact that both the `com.sun.star.chart.ChartTitle` service and the `com.sun.star.chart.ChartLegendPosition` support the `com.sun.star.drawing.Shape` service, which forms the technical program basis for drawing elements.

Users therefore have the opportunity to determine the position and size of the element using the `Size` and `Position` properties.

The other fill and line properties (`com.sun.star.drawing.FillProperties` and `com.sun.star.drawing.LineStyle` services) as well as the character properties (`com.sun.star.style.CharacterProperties` service) are provided for formatting the elements.

`com.sun.star.chart.ChartTitle` contains not only the named format properties, but also two other properties:

- **TextRotation (Long)** – angle of rotation of text in 100ths of a degree.
- **String (String)** – text which to be displayed as the title or sub-title.

The diagram key (`com.sun.star.chart.ChartLegend` service) contains the following additional property:

- **Alignment (Enum)** – position at which the key appears (default value in accordance with `com.sun.star.chart.ChartLegendPosition`).

The following example creates a diagram and assigns it the title "Test", the sub-title "Test 2" and a key. The key has a gray background color, is placed at the bottom of the diagram, and has a character size of 7 points.

```
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
```

Background

Every diagram has a background area. Every area has an object, which can be accessed using the following properties of the diagram object:

- **Area (Object)** – background area of the diagram (supports `com.sun.star.chart.ChartArea` service).

The background of a diagram covers its complete area, including the area under the title, sub-title and diagram key. The associated `com.sun.star.chart.ChartArea` service supports line and fill properties and provides no more extensive properties.

Diagram Walls and Floors

Although the diagram background covers the entire area of the diagram, the diagram back wall is limited to the area directly behind the data area.

Two diagram walls usually exist for 3D diagrams: one behind the data area and one as the left-hand demarcation to the Y-axis. 3D diagrams usually also have a floor.

- **Floor (Object)** - floor panel of the diagram (only for 3D diagrams, supports `com.sun.star.chart.ChartArea` service).
- **Wall (Object)** - diagram walls (only for 3D diagrams, supports `com.sun.star.chart.ChartArea` service).

The specified objects support the `com.sun.star.chart.ChartArea` service, which in turn provides the usual fill and line properties (`com.sun.star.drawing.FillProperties` and `com.sun.star.drawing.LineStyle` services, refer to [Chapter 8](#)).

The diagram walls and floor are accessed through the `Chart` object, which in turn is part of the `Chart` object:

```
Chart.Area.FillBitmapName = "Sky"
```

The following example shows how graphics (named `Sky`) already contained in `StarOffice` can be used as a background to a diagram.

```
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

```

Axes

StarOffice recognizes five different axes that can be used in a diagram. In the simplest scenario, these are the X and Y-axes. When working with 3D diagrams, a Z-axis is also sometimes provided. For diagrams in which the values of the various rows of data deviate significantly from one another, StarOffice provides a second X and Y-axis for second scaling operations.

First X, Y and Z-Axis

In addition to the actual axis, for each of the first X, Y, and Z-axes there can also be a title, a description, a grid, and an auxiliary grid. There is an option for displaying and concealing all of these elements. The diagram object provides the following properties for administration of these features (taking the example of a X-axis; properties for Y and Z-axis are structured in the same way):

- **HasXAxis (Boolean)** – activates the X-axis.
- **XAxis (Object)** – object with detailed information about the X-axis (supports `com.sun.star.chart.ChartAxis` service).
- **HasXAxisDescription (Boolean)** – activates description for the X-axis.
- **HasXAxisGrid (Boolean)** – activates main grid for X-axis.
- **XMainGrid (Object)** – object with detailed information about main grid for X-axis (supports `com.sun.star.chart.ChartGrid` service).
- **HasXAxisHelpGrid (Boolean)** – activates auxiliary grid for X-axis.
- **XHelpGrid (Object)** – object with detailed information about auxiliary grid for X-axis (supports `com.sun.star.chart.ChartGrid` service).
- **HasXAxisTitle (Boolean)** – activates title of X-axis.
- **XAxisTitle (Object)** – object with detailed information about title of X-axis (supports `com.sun.star.chart.ChartTitle` service).

Second X and Y-Axis

The following properties are available for the second X and Y-axes (properties taking example of the second X-axis):

- **HasSecondaryXAxis (Boolean)** – activates the second X-axis.

- **SecondaryXAxis (Object)** – object with detailed information about the second X-axis (supports `com.sun.star.chart.ChartAxis` service).
- **HasSecondaryXAxisDescription (Boolean)** – activates description of X-axis.

Properties of the Axes

The axis objects of a StarOffice diagram support the `com.sun.star.chart.ChartAxis` service. In addition to the properties for characters (`com.sun.star.style.CharacterProperties` service, refer to [Chapter 6](#)) and lines (`com.sun.star.drawing.LineStyle` service, refer to [Chapter 8](#)), it provides the following properties:

- **Max (Double)** - maximum value for axis.
- **Min (Double)** - minimum value for axis.
- **Origin (Double)** - point of intersect for crossing axes.
- **StepMain (Double)** - distance between two primary lines of the axis.
- **StepHelp (Double)** - distance between two secondary lines of the axis.
- **AutoMax (Boolean)** - automatically determines maximum value for axis.
- **AutoMin (Boolean)** - automatically determines minimum value for axis.
- **AutoOrigin (Boolean)** - automatically determines point of intersect for crossing axes.
- **AutoStepMain (Boolean)** - automatically determines distance between primary lines of an axis.
- **AutoStepHelp (Boolean)** - automatically determines distance between secondary lines of an axis.
- **Logarithmic (Boolean)** - scales the axes in logarithmic manner (rather than linear manner).
- **DisplayLabels (Boolean)** - activates the text label for axes.
- **TextRotation (Long)** - angle of rotation of text label of axes in 100ths of a degree.
- **Marks (Const)** - constant that specifies whether the primary lines of the axis should be inside or outside the diagram area (default values in accordance with `com.sun.star.chart.ChartAxisMarks`)
- **HelpMarks (Const)** - constant that specifies whether the secondary lines of the axis should be inside and/or outside the diagram area (default values in accordance with `com.sun.star.chart.ChartAxisMarks`)
- **Overlap (Long)** - percentage which specifies the extent to which the bars of different sets of data may overlap (at 100%, the bars are shown as completely overlapping, at -100%, there is a distance of the width of one bar between them).
- **GapWidth (long)** - percentage which specifies the distance there may be between the different groups of bars of a chart (at 100%, there is a distance corresponding to the width of one bar).

- **ArrangeOrder (enum)** - details of position of inscription; in addition to positioning on a line, there is also the option of splitting the label alternately over two lines (default value according to `com.sun.star.chart.ChartAxisArrangeOrderType`)
- **TextBreak (Boolean)** - permits line breaks.
- **TextCanOverlap (Boolean)** - permits text overlaps.
- **NumberFormat (Long)** - number format (refer to “[Number, Date and Text Format](#)” on page 132 section)

Properties of the axis grid

The object for the axis grid is based on the `com.sun.star.chart.ChartGrid` service, which in turn supports the line properties of the `com.sun.star.drawing.LineStyle` support service (refer to [Chapter 8](#)).

Properties of the axis title

The objects for formatting the axis title are based on the `com.sun.star.chart.ChartTitle` service, which is also used for diagram titles.

Example

The following example creates a line diagram. The color for the rear wall of the diagram is set to white. Both the X and Y-axes have a gray auxiliary grid for visual orientation. The minimum value of the Y-axis is fixed to 0 and the maximum value is fixed to 100 so that the resolution of the diagram is retained even if the values are changed.

```
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 Diagrams

Most diagrams in StarOffice can also be displayed with 3D graphics. All diagram types that provide this option support the `com.sun.star.chart.Dim3DDiagram` service. The service provides just one property:

- **Dim3D (Boolean)** – activates 3D display.

Stacked Diagrams

Stacked diagrams are diagrams that are arranged with several individual values on top of one another to produce a total value. This view shows not only the individual values, but also an overview of all the values.

In StarOffice, various types of diagrams can be displayed in a stacked form. All of these diagrams support the `com.sun.star.chart.StackableDiagram` service, which in turn provides the following properties:

- **Stacked (Boolean)** – activates the stacked viewing mode.
- **Percent (Boolean)** – rather than absolute values, displays their percentage distribution.

Diagram Types

Line Diagrams

Line diagrams (Service `com.sun.star.chart.LineDiagram`) support one X-axis, two Y-axes and one Z-axis. They can be displayed as 2D or 3D graphics (`com.sun.star.chart.Dim3Ddiagram` service). The lines can be stacked (`com.sun.star.chart.StackableDiagram`).

Line diagrams provide the following properties:

- **SymbolType (const)** - symbol for displaying the data points (constant in accordance with `com.sun.star.chart.ChartSymbolType`).
- **SymbolSize (Long)** - size of symbol for displaying the data points in 100ths of a millimeter.
- **SymbolBitmapURL (String)** - file name of graphics for displaying the data points.
- **Lines (Boolean)** - links the data points by means of lines.
- **SplineType (Long)** - spline function for smoothing the lines (0: no spline function, 1: cubic splines, 2: B splines).
- **SplineOrder (Long)** - polynomial weight for splines (only for B splines).
- **SplineResolution (Long)** - number of support points for spline calculation.

Area Diagrams

Area diagrams (`com.sun.star.chart.AreaDiagram` service) support one X-axis, two Y-axes and one Z-axis. They can be displayed as 2D or 3D graphics (`com.sun.star.chart.Dim3Ddiagram` service). The areas can be stacked (`com.sun.star.chart.StackableDiagram`).

Bar Diagrams

Bar diagrams (Service `com.sun.star.chart.BarDiagram`) support one X-axis, two Y-axes and one Z-axis. They can be displayed as 2D or 3D graphics (`com.sun.star.chart.Dim3Ddiagram` service). The bars can be stacked (`com.sun.star.chart.StackableDiagram`).

They provide the following properties:

- **Vertical (Boolean)** — displays the bars vertically, otherwise they are depicted horizontally.

- **Deep (Boolean)** - in 3D viewing mode, positions the bars behind one another rather than next to one another.
- **StackedBarsConnected (Boolean)** - links the associated bars in a stacked diagram by means of lines (only available with horizontal charts).
- **NumberOfLines (Long)** - number of lines to be displayed in a stacked diagram as lines rather than bars.

Pie Diagrams

Pie diagrams (`com.sun.star.chart.PieDiagram` service) do not contain any axes and cannot be stacked. They can be displayed as 2D or 3D graphics (`com.sun.star.chart.Dim3Ddiagram` service).

Database Access

StarOffice has an integrated database interface (independent of any systems) called Star Database Connectivity (SDBC). The objective of developing this interface was to provide access to as many different data sources as possible.

To make this possible, data sources are accessed by drivers. The sources from which the drivers take their data is irrelevant to a SDBC user. Some drivers access file-based databases and take the data directly from them. Others use standard interfaces such as JDBC or ODBC. There are, however, also special drivers which access the MAPI address book, LDAP directories or StarOffice spreadsheets as data sources.

Since the drivers are based on UNO components, other drivers can be developed and therefore open up new data sources. You will find details about this in the StarOffice Developer's Guide.

Note – In terms of its concept, SDBC is comparable with the ADO and DAO libraries available in VBA. It permits high level access to databases, regardless of the underlying database backends.

Note – The database interface of StarOffice has grown through the launch of StarOffice 8. Although in the past, databases were primarily accessed using a range of methods of the `Application` object, the interface in StarOffice 7 sub-divides into several objects. A `DatabaseContext` is used as the root object for the database functions.

SQL: a Query Language

The SQL language is provided as a query language for users of ODBC. To compare the differences between different SQL dialects, the ODBC components from StarOffice have their own SQL parser. This uses the query window to check the SQL commands typed and corrects simple syntax errors, such as those associated with uppercase and lowercase characters.

If a driver permits access to a data source that does not support SQL, then it must independently convert the transferred SQL commands to the native access needed.

Note – SQL implementation from ODBC is oriented towards the SQL-ANSI-Standard. Microsoft-specific extensions, such as the `INNER JOIN` construct are not supported. These should be replaced with standard commands (`INNER JOIN`, for example should be replaced with a corresponding `WHERE` clause).

Types of Database Access

The database interface from StarOffice is available in the StarOffice Writer and StarOffice Calc applications, as well as in the database forms.

In StarOffice Writer, standard letters can be created with the assistance of ODBC data sources and these can then be printed out. There is also an option for moving data from the database window into text documents using the drag-and-drop function.

If the user moves a database table into a spreadsheet, StarOffice creates a table area which can be updated at the click of the mouse if the original data has been modified. Conversely, spreadsheet data can be moved to a database table and a database import performed.

Finally, StarOffice provides a mechanism for forms based on databases. To do this, the user first creates a standard StarOffice Writer or StarOffice Calc form and then links the fields to a database.

All the options specified here are based on the user interface from StarOffice. No programming knowledge is needed to use the corresponding functions.

This chapter, however, provides little information about the functions specified, but instead concentrates on the programming interface from ODBC, which allows for automated database querying and therefore permits a much greater range of applications to be used.

Basic knowledge of the way in which databases function and the SQL query language is however needed to fully understand the following sections.

Data Sources

A database is incorporated into StarOffice by creating what is commonly referred to as a *data source*. The user interface provides a corresponding option for creating data sources in the **Extras** menu. However, you also can create data sources and work with them using StarOffice Basic.

A database context object that is created using the `createUnoService` function serves as the starting point for accessing a data source. This is based on the `com.sun.star.sdb.DatabaseContext` service and is the root object for all database operations.

The following example shows how a database context can be created and then used to determine the names of all data sources available. It displays the names in a message box.

```
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
```

The individual data sources are based on the `com.sun.star.sdb.DataSource` service and can be determined from the database context using the `getByName` method:

```
Dim DatabaseContext As Object
Dim DataSource As Object

DatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
DataSource = DatabaseContext.getByName("Customers")
```

The example creates a `DataSource` object for a data source called *Customers*.

Data sources provide a range of properties, which in turn provide general information about the origin of the data and information about access methods. The properties are:

- **Name (String)** – name of data source.
- **URL (String)** – URL of data source in the form of *jdbc: subprotocol : subname* or *sdbc: subprotocol : subname*.

- **Info (Array)** – array containing `PropertyValue`-pairs with connection parameters (usually at least user name and password).
- **User (String)** – user name.
- **Password (String)** – user password (is not saved).
- **IsPasswordRequired (Boolean)** – the password is needed and is interactively requested from user.
- **IsReadOnly (Boolean)** – permits read-only access to the database.
- **NumberFormatsSupplier (Object)** – object containing the number formats available for the database (supports the `com.sun.star.util.XNumberFormatsSupplier` interface, refer to “[Number, Date and Text Format](#)” on page 132 section).
- **TableFilter (Array)** – list of table names to be displayed.
- **TableTypeFilter (Array)** – list of table types to be displayed. Values available are `TABLE`, `VIEW` and `SYSTEM TABLE`.
- **SuppressVersionColumns (Boolean)** – suppresses the display of columns that are used for version administration.

Note – The data sources from StarOffice are not 1:1 comparable with the data sources in ODBC. Whereas an ODBC data source only covers information about the origin of the data, a data source in StarOffice also includes a range of information about how the data is displayed within the database windows of StarOffice.

Queries

Predefined queries can be assigned to a data source. StarOffice notes the SQL commands of queries so that they are available at all times. Queries are used to simplify working with databases because they can be opened with a simple mouse click and also provide users without any knowledge of SQL with the option of issuing SQL commands.

An object which supports the `com.sun.star.sdb.QueryDefinition` service is concealed behind a query. The queries are accessed by means of the `QueryDefinitions` method of the data source.

The following example lists the names of data source queries can be established in a message box.

```
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

```

In addition to the Name property used in the example, the `com.sun.star.sdb.QueryDefinition` provides a whole range of other properties. These are:

- **Name (String)** – query name.
- **Command (String)** – SQL command (typically a `SELECT` command).
- **UpdateTableName (String)** – for queries that are based on several tables: name of table in which value modifications are possible.
- **UpdateCatalogName (String)** – name of update tables catalogues.
- **UpdateSchemaName (String)** – name of update tables diagrams.

The following example shows how a query object can be created in a program-controlled manner and can be assigned to a data source.

```

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)

```

The query object is first created using the `createUnoService` call, then initialized, and then inserted into the `QueryDefinitions` object by means of `insertByName`.

Links with Database Forms

To simplify work with data sources, StarOffice provides an option for linking the data sources with database forms. The links are available through the `getBookmarks()` method. This returns a named container (`com.sun.star.sdb.DefinitionContainer`) which contains all links of the data source. The bookmarks can either be accessed through `Name` or `Index`.

The following example determines the URL of the *MyBookmark* bookmark.

```

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

```

Database Access

A database connection is needed for access to a database. This is a transfer channel which permits direct communication with the database. Unlike the data sources presented in the previous section, the database connection must therefore be re-established every time the program is restarted.

StarOffice provides various ways of establishing database connections. Here is an explanation for the method based on an existing data source.

```

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

```

The code used in the example first checks whether the database is password protected. If not, it creates the database connection required using the `GetConnection` call. The two empty strings in the command line stand for the user name and password.

If the database is password protected, the example creates an `InteractionHandler` and opens the database connection using the `ConnectWithCompletion` method. The `InteractionHandler` ensures that StarOffice asks the user for the required login data.

Iteration of Tables

A table is usually accessed in StarOffice through the `ResultSet` object. A `ResultSet` is a type of marker that indicates a current set of data within a volume of results obtained using the `SELECT` command.

The example shows how a `ResultSet` can be used to query values from a database table.

```
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.getByNamed("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
```

Once the database connection has been established, the code used in the example first uses the `Connection.createObject` call to create a `Statement` object. This `Statement` object then uses the `executeQuery` call to return the actual `ResultSet`. The program now checks whether the `ResultSet` actually exists and traverses the data records using a loop. The values required (in the example, those from the `CustomerNumber` field) returns the `ResultSet` using the `getString` method, whereby the parameter 1 determines that the call relates to the values of the first column.

Note – The `ResultSet` object from SDBC is comparable with the `Recordset` object from DAO and ADO, since this also provides iterative access to a database.

Note – The database is actually accessed in StarOffice 8 through a `ResultSet` object. This reflects the content of a table or the result of a SQL-SELECT command. In the past, the `ResultSet` object provided the resident methods in the `Application` object for navigation within the data, for example, `DataNextRecord`).

Type-Specific Methods for Retrieving Values

As can be seen in the example from the previous section, StarOffice provides a `getString` method for accessing table contents. The method provides the result in the form of a string. The following `get` methods are available:

- `getBytes()` – supports the SQL data types for numbers, characters and strings.
- `getShort()` – supports the SQL data types for numbers, characters and strings.
- `getInt()` – supports the SQL data types for numbers, characters and strings.
- `getLong()` – supports the SQL data types for numbers, characters and strings.
- `getFloat()` – supports the SQL data types for numbers, characters and strings.
- `getDouble()` – supports the SQL data types for numbers, characters and strings.
- `getBoolean()` – supports the SQL data types for numbers, characters and strings.
- `getString()` – supports all SQL data types.
- `getBytes()` – supports the SQL data types for binary values.
- `getDate()` – supports the SQL data types for numbers, strings, date and time stamp.
- `getTime()` – supports the SQL data types for numbers, strings, date and time stamp.
- `getTimestamp()` – supports the SQL data types for numbers, strings, date and time stamp.
- `getCharacterStream()` – supports the SQL data types for numbers, strings and binary values.
- `getUnicodeStream()` – supports the SQL data types for numbers, strings and binary values.
- `getBinaryStream()` – binary values.
- `getObject()` – supports all SQL data types.

In all instances, the number of columns should be listed as a parameter whose values should be queried.

The ResultSet Variants

Accessing databases is often a matter of critical speed. StarOffice therefore provides several ways of optimizing `ResultSet`s and thereby controlling the speed of access. The more functions a `ResultSet` provides, the more complex its implementation usually is and therefore the slower the functions are.

A simple `ResultSet`, such as that which was presented in the "Iteration of tables" section, provides the minimum scope of functions available. It only allows iteration to be applied forward, and for values to be interrogated. More extensive navigation options, such as the possibility of modifying values, are therefore not included.

The `Statement` object used to create the `ResultSet` provides some properties which allow the functions of the `ResultSet` to be influenced:

- **ResultSetConcurrency (const)** – specifications as to whether the data can be modified (specifications in accordance with `com.sun.star.sdbc.ResultSetConcurrency`).
- **ResultSetType (const)** – specifications regarding type of `ResultSet`s (specifications in accordance with `com.sun.star.sdbc.ResultSetType`).

The values defined in `com.sun.star.sdbc.ResultSetConcurrency` are:

- **UPDATABLE** - `ResultSet` permits values to be modified.
- **READ_ONLY** - `ResultSet` does not permit modifications.

The `com.sun.star.sdbc.ResultSetConcurrency` group of constants provides the following specifications:

- **FORWARD_ONLY** – `ResultSet` only permits forward navigation.
- **SCROLL_INSENSITIVE** – `ResultSet` permits any type of navigation, changes to the original data are, however, not noted.
- **SCROLL_SENSITIVE** – `ResultSet` permits any type of navigation, changes to the original data impact on the `ResultSet`.

Note – A `ResultSet` containing the `READ_ONLY` and `SCROLL_INSENSITIVE` properties corresponds to a record set of the `Snapshot` type in ADO and DAO.

When using the `ResultSet`'s `UPDATABLE` and `SCROLL_SENSITIVE` properties, the scope of function of a `ResultSet` is comparable with a `Dynaset` type `Recordset` from ADO and DAO.

Methods for Navigation in ResultSets

If a `ResultSet` is a `SCROLL_INSENSITIVE` or `SCROLL_SENSITIVE` type, it supports a whole range of methods for navigation in the stock of data. The central methods are:

- **next()** – navigation to the next data record.
- **previous()** – navigation to the previous data record.
- **first()** – navigation to the first data record.
- **last()** – navigation to the last data record.
- **beforeFirst()** – navigation to before the first data record.
- **afterLast()** – navigation to after the last data record.

All methods return a Boolean parameter which specifies whether the navigation was successful.

To determine the current cursor position, the following test methods are provided and all return a Boolean value:

- **isBeforeFirst()** – `ResultSet` is before the first data record.
- **isAfterLast()** – `ResultSet` is after the last data record.
- **isFirst()** – `ResultSet` is the first data record.
- **isLast()** – `ResultSet` is the last data record.

Modifying Data Records

If a `ResultSet` has been created with the `ResultSetConcurrency = UPDATEABLE` value, then its content can be edited. This only applies for as long as the SQL command allows the data to be re-written to the database (depends on principle). This is not, for example, possible with complex SQL commands with linked columns or accumulated values.

The `ResultSet` object provides Update methods for modifying values, which are structured in the same way as the get methods for retrieving values. The `updateString` method, for example, allows a string to be written.

After modification, the values must be transferred into the database using the `updateRow()` method. The call must take place before the next navigation command, otherwise the values will be lost.

If an error is made during the modifications, this can be undone using the `cancelRowUpdates()` method. This call is only available provided that the data has not been re-written into the database using `updateRow()`.

Dialogs

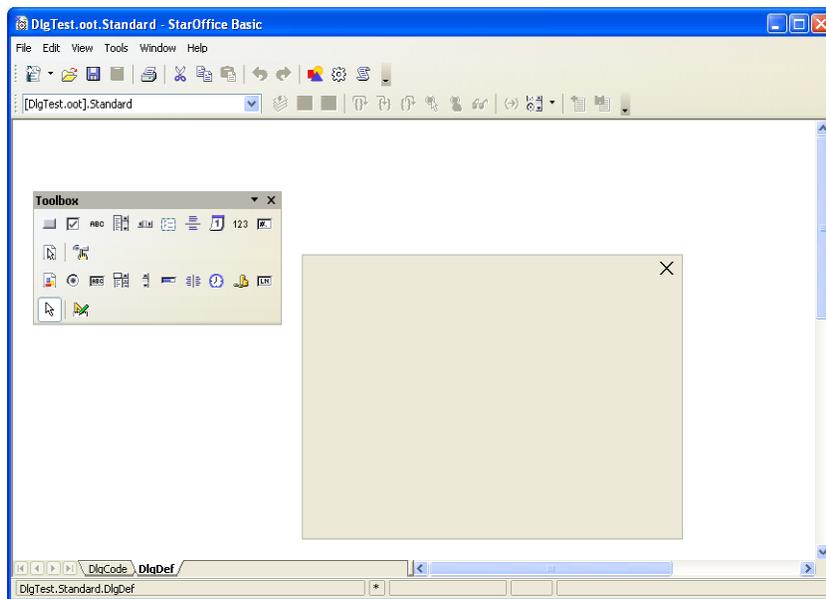
You can add custom dialog windows and forms to StarOffice documents. These in turn can be linked to StarOffice Basic macros to considerably extend the usage range of StarOffice Basic. Dialogs can, for example, display database information or guide users through a step-by-step process of creating a new document in the form of an AutoPilot.

Working With Dialogs

StarOffice Basic dialogs consist of a dialog window that can contain text fields, list boxes, radio buttons, and other control elements.

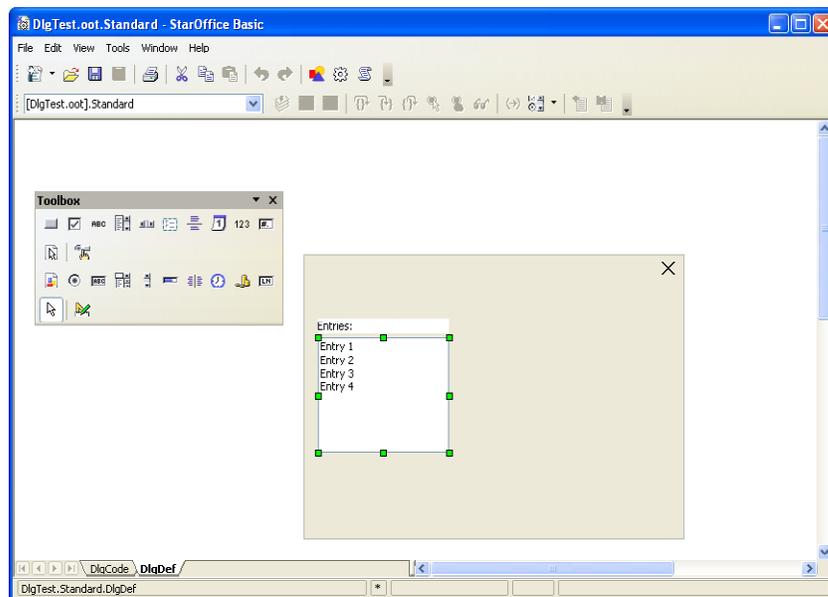
Creating Dialogs

You can create and structure dialogs using the StarOffice dialog editor that you can use in the same way as a StarOffice Draw:



Essentially, you drag the control elements that you want from the design pallet (right) into the dialog area where you can define their position and size.

The example shows a dialog that contains a label and a list box.



You can open a dialog with the following code:

```
Dim Dlg As Object

DialogLibraries.LoadLibrary("Standard")
Dlg = CreateUnoDialog(DialogLibraries.Standard.DlgDef)

Dlg.Execute()
Dlg.dispose()
```

`CreateUnoDialog` creates an object called `Dlg` that references the associated dialog. Before you can create the dialog, you must ensure that the library it uses (in this example, the `Standard` library) is loaded. If not, the `LoadLibrary` method performs this task.

Once the `Dlg` dialog object has been initialized, you can use the `Execute` method to display the dialog. Dialogs such as this one are described as modal because they do not permit any other program action until they are closed. While this dialog is open, the program remains in the `Execute` call.

The `dispose` method at the end of the code approves the resources used by the dialog once the program ends.

Closing Dialogs

Closing With OK or Cancel

If a dialog contains an **OK** or a **Cancel** button, the dialog is automatically closed when you press one of these buttons. More information about working with these buttons are discussed in this Dialog Control Elements in Detail section of this chapter.

If you close a dialog by clicking the **OK** button, the `Execute`-method returns a return value of 1, otherwise a value of 0 is returned.

```
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
```

Closing With the Close Button in the Title Bar

If you want, you can close a dialog by clicking the close button on the title bar of the dialog window. In this instance, the `Execute` method of the dialog returns the value 0, the same as when you press the Cancel button.

Closing With an Explicit Program Call

You can also close an open dialog window with the `endExecute` method:

```
Dlg.endExecute()
```

Access to Individual Control Elements

A dialog can contain any number of control elements. You can access these elements through the `getControl` method that returns the name of the control element.

```
Dim Ctl As Object

Ctl = Dlg.getControl("MyButton")
Ctl.Label = "New Label"
```

This code determines the object for the `MyButton` control element and then initializes the `Ctl` object variable with a reference to the element. Finally the code sets the `Label` property of the control element to the `New Label` value.

Note – StarOffice Basic distinguishes between uppercase and lowercase characters for the names of control elements.

Working With the *Model* of Dialogs and Control Elements

The division between visible program elements (*View*) and the data or documents behind them (*Model*) occurs at many places in StarOffice API. In addition to the methods and properties of control elements, both dialog and control element objects have a subordinate `Model` object. This object allows you to directly access the content of a dialog or control element.

In dialogs, the distinction between data and depiction is not always as clear as in other API areas of StarOffice. Elements of the API are available through both the *View* and the *Model*.

The `Model` property provides program-controlled access to the model of dialog and control element objects.

```
Dim cmdNext As Object  
  
cmdNext = Dlg.getControl("cmdNext")  
cmdNext.Model.Enabled = False
```

This example deactivates the `cmdNttext` button in the `Dlg` dialog with the aid of the model object from `cmdNttext`.

Properties

Name and Title

Every control element has its own name that can be queried using the following model property:

- **Model.Name (String)** – control element name

You can specify the title that appears in the title bar of a dialog with the following model property:

- **Model.Title (String)** – dialog title (only applies to dialogs).

Position and Size

You can query the size and position of a control element using the following properties of the model object:

- **Model.Height (long)** – height of control element (in ma units)
- **Model.Width (long)** – width of control element (in ma units)
- **Model.PositionX (long)** – X-position of control element, measured from the left inner edge of the dialog (in ma units)
- **Model.PositionY (long)** – Y-position of control element, measured from top inner edge of the dialog (in ma units)

To ensure platform independence for the appearance of dialogs, StarOffice uses the *Map AppFont (ma)* internal unit to specify the position and size within dialogs. An ma unit is defined as being one eighth of the average height of a character from the system font defined in the operating system and one quarter of its width. By using ma units, StarOffice ensures that a dialog looks the same on different systems under different system settings.

If you want to change the size or position of control elements for runtime, determine the total size of the dialog and adjust the values for the control elements to the corresponding part ratios.

Note – The Map AppFont (ma) replaces the Twips unit to achieve better platform independence.

Focus and Tabulator Sequence

You can navigate through the control elements in any dialog by pressing the Tab key. The following properties are available in this context in the control elements model:

- **Model.Enabled (Boolean)** – activates the control element
- **Model.Tabstop (Boolean)** – allows the control element to be reached through the Tab key
- **Model.TabIndex (Long)** – position of control element in the order of activation

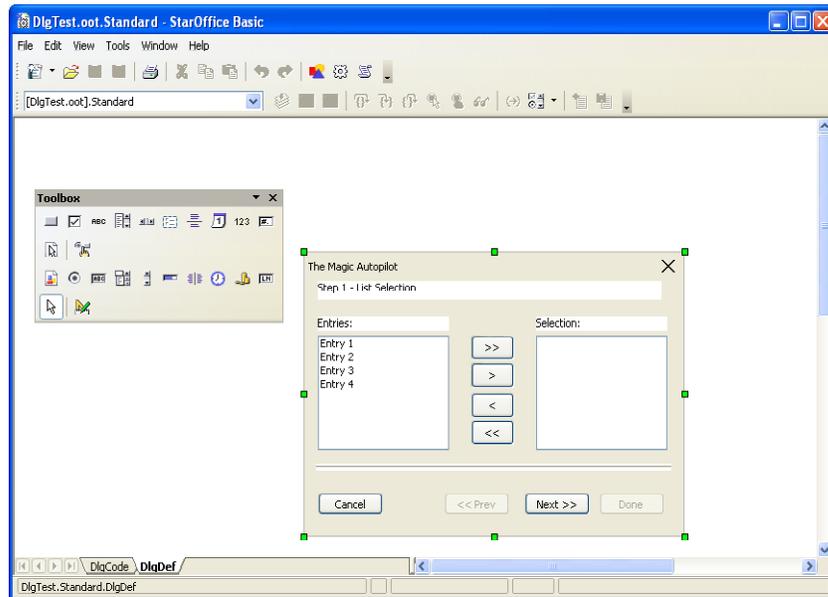
Finally, the control element provides a `getFocus` method that ensures that the underlying control element receives the focus:

- **getFocus** – control element receives the focus (only for dialogs)

Multi-Page Dialogs

A dialog in StarOffice can have more than one tab page. The `Step` property of a dialog defines the current tab page of the dialog whereas the `Step` property for a control element specifies the tab page where the control element is to be displayed.

The `Step`-value of 0 is a special case. If you set this value to zero in a dialog, all of the control elements are visible regardless of their `Step` value. Similarly, if you set this value to zero for a control element, the element is displayed on all of the tab pages in a dialog.



In the preceding example, you can also assign the `Step` value of 0 to the dividing line as well as the `Cancel`, `Prev`, `Next`, and `Done` buttons to display these elements on all pages. You can also assign the elements to an individual tab page (for example page 1).

The following program code shows how the `Step` value in event handlers of the `Next` and `Prev` buttons can be increased or reduced and changes the status of the buttons.

```
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")

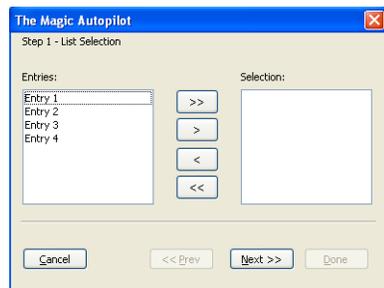
    cmdPrev.Model.Enabled = False
    cmdNext.Model.Enabled = True

    Dlg.Model.Step = Dlg.Model.Step - 1
End Sub

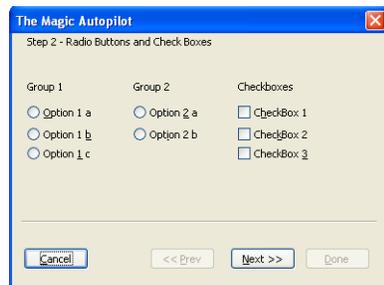
```

A global Dlg variable that references an open dialog must be included to make this example possible. The dialog then changes its appearance as follows:

Page 1:



Page 2:



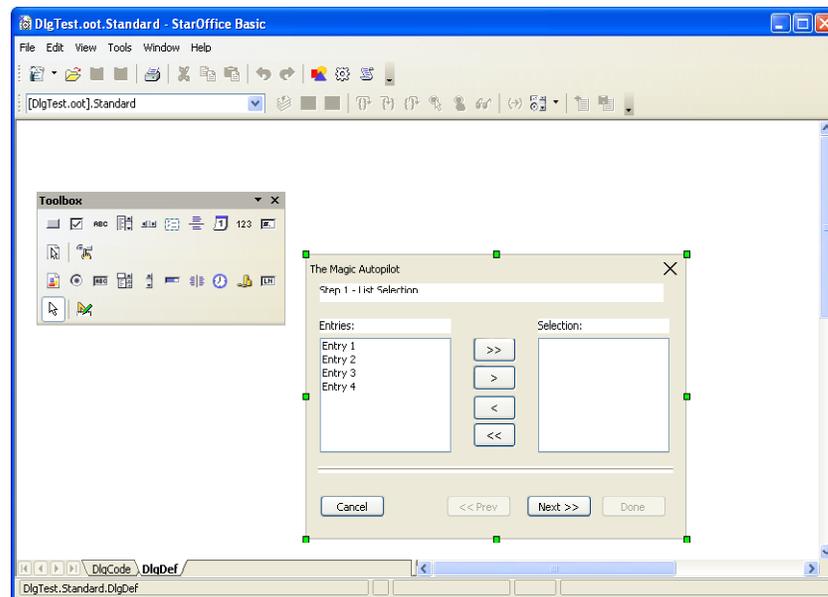
Events

StarOffice dialogs and forms are based on an event-oriented programming model where you can assign *event handlers* to the control elements. An event handler runs a predefined procedure when a particular action occurs, even when the action is another event. You can also edit documents or open databases with event handling as well as access other control elements.

StarOffice control elements recognize different types of events that can be triggered in different situations. These event types can be divided into four groups:

- **Mouse control:** Events that correspond to mouse actions (for example, simple mouse movements or a click on a particular screen location)
- **Keyboard control:** Events that are triggered by keyboard strokes
- **Focus modification:** Events that StarOffice perform when control elements are activated or deactivated
- **Control element-specific events:** Events that only occur in relation to certain control elements

When you work with events, ensure that you create the associated dialog in the StarOffice development environment and that it contains the required control elements or documents (if you the events apply to a form).



The preceding figure shows the StarOffice Basic development environment with a dialog window that contains two list boxes. You can move the data from one list to the other using the buttons between the two list boxes.

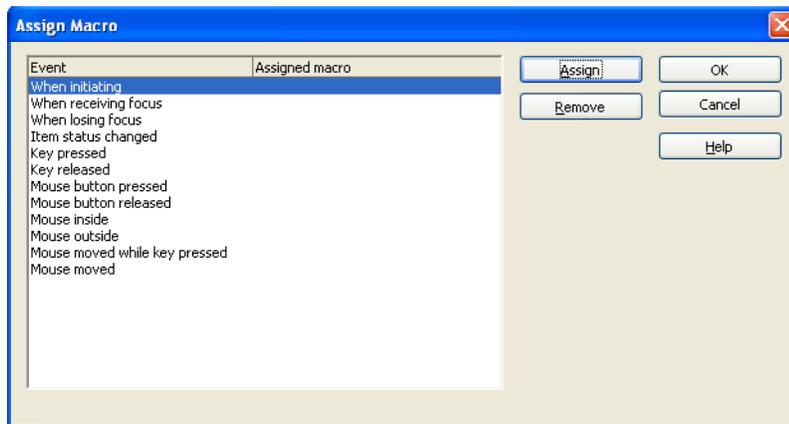
If you want to display the layout on screen, then you should create the associated StarOffice Basic procedures so that they can be called up by the event handlers. Even though you can use these procedures in any module, it is best to limit their use to two modules. To make your code easier to read, you should assign meaningful names to these procedures. Jumping directly to a general program procedure from a macro can result in unclear code. Instead, to simplify code maintenance and troubleshooting, you should create another procedure to serve as an entry point for event handling - even if it only executes a single call to the target procedure.

The code in the following example moves an entry from the left to the right list box of a dialog.

```
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.SelectItemPos, 1)
  Else
    Beep
  End If
End Sub
```

If this procedure was created in StarOffice Basic, you can assign it to an event required using the property window of the dialog editor.



The assignment dialog lists all of the StarOffice Basic procedures. To assign a procedure to an event, select the procedure, and then click **Assign**.

Parameters

The occurrence of a particular event is not always enough for an appropriate response. Additional information may be required. For example, to process a mouse click, you may need the screen position where the mouse button was pressed.

In StarOffice Basic, you can use object parameters to provide more information about an event to a procedure, for example:

```
Sub ProcessEvent(Event As Object)
```

```
End Sub
```

The accuracy with which the `Event` object is structured and its properties depend on the type of event that the procedure call triggers. The following sections describe event types in detail.

Regardless of the type of event, all objects provide access to the relevant control element and its model. The control element can be reached using

```
Event.Source
```

and its model using

```
Event.Source.Model
```

You can use these properties to trigger an event within an event handler.

Mouse Events

StarOffice Basic recognizes the following mouse events:

- **Mouse moved** – user moves mouse
- **Mouse moved while key pressed** – user drags mouse while holding down a key
- **Mouse button pressed** – user presses a mouse button
- **Mouse button released** – user releases a mouse button
- **Mouse outside** – user moves mouse outside of the current window

The structure of the associated event objects is defined in the `com.sun.star.awt.MouseEvent` structure which provides the following information:

- **Buttons (short)** – button pressed (one or more constants in accordance with `com.sun.star.awt.MouseButton`).
- **X (long)** – X-coordinate of mouse, measured in pixels from the top left corner of the control element
- **Y (long)** – Y-coordinate of mouse, measured in pixels from the top left corner of the control element

- **ClickCount (long)** – number of clicks associated with the mouse event (if StarOffice can respond fast enough, ClickCount is also 1 for a double-click because only an individual event is initiated).

The constants defined in `com.sun.star.awt.MouseButton` for the mouse buttons are:

- **LEFT** - left mouse button
- **RIGHT**- right mouse button
- **MIDDLE**- middle mouse button

The following example outputs the mouse position as well as the mouse button that was pressed:

```
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
```

Note – The VBA `Click` and `DoubleClick` events are not available in StarOffice Basic. Instead use the StarOffice Basic `MouseUp` event for the `click` event and imitate the `DoubleClick` event by changing the application logic.

Keyboard Events

The following keyboard events are available in StarOffice Basic:

- **Key pressed** – user presses a key
- **Key released** – user releases a key

Both events relate to *logical* key actions and not to *physical* actions. If the user presses several keys to output a single character (for example, to add an accent to a character), then StarOffice Basic only creates one event.

A single key action on a modification key, such as the Shift key or the Alt key does not create an independent event.

Information about a pressed key is provided by the event object that StarOffice Basic supplies to the procedure for event handling. It contains the following properties:

- **KeyCode (short)** – code of the pressed key (default values in accordance with `com.sun.star.awt.Key`)
- **KeyChar (String)** – character that is entered (taking the modification keys into consideration)

The following example uses the `KeyCode` property to establish if the Enter key, the Tab key, or one of the other control keys has been pressed. If one of these keys has been pressed, the name of the key is returned, otherwise the character that was typed is returned:

```
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
```

Information about other keyboard constants can be found in the API Reference under the `com.sun.star.awt.Key` group of constants.

Focus Events

Focus events indicate if a control element receives or loses focus. You can use these events to, for example, determine if a user has finished processing a control element so that you can update other elements of a dialog. The following focus events are available:

- **When receiving focus** – element receives focus
- **When losing focus** – element loses focus

The Event objects for the focus events are structured as follows:

- **FocusFlags (short)** – cause of focus change (default value in accordance with `com.sun.star.awt.FocusChangeReason`).

- **NextFocus (Object)** – object that receives focus (only for the `When losing focus` event)
- **Temporary (Boolean)** – the focus is temporarily lost

Control Element-Specific Events

In addition to the preceding events, which are supported by all control elements, there are also some control element-specific events that are only defined for certain control elements. The most important of these events are:

- **When Item Changed** – the value of a control element changes
- **Item Status Changed** – the status of a control element changes
- **Text modified** – the text of a control element changes
- **When initiating** – an action that can be performed when the control element is triggered (for example, a button is pressed)

When you work with events, note that some events, such as the `When initiating` event, can be initiated each time you click the mouse on some control elements (for example, on radio buttons). No action is performed to check if the status of the control element has actually changed. To avoid such “blind events”, save the old control element value in a global variable, and then check to see if the value has changed when an event is executing.

The properties for the `Item Status Changed` event are:

- **Selected (long)** – currently selected entry
- **Highlighted (long)** – currently highlighted entry
- **ItemId (long)** – ID of entry

Dialog Control Elements in Detail

StarOffice Basic recognizes a range of control elements which can be divided into the following groups:

Entry fields:

- Text fields
- Date fields
- Time fields
- Numerical fields
- Currency fields
- Fields adopting any format

Buttons:

- Standard buttons
- Checkboxes
- Radio Buttons

Selection lists:

- List boxes
- Combo-boxes

Other control elements:

- Scrollbars (horizontal and vertical)
- Fields of groups
- Progress bars
- Dividing lines (horizontal and vertical)
- Graphics
- File selection fields

The most important of these control elements are presented below.

Buttons

A button performs an action when you click it.

The simplest scenario is for the button to trigger a `When Initiating` event when it is clicked by a user. You can also link another action to the button to open a dialog using the `PushButtonType` property. When you click a button that has this property set to the value of 0, the dialog remains unaffected. If you click a button that has this property set to the value of 1, the dialog is closed, and the `Execute` method of the dialog returns the value 1 (dialog sequence has been ended correctly). If the `PushButtonType` has the value of 2, the dialog is closed and the `Execute` method of the dialog returns the value 0 (dialog closed).

The following are all of the properties that are available through the button model:

- **Model.BackgroundColor (long)** – color of background
- **Model.DefaultButton (Boolean)** – The button is used as the default value and responds to the Enter key if it has no focus.
- **Model.FontDescriptor (struct)** – structure that specifies the details of the font to be used (in accordance with `com.sun.star.awt.FontDescriptor` structure)
- **Model.Label (String)** – label that is displayed on the button
- **Model.Printable (Boolean)** – the control element can be printed
- **Model.TextColor (Long)** – text color of the control element
- **Model.HelpText (String)** – help text that is displayed when you move the mouse cursor over the control element

- **Model.HelpURL (String)** – URL of the online help for the corresponding control element
- **PushButtonType (short)** – action that is linked to the button (0: no action, 1: OK, 2: Cancel)

Option Buttons

These buttons are generally used in groups and allow you to select from one of several options. When you select an option, all of the other options in the group are deactivated. This ensures that at any one time, only one option button is set.

An option button control element provides two properties:

- **State (Boolean)** – activates the button
- **Label (String)** – label that is displayed on the button

You can also use the following properties from the model of the option buttons:

- **Model.FontDescriptor (struct)** – structure with details of the font to be used (in accordance with `com.sun.star.awt.FontDescriptor`)
- **Model.Label (String)** – label that is displayed on the control element
- **Model.Printable (Boolean)** – control element can be printed
- **Model.State (Short)** – if this property is equal to 1, the option is activated, otherwise it is deactivated
- **Model.TextColor (Long)** – text color of control element
- **Model.HelpText (String)** – help text that is displayed when the mouse cursor rests over the control element
- **Model.HelpURL (String)** – URL of online help for the corresponding control element

To combine several option buttons in a group, you must position them one after another in the activation sequence without any gaps (`Model.TabIndex` property, described as `Order` in the dialog editor). If the activation sequence is interrupted by another control element, then StarOffice automatically starts with a new control element group that can be activated regardless of the first group of control elements.

Note – Unlike VBA, you cannot insert option buttons in a group of control elements in StarOffice Basic. The grouping of control elements in StarOffice Basic is only used to ensure a visual division by drawing a frame around the control elements.

Checkboxes

Checkboxes are used to record a Yes or No value and depending on the mode, they can adopt two or three states. In addition to the Yes and No states, a check box can have an in-between state if the corresponding Yes or No status has more than one meaning or is unclear.

Checkboxes provide the following properties:

- **State (Short)** – state of the checkbox (0: no, 1: yes, 2: in-between state)
- **Label (String)** – label for the control element
- **enableTriState (Boolean)** – in addition to the activated and deactivated states, you can also use the in-between state

The model object of a checkbox provides the following properties:

- **Model.FontDescriptor (struct)** – structure with details of the font used (in accordance with `com.sun.star.awt.FontDescriptor` structure)
- **Model.Label (String)** – label for the control element
- **Model.Printable (Boolean)** – the control element can be printed
- **Model.State (Short)** – state of the checkbox (0: no, 1: yes, 2: in-between state)
- **Model.Tabstop (Boolean)** – the control element can be reached with the Tab key
- **Model.TextColor (Long)** – text color of control element
- **Model.HelpText (String)** – help text that is displayed when you rest the mouse cursor over the control element
- **Model.HelpURL (String)** – URL of online help for the corresponding control element

Text Fields

Text fields allow users to type numbers and text. The `com.sun.star.awt.UnoControlEdit` service forms the basis for text fields.

A text field can contain one or more lines and can be edited or blocked for user entries. Text fields can also be used as special currency and numerical fields as well as screen fields for special tasks. As these control elements are based on the `UnoControlEdit` Uno service, their program-controlled handling is similar.

Text fields provide the following properties:

- **Text (String)** – current text
- **SelectedText (String)** – currently highlighted text
- **Selection (Struct)** – read-only highlighting of details (structure in accordance with `com.sun.star.awt.Selection`, with the `Min` and `Max` properties to specify the start and end of the current highlighting)

- **MaxTextLen (short)** – maximum number of characters that you can type in the field
- **Editable (Boolean)** – `True` activates the option for entering text, `False` blocks the entry option (the property cannot be called up directly but only through `IsEditable`)
- **IsEditable (Boolean)** – the content of the control element can be changed, read-only.

Furthermore, the following properties are provided through the associated model object:

- **Model.Align (short)** – orientation of text (0: left-aligned, 1: centered, 2: right-aligned)
- **Model.BackgroundColor (long)** – color of the background of the control element
- **Model.Border (short)** – type of border (0: no border, 1: 3D border, 2: simple border)
- **Model.EchoChar (String)** – echo character for password fields
- **Model.FontDescriptor (struct)** – structure with details of font used (in accordance with `com.sun.star.awt.FontDescriptor` structure)
- **Model.HardLineBreaks (Boolean)** – automatic line breaks are permanently inserted in the control element text
- **Model.HScroll (Boolean)** – the text has a horizontal scrollbar
- **Model.MaxTextLen (Short)** – maximum length of text, where 0 corresponds to no length limit
- **Model.MultiLine (Boolean)** – permits entry to spans several lines
- **Model.Printable (Boolean)** – the control element can be printed
- **Model.ReadOnly (Boolean)** – the content of the control element is read-only
- **Model.Tabstop (Boolean)** – the control element can be reached with the Tab key
- **Model.Text (String)** – text associate with the control element
- **Model.TextColor (Long)** – text color of control element
- **Model.VScroll (Boolean)** – the text has a vertical scrollbar
- **Model.HelpText (String)** – help text that is displayed when the mouse cursor rests over the control element
- **Model.HelpURL (String)** – URL of online help for the corresponding control element

List Boxes

List boxes (`com.sun.star.awt.UnoControlListBox` service) support the following properties:

- **ItemCount (Short)** – number of elements, read-only

- **SelectedItem (String)** – text of highlighted entry, read-only
- **SelectedItems (Array Of Strings)** – data field with highlighted entries, read-only
- **SelectItemPos (Short)** – number of the entry highlighted at present, read-only
- **SelectItemsPos (Array of Short)** – data field with the number of highlighted entries (for lists which support multiple selection), read-only
- **MultipleMode (Boolean)** – True activates the option for multiple selection of entries, False blocks multiple selections (the property cannot be called up directly but only through `IsMultipleMode`)
- **IsMultipleMode (Boolean)** – permits multiple selection within lists, read-only

List boxes provide the following methods:

- **addItem (Item, Pos)** – enters the string specified in the `Item` into the list at the `Pos` position
- **addItems (ItemArray, Pos)** – enters the entries listed in the string's `ItemArray` data field into the list at the `Pos` position
- **removeItems (Pos, Count)** – removes `Count` entries as of the `Pos` position
- **selectItem (Item, SelectMode)** – activates or deactivates highlighting for the element specified in the string `Item` depending on the `SelectMode` Boolean variable
- **makeVisible (Pos)** – scrolls through the list field so that the entry specified with `Pos` is visible

The model object of the list boxes provides the following properties:

- **Model.BackgroundColor (long)** – background color of control element
- **Model.Border (short)** – type of border (0: no border, 1: 3D border, 2: simple border)
- **Model.FontDescriptor (struct)** – structure with details of font used (in accordance with `com.sun.star.awt.FontDescriptor` structure)
- **Model.LineCount (Short)** – number of lines in control element
- **Model.MultiSelection (Boolean)** – permits multiple selection of entries
- **Model.SelectedItems (Array of Strings)** – list of highlighted entries
- **Model.StringItemList (Array of Strings)** – list of all entries
- **Model.Printable (Boolean)** – the control element can be printed
- **Model.ReadOnly (Boolean)** – the content of the control element is read-only
- **Model.Tabstop (Boolean)** – the control element can be reached with the Tab key.
- **Model.TextColor (Long)** – text color of control element
- **Model.HelpText (String)** – automatically displayed help text which is displayed if the mouse cursor is above the control element
- **Model.HelpURL (String)** – URL of online help for the corresponding control element

Note – The VBA option for issuing list entries with a numerical additional value (`ItemData`) does not exist in StarOffice Basic. If you want to administer a numerical value (for example a database ID) in addition to the natural language text, you must create an auxiliary data field that administers in parallel to the list box.

Forms

In many respects, the structure of StarOffice-forms corresponds to the dialogs discussed in the previous chapter. There are, however, a few key differences:

- Dialogs appear in the form of one single dialog window, which is displayed over the document and does not permit any actions other than dialog processing until the dialog is ended. Forms, on the other hand, are displayed directly in the document, just like drawing elements.
- A dialog editor is provided for creating dialogs, and this can be found in the StarOffice Basic development environment. Forms are created using the **Form Functions Toolbar** directly within the document.
- Whereas the dialog functions are available in all StarOffice documents, the full scope of the form functions are only available in text and spreadsheets.
- The control elements of a form can be linked with an external database table. This function is not available in dialogs.
- The control elements of dialogs and forms differ in several aspects.

Users who want to provide their forms with their own methods for event handling, should refer to Chapter 11, Dialogs. The mechanisms explained there are identical to those for forms.

Working with Forms

StarOffice forms may contain text fields, list boxes, radio buttons, and a range of other control elements, which are inserted directly in a text or spreadsheet. The **Form FunctionsToolbar** is used for editing forms.

A StarOffice form may adopt one of two modes: the draft mode and the display mode. In draft mode, the position of control elements can be changed and their properties can be edited using a properties window.

The **Form Functions Toolbar** is also used to switch between modes.

Determining Object Forms

StarOffice positions the control elements of a form at drawing object level. The actual object form can be accessed through the Forms list at drawing level. The objects are accessed as follows in text documents:

```
Dim Doc As Object
Dim DrawPage As Object
Dim Form As Object

Doc = StarDesktop.CurrentComponent
DrawPage = Doc.DrawPage
Form = DrawPage.Forms.GetByIndex(0)
```

The `GetByIndex` method returns the form with the index number 0.

When working with spreadsheets, an intermediate stage is needed the Sheets list because the drawing levels are not located directly in the document but in the individual sheets:

```
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)
```

As is already suggested by the `GetByIndex` method name, a document may contain several forms. This is useful, for example, if the contents of different databases are displayed within one document, or if a 1:n database relationship is displayed within a form. The option of creating sub-forms is also provided for this purpose.

The Three Aspects of a Control Element Form

A control element of a form has three aspects:

- First, there is the *Model* of the control element. This is the key object for the StarOffice Basic-programmer when working with control element forms.
- The counterpart to this is the *View* of the control element, which administers the display information.
- Since control element forms within the documents are administered like a special drawing element, there is also a *Shape object* which reflects the drawing element-specific properties of the control element (in particular its position and size).

Accessing the Model of Control Element Forms

The models of the control elements of a form are available through the `GetByName` method of the `Object` form:

```
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")
```

The example determines the model of the `MyListBox` control element, which is located in the first form of the text document currently open.

If you is not sure of the form of a control element, you can use the option for searching through all forms for the control element required:

```
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
```

The example uses the `HasByName` method to check all forms of a text document to determine whether they contain a control element model called `MyListBox`. If a corresponding model is found, then a reference to this is saved in the `Ctl` variable and the search is terminated.

Accessing the View of Control Element Forms

To access the view of a control element form, the associated model is first needed. The view of the control element can then be determined with the assistance of the model and using the document controller.

```
Dim Doc As Object
Dim DocCtrl 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.GetCurrentController()
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

```

The code listed in the example is very similar to the code listed in the previous example for determining a control element model. It uses not only the `Doc` document object but also the `DocCrl` document controller object which makes reference to the current document window. With the help of this controller object and the model of the control element, it then uses the `GetControl` method to determine the view (`CtlView` variable) of the control element form.

Accessing the Shape Object of Control Element Forms

The method for accessing the shape objects of a control element also uses the corresponding drawing level of the document. To determine a special control element, all drawing elements of the drawing level must be searched through.

```

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

```

The example checks all drawing elements to determine whether they support the `com.sun.star.drawing.XControlShape` interface needed for control element forms. If this is the case, the `Control.Name` property then checks whether the name of the control element is `MyListBox`. If this is true, the function ends the search.

Determining the Size and Position of Control Elements

As already mentioned, the size and position of control elements can be determined using the associated `shape` object. The control element `shape`, like all other `shape` objects, provides the `Size` and `Position` properties for this purpose:

- **Size (struct)** - size of control element (`com.sun.star.awt.Size` data structure).
- **Position (struct)** - position of control element (`com.sun.star.awt.Point` data structure).

The following example shows how the position and size of a control element can be set using the associated `shape` object:

```
Dim Shape As Object

Point.x = 1000
Point.y = 1000
Size.Width = 10000
Size.Height = 10000

Shape.Size = Size
Shape.Position = Point
```

The `shape` object of the control element must already be known if the code is to function. If this is not the case, it must be determined using the preceding code.

Control Element Forms in Detail

The control elements available in forms are similar to those of dialogs. The selection ranges from simple text fields through list and combo boxes to various buttons.

Below, you will find a list of the most important properties for control element forms. All properties form part of the associated model objects.

In addition to the standard control elements, a table control element is also available for forms, which enables the complete incorporation of database tables. This is described in the [“Database Forms” on page 216](#) section in [Chapter 12](#).

Buttons

The model object of a form button provides the following properties:

- **BackgroundColor (long)** - background color.
- **DefaultButton (Boolean)** - the button serves as a default value. In this case, it also responds to the entry button if it has no focus.

- **Enabled (Boolean)** - the control element can be activated.
- **Tabstop (Boolean)** - the control element can be reached through the tabulator button.
- **TabIndex (Long)** - position of control element in activation sequence.
- **FontName (String)** - name of font type.
- **FontHeight (Single)** - height of character in points (pt).
- **Tag (String)** - string containing additional information, which can be saved in the button for program-controlled access.
- **TargetURL (String)** - target URL for buttons of the URL type.
- **TargetFrame (String)** - name of window (or frame) in which TargetURL is to be opened when activating the button (for buttons of the URL type).
- **Label (String)** - button label.
- **TextColor (Long)** - text color of control element.
- **HelpText (String)** - automatically displayed help text which is displayed if the mouse cursor is above the control element.
- **HelpURL (String)** - URL of online help for the corresponding control element.
- **ButtonType (Enum)** - action that is linked with the button (default value from `com.sun.star.form.FormButtonType`).

Through the `ButtonType` property, you have the opportunity to define an action that is automatically performed when the button is pressed. The associated `com.sun.star.form.FormButtonType` group of constants provides the following values:

- **PUSH** - standard button.
- **SUBMIT** - end of form entry (particularly relevant for HTML forms).
- **RESET** - resets all values within the form to their original values.
- **URL** - call of the URL defined in `TargetURL` (is opened within the window which was specified through `TargetFrame`).

The **OK** and **Cancel** button types provided in dialogs are not supported in forms.

Option Buttons

The following properties of an option button are available through its model object:

- **Enabled (Boolean)** - the control element can be activated.
- **Tabstop (Boolean)** - the control element can be reached through the tab key.
- **TabIndex (Long)** - position of control element in the activation sequence.
- **FontName (String)** - name of font type.
- **FontHeight (Single)** - height of character in points (pt).

- **Tag (String)** - string containing additional information, which can be saved in the button for program-controlled access.
- **Label (String)** - inscription of button.
- **Printable (Boolean)** - the control element can be printed.
- **State (Short)** - if 1, the option is activated, otherwise it is deactivated.
- **RefValue (String)** - string for saving additional information (for example, for administering data record IDs).
- **TextColor (Long)** - text color of control element.
- **HelpText (String)** - automatically displayed help text, which is displayed if the mouse cursor is above the control element.
- **HelpURL (String)** - URL of online help for the corresponding control element.

The mechanism for grouping option buttons distinguishes between the control elements for dialogs and forms. Whereas control elements appearing one after another in dialogs are automatically combined to form a group, grouping in forms is performed on the basis of names. To do this, all option buttons of a group must contain the same name. StarOffice combines the grouped control elements into an array so that the individual buttons of a StarOffice Basic program can be reached in the same way as previously.

The following example shows how the model of a control element group can be determined.

```
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
```

The code corresponds to the previous example for determining a simple control element model. It searches through all forms in the current text document in a loop and uses the `HasByName` method to check whether the corresponding form contains an element with the `MyOptions` name it is searching for. If this is the case, then the model array is accessed using the `GetGroupByName` method (rather than the `GetByName` method to determine simple models).

Checkboxes

The model object of a checkbox form provides the following properties:

- **Enabled (Boolean)** - the control element can be activated.
- **Tabstop (Boolean)** - the control element can be reached through the tab key.
- **TabIndex (Long)** - position of control element in the activation sequence.
- **FontName (String)** - name of font type.
- **FontHeight (Single)** - height of character in points (pt).
- **Tag (String)** - string containing additional information, which can be saved in the button for program-controlled access.
- **Label (String)** - button label.
- **Printable (Boolean)** - the control element can be printed.
- **State (Short)** - if 1, the option is activated, otherwise it is deactivated.
- **RefValue (String)** - string for saving additional information (for example, for administrating data record IDs).
- **TextColor (Long)** - text color of control element.
- **HelpText (String)** - automatically displayed help text, which is displayed if the mouse cursor is above the control element.
- **HelpURL (String)** - URL of online help for the corresponding control element.

Text Fields

The model objects of text field forms offer the following properties:

- **Align (short)** - orientation of text (0: left-aligned, 1: centered, 2: right-aligned).
- **BackgroundColor (long)** - background color of control element.
- **Border (short)** - type of border (0: no border, 1: 3D border, 2: simple border).
- **EchoChar (String)** - echo character for password field.
- **FontName (String)** - name of font type.
- **FontHeight (Single)** - height of character in points (pt).
- **HardLineBreaks (Boolean)** - the automatic line breaks are permanently inserted in the text of the control element.
- **HScroll (Boolean)** - the text has a horizontal scrollbar.
- **MaxTextLen (Short)** - maximum length of text; if 0 is specified, there are no limits.
- **MultiLine (Boolean)** - permits multi-line entries.
- **Printable (Boolean)** - the control element can be printed.
- **ReadOnly (Boolean)** - the content of the control element is read-only.
- **Enabled (Boolean)** - the control element can be activated.

- **Tabstop (Boolean)** - the control element can be reached through the tab key.
- **TabIndex (Long)** - position of the control element in the activation sequence.
- **FontName (String)** - name of font type.
- **FontHeight (Single)** - height of character in points (pt).
- **Text (String)** - text of control element.
- **TextColor (Long)** - text color of control element.
- **VScroll (Boolean)** - the text has a vertical scrollbar.
- **HelpText (String)** - automatically displayed help text, which is displayed if the mouse cursor is above the control element.
- **HelpURL (String)** - URL of online help for the corresponding control element.

List Boxes

The model object of the list box forms provides the following properties:

- **BackgroundColor (long)** - background color of control element.
- **Border (short)** - type of border (0: no border, 1: 3D frame, 2: simple frame).
- **FontDescriptor (struct)** - structure with details of font to be used (in accordance with `com.sun.star.awt.FontDescriptor` structure).
- **LineCount (Short)** - number of lines of control element.
- **MultiSelection (Boolean)** - permits multiple selection of entries.
- **SelectedItems (Array of Strings)** - list of highlighted entries.
- **StringItemList (Array of Strings)** - list of all entries.
- **ValueItemList (Array of Variant)** - list containing additional information for each entry (for example, for administrating data record IDs).
- **Printable (Boolean)** - the control element can be printed.
- **ReadOnly (Boolean)** - the content of the control element is read-only.
- **Enabled (Boolean)** - the control element can be activated.
- **Tabstop (Boolean)** - the control element can be reached through the tab key.
- **TabIndex (Long)** - position of control element in the activation sequence.
- **FontName (String)** - name of font type.
- **FontHeight (Single)** - height of character in points (pt).
- **Tag (String)** - string containing additional information which can be saved in the button for program-controlled access.
- **TextColor (Long)** - text color of control element.
- **HelpText (String)** - automatically displayed help text, which is displayed if the mouse cursor is above the control element.
- **HelpURL (String)** - URL of online help for the corresponding control element.

Note – Through their `ValueItemList` property, list box forms provide a counterpart to the VBA property, `ItemData`, through which you can administer additional information for individual list entries.

Furthermore, the following methods are provided through the view object of the list box:

- **addItem (Item, Pos)** - inserts the string specified in the `Item` at the `Pos` position in the list.
- **addItems (ItemArray, Pos)** - inserts the entries listed in the string's `ItemArray` data field in the list at the `Pos` position
- **removeItems (Pos, Count)** - removes `Count` entries as of the `Pos` position.
- **selectItem (Item, SelectMode)** - activates or deactivates the highlighting for the element specified in the string `Item` depending on the `SelectMode` variable.
- **makeVisible (Pos)** - scrolls through the list field so that the entry specified by `Pos` is visible.

Database Forms

StarOffice forms can be directly linked to a database. The forms created in this way provide all the functions of a full database front end without requiring independent programming work.

The user has the option of paging through and searching the selected tables and queries, as well as changing data records and inserting new data records. StarOffice automatically ensures that the relevant data is retrieved from the database, and that any changes made are written back to the database.

A database form basically corresponds to a standard StarOffice form. In addition to the standard properties, the following database-specific properties must also be set in the form:

- **DataSourceName (String)** - name of data source (refer to [Chapter 10](#) ; the data source must be globally created in StarOffice).
- **Command (String)** - name of table, query, or the SQL select command to which a link is to be made.
- **CommandType (Const)** - specifies whether the `Command` is a table, a query or a SQL command (value from `com.sun.star.sdb.CommandType` enumeration).

The `com.sun.star.sdb.CommandType` enumeration covers the following values:

- **TABLE** - Table
- **QUERY** - Query
- **COMMAND** - SQL command

The database fields are assigned to the individual control elements through this property:

- **DataField (String)** - name of linked database field.

Tables

Another control element is provided for work with databases: the table control element. This represents the content of a complete database table or query. In the simplest scenario, a table control element is linked to a database using the autopilot form, which links all columns with the relevant database fields in accordance with the user specifications. Because the associated API is relatively complex, we shall not provide a complete description of the API at this point.

