

[1 Tabellendokument](#)

[2 Tabellenblätter](#)

[-2.1 Zugriff über Index oder Name](#)

[-2.2 Einfügen, verschieben, kopieren und löschen](#)

[3 Spalten und Zeilen](#)

[-3.1 Optimale Spaltenbreite und Zeilenhöhe](#)

[4 Zellen](#)

[-4.1 Zellen ansprechen](#)

[-4.2 Die Inhalte von Zellen](#)

[-4.3 Löschen von Zellinhalten](#)

[-4.4 Text in Zellen](#)

[-4.5 Zelladressen](#)

[5 Zellbereiche](#)

[-5.1 Zellbereichsadressen](#)

[-5.2 Zellbereiche einfügen, entfernen, kopieren und verschieben](#)

[-5.3 Zellbereiche verbinden](#)

[-5.4 Benannte Zellbereiche](#)

[-5.5 Berechnungen mit Zellbereichen](#)

[6 Zellen oder Zellbereiche formatieren](#)

[-6.1 com.sun.star.table.CellProperties](#)

[-6.2 com.sun.star.style.CharacterProperties](#)

[-6.3 com.sun.star.style.ParagraphProperties](#)

[-6.4 Formate für Zahlen, Datum und Zeit](#)

[-6.5 Bedingte Formatierung](#)

[-6.6 Rahmen von Zellen und Zellbereichen](#)

[7 Selektionen in Calc](#)

[-7.1 Selektionen behandeln](#)

[-7.2 Zellbereiche selektieren](#)

[8 Notizen in Calc-Dokumenten](#)

[-8.1 Zugriff auf Notizen über Tabellenblatt](#)

[-8.2 Notizen hinzufügen und Zugriff über Zellen](#)

[9 Suchen und Ersetzen](#)

[-9.1 SearchDescriptor und ReplaceDescriptor](#)

[-9.2 Die Interfaces XSearchable und XReplaceable](#)

[10 Calc-Funktionen](#)

[-10.1 Aufruf von eingebauten Calc-Funktionen](#)

[-10.2 Eigene Calc-Funktionen in Basic](#)

[11 Formulare in Calc-Dokumenten](#)

OpenOffice.org-Basic: Einstieg in die Programmierung mit Calc

Autor:

[Marc Bächinger\(?\)](#) mbae@bcwin.ch

Korrektur gelesen:

letzte Änderung:

21.02.2004

Kontakt:Bächinger Consulting »<http://www.bcwin.ch> OpenOffice.org deutschsprachig »<http://de.openoffice.org>**Beiträge von:**

1 Tabellendokument

Ein Tabellendokument unterstützt den Service `com.sun.star.sheet.SpreadsheetDocument`. Ein Test nach der Unterstützung dieses Services dient dann auch zur Identifizierung eines Tabellendokumentes:

```
Sub aufCalcDokumentTesten
    ' Variable deklarieren
    Dim calcDokument as Object

    ' aktives Dokument holen
    calcDokument = ThisComponent

    ' testen ob es ein Calc-Dokument ist
    If Not calcDokument.supportsService _
        ( "com.sun.star.sheet.SpreadsheetDocument" ) then
        MsgBox "Das ist ein KEIN Calc-Dokument "
    Else
        MsgBox "Das ist ein Calc-Dokument "
    End If
End Sub
```

Eigenschaften des Service <code>com.sun.star.sheet.SpreadsheetDocument</code>	
XNamedRanges NamedRanges	Die benannten Zellbereiche eines Dokumentes
XDatabaseRanges DatabaseRanges	Die Datenbankbereiche eines Dokumentes
XLabelRanges ColumnLabelRanges	Die ColumnLabelRanges eines Dokuments (benannte Spalten?)
XLabelRanges RowLabelRanges	Die RowLabelRanges eines Dokumentes (benannte Zeilen?)
XNameAccess SheetLinks	Verweise zu Tabellenblätter
XAreaLinks AreaLinks	Verweise zu Bereichen
XNameAccess DDELinks	DDE links des Dokumentes (nur für Windows)

2 Tabellenblätter

Ein Tabellendokument besteht aus einem oder mehreren Tabellenblätter. Auf diese einzelnen Tabellenblätter kann über das Objekt `oCalcDokument.Sheets` zugegriffen werden. Das Objekt, das wir so erhalten unterstützt unter anderen die Interfaces `XNameAccess` und `XIndexAccess`, welche einen Zugriff über den Namen oder den Index des Tabellenblattes

ermöglichen.

2.1 Zugriff über Index oder Name

```
Sub tabellenblattAnsprechen
```

```
    ' Variablen deklarieren
    Dim oCalcDokument as Object
    Dim oTabellenblatt as Object

    ' aktuelles Dokument holen
    oCalcDokument = ThisComponent
    ' erstes Tabellenblatt über Index holen
    oTabellenblatt = oCalcDokument.Sheets(0)
    ' Name anzeigen
    MsgBox oTabellenblatt.Name

    ' zweites Tabellenblatt über Name holen
    oTabellenblatt = oCalcDokument.Sheets._
        getByName( "Tabelle2" )
    ' Name anzeigen
    MsgBox oTabellenblatt.Name
```

```
End Sub
```

2.2 Einfügen, verschieben, kopieren und löschen

Das Objekt, das über `calcDokument.Sheets` erreicht werden kann, unterstützt das Interface `com.sun.star.sheet.XSpreadsheets`. Dieses gibt uns die Möglichkeit neue Tabellenblätter hinzuzufügen, zu entfernen, zu kopieren oder zu verschieben. Der Zugriff auf die Tabellenblätter erfolgt bei diesen Methoden über den Namen.

```
Sub tabellenBlaetterSwap
```

```
    ' Variable deklarieren
    Dim oCalcDokument as Object

    ' aktives Dokument holen
    oCalcDokument = ThisComponent

    ' neues Tabelleblatt einfügen...
    oCalcDokument.Sheets.insertNewByName( "neuesblatt", 0 )
    MsgBox "weiter"

    ' ... ans Ende schieben...
    oCalcDokument.Sheets.moveByName( "neuesblatt", _
        oCalcDokument.Sheets.getCount() )
    MsgBox "weiter"

    ' ... kopieren ...
    oCalcDokument.Sheets.copyByName( "neuesblatt", _
        "neuesblatt2", 0 )
    MsgBox "weiter"

    ' ... und wieder alles löschen
    oCalcDokument.Sheets.removeByName( "neuesblatt" )
    oCalcDokument.Sheets.removeByName( "neuesblatt2" )
```

End Sub

3 Spalten und Zeilen

Ein Tabellenblatt besteht aus Zeilen und Spalten. Diese können über ihren Index (beginnend mit 0) direkt angesprochen werden. In der kürzesten Form gelangt man also mit `ThisComponent.Sheets(0).Columns(0)` an die Spalte A des ersten Tabellenblattes des aktiven Calc-Dokumentes. An Spalten und Zeilen lassen sich alle Formatierungen anwenden, welche man auch für Zellen oder Zellbereiche verfügbar sind. Durch eine Anweisung können so alle Zellen einer Spalte oder Zeile formatiert werden.

```
Sub zeilenUndSpalten
    ' Variablen deklarieren
    Dim oCalcDokument as Object
    Dim oZeile as Object
    Dim oSpalte as Object

    ' Calc-Dokument holen
    oCalcDokument = ThisComponent
    ' Spalte A auf Index 0 holen
    oSpalte = ThisComponent.Sheets(0).Columns(0)
    ' z.B. Zellhintergrund aller Zellen der Spalte einfärben
    oSpalte.CellBackColor = RGB ( 230, 230, 230 )

    ' Zeile 1 auf Index 0 holen
    oZeile = ThisComponent.Sheets(0).Rows(0)
    ' z.B. Schriftgrösse aller Zellen der Zeile setzten
    oZeile.CharHeight = 24
End Sub
```

Mit den Eigenschaften der Services `com.sun.star.table.TableRow` und `com.sun.star.table.TableColumn` lässt sich die Zeile bzw. Spalte unter anderem ein- und ausblenden oder in optimale Dimensionen setzen.

Eigenschaften von <code>com.sun.star.table.TableRow</code>	
Long Height	Höhe der Zeile in 1/100 Millimeter
Boolean OptimalHeight	Passt mit TRUE die Höhe der Zeile auf Inhalte an
Boolean IsVisible	TRUE für sichtbar und FALSE für ausgeblendet
Boolean IsStartOfNewPage	Manueller, vertikaler Seitenumbruch vor dieser Zeile wenn TRUE

Eigenschaften von <code>com.sun.star.table.TableColumn</code>	
Long Width	Breite der Spalte in 1/100 Millimeter
Boolean OptimalWidth	Passt mit TRUE die Breite der Spalte auf Inhalte an
Boolean IsVisible	TRUE für sichtbar und FALSE für ausgeblendet
Boolean IsStartOfNewPage	Manueller, horizontalen Seitenumbruch vor dieser Spalte wenn TRUE

3.1 Optimale Spaltenbreite und Zeilenhöhe

Nach Benutzereingaben aber auch nach der Eingabe von Zellwerten über OpenOffice.org-Basic sind die Inhalte einiger Spalten oder auch Zeilen oft nicht an die neuen Inhalte angepasst; sie sind zu breit oder zu schmal.

Über die Maus kann man in diesem Fall eine oder mehrere Spalte anwählen und mit einem Doppelklick auf die Zellbegrenzung eine angepasste Spaltenbreite erzwingen. Dasselbe lässt sich über die `com.sun.star.table.TableColumn` und `com.sun.star.table.TableRow` Services auch über Basic erreichen:

```
Sub zeilenUndSpaltenOptimaleGroesse
    ' optimale Höhe für Zeile 1 auf Index 0 , Tabellenblatt 0
    ThisComponent.Sheets(0).Rows(0).OptimalHeight = TRUE

    ' optimale Breite für Spalte A auf Index 0 , Tabellenblatt 0
    ThisComponent.Sheets(0).Columns(0).OptimalWidth = TRUE
End Sub
```

4 Zellen

Eine Zelle eines Tabellendokumente unterstützt eine Vielzahl von Interfaces. Einige die besonders wichtig erscheinen sind in der folgenden Tabelle 1 aufgeführt und lassen die umfassende Funktionalität eine Zelle erahnen. Viele der Interfaces von Zellen, Zellbereiche und Tabellenblätter sind übrigens identisch und werden exakt auf dieselbe Weise angewendet.

Unterstützte Interfaces vom Calc-Zellen*	
<code>c.s.s.util.XReplaceable</code>	Ermöglicht das Suchen und Ersetzen mit eine <code>ReplaceDescriptor</code> (Abschnitt 1.9)
<code>c.s.s.sheet.XCellAddressable</code>	Hat eine Zelladresse (Abschnitt 1.4.5)
<code>c.s.s.sheet.XCellRangeAddressable</code>	Hat eine Zellbereichsadresse (Abschnitt 1.5.1)
<code>c.s.s.sheet.XFormulaQuery</code>	Zugriff auf Zellen, welche über Funktionen mit dieser Zelle in Beziehung stehen oder solche, welche diese Zelle in ihren Funktionen verwenden. (Dependents/Precedents)
<code>c.s.s.sheet.XSheetAnnotationAnchor</code>	Anker für eine Notiz (Abschnitt 1.8)
<code>c.s.s.sheet.XSheetCellRange</code>	Eine Zelle ist auch ein Zellbereich
<code>c.s.s.sheet.XSheetOperation</code>	An einer Zelle lassen sich Grundberechnungen (<code>SUM()</code> , <code>AVG()</code> etc.) durchführen wie an einem Zellbereich (Abschnitt 1.5.5)
<code>c.s.s.table.XAutoFormattable</code>	ist auto-formatierbar
<code>c.s.s.table.XCell</code>	<code>XCell</code> von <code>com.sun.star.table</code>
<code>c.s.s.table.XCellRange</code>	Eine Zelle ist auch ein Zellbereich (Abschnitt 1.5)
<code>c.s.s.table.XColumnRowRange</code>	
<code>c.s.s.text.XText</code>	zu Bearbeitung von Zeichenketten in einer Zelle (Abschnitt 1.4.5)
<code>c.s.s.text.XTextFieldsSupplier</code>	für den Zugriff auf Textfelder innerhalb der Zelle
<code>c.s.s.util.XIndent</code>	zur Manipulation der Einrückung der Zelle
<code>c.s.s.util.XSearchable</code>	eine Zelle lässt sich durchsuchen (Abschnitt 1.9)

4.1 Zellen ansprechen

Eine Zelle kann über die Funktion `getCellByPosition(col, row)` eines Tabellenblattes angesprochen werden. Die Zelle A1 kommt dabei auf die Position 0/0 zu liegen, A2 auf 0/1, B1 auf 1/0 usw...

```
Sub einzelZelleUeberIndex
    ' Variablen deklarieren
    Dim oCalcDokument as Object
    Dim oTabellenblatt as Object
    Dim oZelle as Object

    ' aktuelles Dokument holen
    oCalcDokument = ThisComponent
    ' erstes Tabellenblatt holen
    oTabellenblatt = oCalcDokument.Sheets(0)

    ' Zelle A1 liegt auf Position 0,0
    oZelle = oTabellenblatt.getCellByPosition( 0, 0 )

    ' Zeichenkette in die Zelle schreiben
    oZelle.String = "Ein Text"

    MsgBox "weiter"
    ' lokalisierte Formel in die Zelle schreiben
    ' nicht empfehlenswert!!!!
    oZelle.FormulaLocal = "=SUMME( A2:A3 )"

    MsgBox "weiter"
    ' numerischen Wert in die Zelle schreiben
    oZelle.Value = 20.22
End Sub
```

Im Beispiel werden die Eigenschaften `String`, `Value` und `FormulaLocal` verwendet, um einer Zelle Werte zuzuweisen. Während die ersten Zeichenketten und numerische Werte betreffen, erlaubt es `Formula` oder `FormulaLocal` eine Calc-Funktion einzufügen.

`FormulaLocal` weist darauf hin, dass die Formel lokalisiert, also nicht in der Standardsprache, sondern in der eingestellten Sprache vorliegt. Es empfiehlt sich aber, nie die lokalisierten Namen der Formeln zu verwenden. Anwender, welche eine andere Lokalisierung verwenden, müssten sonst die Skripts umschreiben. Deshalb besser immer die Basissprache Englisch verwenden.

4.2 Die Inhalte von Zellen

Bei der Unterscheidung von Zellinhalten gibt es zwei verschiedene Ansätze, welche in verschiedenen Situationen von Bedeutung sind.

Geht es darum zu bestimmen, von welchem Typ der Wert einer Zelle ist, dann bedient man sich der Konstantengruppe `com.sun.star.table.CellContentType`:

Konstanten des Service <code>com.sun.star.table.CellContentType</code>	
EMPTY	Die Zelle ist leer
VALUE	ein numerischer Wert

TEXT	eine Zeichenkette
FORMULA	eine Funktion

Mittels dieser Konstantengruppe kann geprüft werden, von welchem Typ der Wert einer Zelle ist:

```
Sub inhaltsTypen
```

```
    ' Variablen deklarieren
    Dim oCalcDokument as Object
    Dim oTabellenblatt as Object
    Dim oZelle as Object
    Dim ausgabe as String

    ' aktuelles Dokument holen
    oCalcDokument = ThisComponent
    ' erstes Tabellenblatt holen
    oTabellenblatt = oCalcDokument.Sheets(0)
    ' Zelle A1 liegt auf Position 0,0
    oZelle = oTabellenblatt.getCellByPosition( 0, 0 )

    ' Inhalte auf Typ prüfen
    Select Case oZelle.Type
        Case com.sun.star.table.CellContentType.TEXT
            ausgabe = "Zelle enthält " & _
                "eine Zeichenkette"
        Case com.sun.star.table.CellContentType.VALUE
            ausgabe = "Zelle enthält " & _
                "einen numerischen Wert"
        Case com.sun.star.table.CellContentType.FORMULA
            ausgabe = "Zelle enthält " & _
                "eine Formel"
        Case com.sun.star.table.CellContentType.EMPTY
            ausgabe = "Zelle ist leer"
    End Select

    ' Info-Anzeige
    MsgBox ausgabe
```

```
End Sub
```

4.3 Löschen von Zellinhalten

Beim Löschen von Zellinhalten kommt eine andere Unterscheidung von Zellinhalten zur Anwendung. Diese ist aus dem Dialog Inhalte löschen bekannt. Selektiert man eine oder mehrere Zellen und drückt die Delete-Taste der Tastatur, so erscheint der Dialog und fragt nach, was denn alles gelöscht werden soll.

Die vorhandenen Optionen und wenige mehr sind auch über Basic verfügbar und können für das Löschen von Inhalten in Zellen oder Zellbereichen verwendet werden. Die Konstantengruppe `com.sun.star.sheet.CellFlags` bietet eine Reihe von Konstanten an, die dem Zweck dienen, bei einer Löschoption die Typen von Inhalten zu deklarieren, die gelöscht werden sollen.

Da die Werte der Konstanten der Binärreihe (1, 2, 4, 8 usw.) folgen, können diese einfach addiert werden und die Summe an die Methode `clearContents(nInhaltsTypen)` übergeben werden, die vom Interface `com.sun.star.sheet.XSheetOperation` bereitgestellt wird.

```

Sub zellInhalteLoeschen
  ' Variablen deklarieren
  Dim oCalcDokument as Object
  Dim oTabellenblatt as Object
  Dim oZelle as Object
  Dim oZellAdresse as Object
  Dim nInhaltsTypen as Long

  ' aktuelles Dokument holen
  oCalcDokument = ThisComponent
  ' erstes Tabellenblatt holen
  oTabellenblatt = oCalcDokument.Sheets(0)
  ' Zelle A1 liegt auf Position 0,0
  oZelle = oTabellenblatt.getCellByPosition( 0, 0 )

  ' addieren der Zellinhaltstypen, die
  ' gelöscht werden sollen
  nInhaltsTypen = com.sun.star.sheet.CellFlags.STRING + _
                 com.sun.star.sheet.CellFlags.VALUE + _
                 com.sun.star.sheet.CellFlags.FORMULA

  ' löschen über com.sun.star.sheet.XSheetOperation
  oZelle.clearContents( nInhaltsTypen )
End Sub

```

In der folgenden Tabelle sind alle Konstanten der Gruppe `com.sun.star.sheet.CellFlags` aufgeführt:

Konstanten von <code>com.sun.star.sheet.CellFlags</code>	
VALUE	numerische Werte
DATETIME	numerische Werte mit Datums- oder Zeitformat
STRING	Zeichenketten
ANNOTATION	Notizen
FORMULA	Funktionen
HARDATTR	Harte Formatierung, die nicht über Formatvorlagen angebracht wurden
STYLES	Formatvorlagen
OBJECTS	Zeichenobjekte
EDITATTR	???

4.4 Text in Zellen

Der einfachste Weg auf den Text in einer Zelle zuzugreifen, ist die Eigenschaft `String` einer Zelle:

```

Sub textInZelle
  ' Variablen deklarieren
  Dim oZelle as Object
  Dim sText as String

```

```

' Zelle A1 über Position holen
oZelle = ThisComponent.Sheets(0).getCellByPosition(0,0)
' Text anzeigen
MsgBox oZelle.String

```

```
End Sub
```

Zelltext erweitern

Um einem bereits existierenden Text in einer Zelle weiteren Text anzuhängen oder in diesen einzufügen, benötigen wir den Rückgriff auf einen Textcursor (XTextCursor), mit dem wir uns innerhalb des Textes bewegen können. Nachdem der Cursor im Text an die gewünschte Stelle bewegt worden ist, benutzt man die Methode `insertString(oTextCursor, sText, bAbsorb)`. Wird für den dritten Parameter jeweils `FALSE` übergeben, so wird der bestehende Text nicht überschrieben.

```

Sub textInZelleErweitern
' Variablen deklarieren
Dim oZelle as Object
Dim oTextCursor as Object

' Zelle A1 über Position holen
oZelle = ThisComponent.Sheets(0).getCellByPosition(0,0)
' TextCursor erzeugen
oTextCursor = oZelle.createTextCursor
' TextCursor ans Ende des Textes bewegen
oTextCursor.goToEnd( false )
' Text am Ende einfügen
oZelle.insertString( oTextCursor, " neuer Text am Ende" , false )
' TextCursor an den Anfang des Textes bewegen
oTextCursor.goToStart( false )
' Text am Anfang einfügen
oZelle.insertString( oTextCursor, "Und am Anfang " , false )

```

```
End Sub
```

Zelltext überschreiben

Wird der Cursor verschoben, so muss immer ein boolean-Parameter mitgegeben werden. Im obigen Beispiel wurde dabei immer `false` verwendet. Wird `true` übergeben, so wird der Text, über welchen sich der Cursor hinwegbewegt, selektiert. Das geschieht jedoch nur virtuell und wird nicht visuell angezeigt. Dennoch kann man die Vorstellung der Selektion mit der Maus als Analogie verwenden. Der so selektierte Text lässt sich dann ersetzen, indem bei der Methode `insertString(oTextCursor, sText, bAbsorb)` als Parameter `bAbsorb` auf `true` mitgegeben wird. Der Text wird 'absorbiert' oder eben überschrieben:

```

Sub textUeberschreiben
' Variablen deklarieren
Dim oZelle as Object
Dim oTextCursor as Object

' Zelle A1 über Position holen
oZelle = ThisComponent.Sheets(0).getCellByPosition(0,0)
' TextCursor erzeugen
oTextCursor = oZelle.createTextCursor

' TextCursor an den Anfang des Textes bewegen
oTextCursor.goToStart( false )

```

```

' drei Zeichen überspringen
oTextCursor.goRight( 3, false )
' TextCursor ans Ende des Textes bewegen
' und selektieren
oTextCursor.goToEnd( true )
' selektierten Text ersetzen
oZelle.insertString( oTextCursor, "ersetze Text" , true )

```

End Sub

Zelltext formatieren

Selektierter Text lässt sich nicht nur überschreiben, sondern auch formatieren, indem man nach der virtuellen Selektion mit dem TextCursor die Format-Eigenschaften des TextCursors manipuliert:

```

Sub textSelektieren
' Variablen deklarieren
Dim oZelle as Object
Dim oTextCursor as Object

' Zelle A1 über Position holen
oZelle = ThisComponent.Sheets(0).getCellByPosition(0,0)

' TextCursor erzeugen
oTextCursor = oZelle.createTextCursor
' TextCursor an den Anfang des Textes bewegen
oTextCursor.goToStart( false )

' TextCursor 1 Zeichen nach rechts
' und selektieren(true)
oTextCursor.goRight( 1 , true )
' Erstes Zeichen fett
oTextCursor.CharWeight = com.sun.star.awt.FontWeight.BOLD

' auch zweites Zeich in Selektion einschliessen
oTextCursor.goRight( 1 , true )
' erstes und zweites Zeichen 18 pt
oTextCursor.CharHeight = "18"

```

End Sub

4.5 Zelladressen

Ein Zelle enthält in der Eigenschaft `CellAddress` ein Struct vom Typ `com.sun.star.table.CellAddress`, das Informationen über die Position der Zelle im Dokument enthält.

Eigenschaften des Struct <code>com.sun.star.table.CellAddress</code>	
Sheet	Index des Tabellenblattes, welches den Zellbereich enthält
Column	Index der Spalte, in der die Zelle liegt
Row	Index der Zeile in der die Zelle liegt

Neben dem Index der Zeile und der Spalte kann über die Zelladresse auch der Index des Tabellenblattes in Erfahrung

gebracht werden:

```
Sub zelladresse
    ' Variablen deklarieren
    Dim oZelle as Object
    Dim oZellAdresse as Object

    ' Zelle A1 liegt auf Position 0,0 in Blatt 0
    oZelle = ThisComponent.Sheets(0).getCellByPosition( 1, 0 )

    ' Zelladresse holen
    oZellAdresse = oZelle.getCellAddress()

    ' Zeichenkette für Info-Ausgabe
    Dim ausgabe as String
    ausgabe = "Tabellenblatt: " & _
        oZellAdresse.Sheet & chr(13)
    ausgabe = ausgabe & "Spalte: " & _
        oZellAdresse.Column & chr(13)
    ausgabe = ausgabe & "Zeile: " & _
        oZellAdresse.Row

    ' Info anzeigen
    MsgBox ausgabe
End Sub
```

5 Zellbereiche

5.1 Zellbereichsadressen

Auch Zellbereiche können über eine Adresse angesprochen werden (`com.sun.star.table.CellRangeAddress`). Diese enthält die dreidimensionalen Koordinaten (Tabellenblatt, Zeile, Spalte) eines Zellbereiches.

```
Sub zellbereichsAdressen
    ' Variablen deklarieren
    Dim oZellBereich as Object
    ' neues Struct erzeugen
    Dim oZellbereichsAdresse as new com.sun.star.table.CellRangeAddress

    ' Zellbereichsadresse konfigurieren
    oZellbereichsAdresse.Sheet = 0
    oZellbereichsAdresse.StartColumn = 0
    oZellbereichsAdresse.StartRow = 0
    oZellbereichsAdresse.EndColumn = 3
    oZellbereichsAdresse.EndRow = 4

    ' Zellbereich holen
    oZellbereich = ThisComponent.Sheets( oZellbereichsAdresse.Sheet )._
        getCellRangeByPosition( oZellbereichsAdresse.StartColumn, _
oZellbereichsAdresse.StartRow, _
oZellbereichsAdresse.EndColumn, _
```

```

oZellbereichsAdresse.EndRow )

' ... und selektieren
ThisComponent.currentController.select( oZellbereich )
End Sub

```

Eigenschaften des Struct <code>com.sun.star.table.CellRangeAddress</code>	
short Sheet	Index des Tabellenblattes, welches den Zellbereich enthält
long StartColumn	Index der Spalte am linken Ende des Zellbereiches.
long StartRow	Index der Zeile am oberen Ende des Zellbereiches.
long EndColumn	Index der Spalte am rechten Ende des Zellbereiches
long EndRow	Index der Zeile am unteren Ende des Zellbereiches

5.2 Zellbereiche einfügen, entfernen, kopieren und verschieben

Das Einfügen, Löschen, Kopieren und Verschieben von Zellen und Zellbereichen wird über das Interface `com.sun.star.sheet.XCellRangeMovement` ermöglicht, das von einem Tabellenblatt exportiert wird.

Methoden des Interfaces <code>com.sun.star.sheet.XCellRangeMovement</code>	
<code>insertCells(oCellrange, nCellInsertMode)</code>	Zellen einfügen, verschiebt die anderen Zellen nach unten oder nach rechts
<code>removeRange(cellrange, nCellDeleteMode)</code>	Zellen löschen, verschiebt andere Zellen nach oben oder nach links
<code>moveRange(oDestinationCell, oCellRange)</code>	Zellen nach einem anderen Ort im Dokument verschieben
<code>copyRange(oDestinationCell, oCellRange)</code>	Zellen nach einem anderen Ort im Dokument kopieren

```

Sub zellenEinfuegenVerschieben
' Variablen deklarieren
Dim oZellBereich as Object
Dim oTabelleblatt as Object
' neues Struct erzeugen
Dim oZellbereichsAdresse as new com.sun.star.table.CellRangeAddress
' neues Struct erzeugen
Dim oZielZellAdresse as new com.sun.star.table.CellAddress

' Zellbereichsadresse konfigurieren
oZellbereichsAdresse.Sheet = 0
oZellbereichsAdresse.StartColumn = 0

```

```

oZellbereichsAdresse.StartRow = 1
oZellbereichsAdresse.EndColumn = 2
oZellbereichsAdresse.EndRow = 2

' Zelladresse konfigurieren
oZielZellAdresse.Sheet = 1
oZielZellAdresse.Column = 0
oZielZellAdresse.Row = 0

' Tabellenblatt implementiert XCellRangeMovement
oTaschenblatt = ThisComponent.Sheets(0)

' kopieren nach zweitem Tabellenblatt
oTaschenblatt.copyRange( oZielZellAdresse, oZellbereichsAdresse )
' einfügen eines Zellbereichs
oTaschenblatt.insertCells( oZellbereichsAdresse, _
    com.sun.star.sheet.CellInsertMode.DOWN )
' verschieben nach zweitem Tabellenblatt
oTaschenblatt.copyRange( oZielZellAdresse, oZellbereichsAdresse )
' entfernen eines Zellbereichs
oTaschenblatt.removeRange( oZellbereichsAdresse, _
    com.sun.star.sheet.CellDeleteMode.UP )

```

End Sub

5.3 Zellbereiche verbinden

Zellbereich können zu einer einzelnen Zelle verbunden werden. Über die grafische Oberfläche geschieht das über den Menüpunkt Format ? Zelle zusammenfassen ? Festlegen. In OOO-Basic get der Weg über das Interface `com.sun.star.util.XMergeable`, welches die Methode `merge(boolean)` anbietet und von einem Zellbereich (`XCellRange`) implementiert wird.

Die Anwendung ist denkbar einfach. Mit dem boolean-Parameter wird angegeben, ob der Zellbereich verbunden (`true`) oder die Verbindung aufgehoben werden soll (`false`). Eine weitere Methode `getIsMerged():boolean` lässt prüfen, ob ein Zellbereich bereits verbunden wurde oder nicht:

```

Sub zellbereichVerbindenOderAufheben
' Variablen deklarieren
Dim oZellBereich as Object

' Zellbereich holen
oZellBereich = ThisComponent.Sheets(0)._
    getCellRangeByPosition( 0, 0, 4, 0 )
' Zellbereich zu einer einzigen Zelle verbinden oder Verbindung aufheben
If oZellBereich.getIsMerged() then
    oZellBereich.merge( false )
Else
    oZellBereich.merge( true )
End If
End Sub

```

5.4 Benannte Zellbereiche

In Calc kann einem Zellbereich über Einfügen > Name > Festlegen ein Name verliehen werden, über den ein Zellbereich angesprochen werden kann. Solche benannte Zellbereiche können über die Methode `getCellRangeByName()` eines Tabellenblattes angesprochen werden. Anstatt eines Namens des Zellbereiches wie `A1:C10` kann einfach der verliehene Name verwendet werden.

```
Sub benannterZellbereich
    ' Variablen deklarieren
    Dim oZellbereich as Object

    ' benannter Zellbereich mit dem Namen "test" holen
    oZellbereich = ThisComponent.Sheets(0).getCellRangeByName( "test" )

    ' benannter Zellbereich selektieren
    ThisComponent.currentController.select( oZellbereich )
End Sub
```

5.5 Berechnungen mit Zellbereichen

Die Enumeration `com.sun.star.sheet.GeneralFunction` enthält einige Grundfunktionen, welche auf einen Zellbereich angewendet werden können. Über diese Grundfunktionen lassen sich einige Berechnungen auf einfachste Art und Weise anstellen.

```
Sub grundFunktionenAnwenden
    ' Variablen deklarieren
    Dim oCalcDokument as Object
    Dim oSelektion as Object
    Dim sMessage as String

    sMessage = "berechnete Werte:" & chr(13)
    ' aktuelles Dokument holen
    oCalcDokument = ThisComponent
    ' aktuelle Selektion holen
    ' (unterstützt XCellRange)
    oSelektion = oCalcDokument.getCurrentSelection()

    ' Summe berechnen
    sMessage = sMessage & "Summe: " & _
        oSelektion.computeFunction( _
            com.sun.star.sheet.GeneralFunction.SUM )_
        & chr(13)
    ' maximaler Wert
    sMessage = sMessage & "Max: " & _
        oSelektion.computeFunction( _
            com.sun.star.sheet.GeneralFunction.MAX )_
        & chr(13)
    ' Durchschnitt
    sMessage = sMessage & "Durchschnitt: " & _
        oSelektion.computeFunction( _
            com.sun.star.sheet.GeneralFunction.AVERAGE )_
        & chr(13)
    ' Anzahl numerischer Werte
    sMessage = sMessage & "Anzahl num. Werte: " & _
        oSelektion.computeFunction( _
```

```
com.sun.star.sheet.GeneralFunction.COUNTNUMS )_
& chr(13)
```

```
' an Benutzer ausgeben
MsgBox sMessage
```

```
End Sub
```

com.sun.star.sheet.GeneralFunction	
NONE	Keine Berechnung
AUTO	Funktion wird automatisch determiniert
SUM	berechnet die Summe aller numerischen Werte
COUNT	Alle Werte, auch alphanumerische, werden gezählt (Anzahl Werte)
AVERAGE	berechnet den Durchschnitt aller numerischen Werte
MAX	maximaler Wert aller numerischer Werte wird evaluiert
MIN	minimaler Wert aller numerischen Werte wird evaluiert
PRODUCT	berechnet das Produkt aller numerischen Werte
COUNTNUMS	zählt die numerischen Werte
STDEV	berechnet die Standardabweichung basierend auf einer Stichprobe
STDEVP	berechnet die Standardabweichung basierend auf der Grundgesamtheit
VAR	berechnet die Varianz basierend auf einer Stichprobe
VARP	berechnet die Varianz basierend auf die Grundgesamtheit

6 Zellen oder Zellbereiche formatieren

Die meisten Zellformatierungen können über die drei Eigenschaftsgruppen `c.s.s.table.CellProperties`, `c.s.s.style.CharacterProperties` und `c.s.s.style.ParagraphProperties` angebracht werden. Die Anwendung dieser erfolgt über das Setzen dieser Eigenschaften, welche an Zellen, Zellbereichen und ganzen Tabellenblätter angebracht werden können:

6.1 com.sun.star.table.CellProperties

`com.sun.star.table.CellProperties` können auf Zellen, Zellbereiche sowie Spalten und Zeilen angewendet werden. Im folgenden Beispiel wird das Cell-Property `CellBackColor` einer Zelle auf ein helles Grau gesetzt.

```
Sub cellProperties
    ' Variablen deklarieren
    Dim oZelle as Object
    ' hole Zelle A1
    oZelle = ThisComponent.Sheets(0).getCellByPosition( 0, 0 )

    ' setzen des CharacterProperty CellBackColor
    oZelle.CellBackColor = RGB( 200, 200 , 200 )
End Sub
```

Eigenschaft	mögliche Werte
--------------------	-----------------------

String CellStyle »OPTIONAL	Name des Styles der Zelle
Long CellBackColor	Hintergrundfarbe der Zelle: RGB(nRot, Ngruen, Nblau)
Boolean IsCellBackgroundTransparent	True, wenn der Hintergrund transparent ist
HoriJustify	Horizontale Ausrichtung der Zelle com.sun.star.table.CellHoriJustify.RIGHT com.sun.star.table.CellHoriJustify.CENTER com.sun.star.table.CellHoriJustify.LEFT com.sun.star.table.CellHoriJustify.STANDARD
VertJustify	Vertikale Ausrichtung der Zelle com.sun.star.table.CellVertJustify.STANDARD com.sun.star.table.CellVertJustify.TOP com.sun.star.table.CellVertJustify.CENTER com.sun.star.table.CellVertJustify.BOTTOM
Boolean IsTextWrapped	True, wenn der Text automatisch umgebrochen wird
Long ParaIndent	Einrückung des Zellinhaltes in 1/100 mm
Orientation	Orientierung des Zellinhaltes com.sun.star.table.CellOrientation.STANDARD com.sun.star.table.CellOrientation.TOPBOTTOM com.sun.star.table.CellOrientation.BOTTOMTOP com.sun.star.table.CellOrientation.STACKED
Long RotateAngle	Rotation des Zellinhaltes in 1/100 Grad
RotateReference	die Ecke um die rotiert wird
AsianVerticalMode »OPTIONAL	Wählt vertikale, asiatische Zeichen-Orientierung aus
TableBorder	Beschreibung des Rahmens um die Zelle oder den Zellbereich
TopBorder	Beschreibung des oberen Rahmens
BottomBorder	Beschreibung des unteren Rahmens
LeftBorder	Beschreibung des linken Rahmens
RightBorder	Beschreibung des rechten Rahmens
NumberFormat	Index des Zahlenformats
ShadowFormat	Format der Schattierung der Zelle
CellProtection	Beschreibung des Zellschutzes
UserDefinedAttributes	Zusätzlich gespeicherte Attribute

6.2 com.sun.star.style.CharacterProperties

com.sun.star.style.CharacterProperties können auf Zellen, Zellbereiche sowie Spalten und Zeilen angewendet werden. Im folgenden Beispiel wird das CharacterProperty CharHeight einer Zelle auf 24 Punkte gesetzt.

```
Sub characterProperties
    ' Variablen deklarieren
    Dim oZelle as Object
    ' hole Zelle A1
    oZelle = ThisComponent.Sheets(0).getCellByPosition( 0, 0 )

    ' setzen des CharacterProperty CellBackColor
```

```
oZelle.CharHeight = 24
```

```
End Sub
```

Eigenschaft	mögliche Werte
Long CharColor	Zeichenfarbe: RGB(nRotwert, nGruenwert, nBlauwert) Farbeanteile zwischen 0 und 255
Boolean CharContoured	Zeichen mit Contouren:
Boolean CharCrossedOut	Durchgestrichene Zeichen
Integer CharEmphasis	Betonung
Integer CharFont	Schrift
Integer CharFontCharSet	
Integer CharFontCharSetAsian	
Integer CharFontCharSetComplex	
Integer CharFontFamily	Schriftfamilie
Integer CharFontFamilyAsian	asiatische Schriftfamilie
Integer CharFontFamilyComplex	
String CharFontName	Schriftname
String CharFontNameAsian	
String CharFontNameComplex	
Integer CharFontPitch	
Integer CharFontPitchAsian	
Integer CharFontPitchComplex	
String CharFontStyleName	Zeichen-Formatvorlage
String CharFontStyleNameAsian	
String CharFontStyleNameComplex	
Single CharHeight	Zeichengrösse in Punkten
Single CharHeightAsian	
Single CharHeightComplex	
Object CharLocale	Sprache der Zeichen
Object CharLocaleAsian	
Object CharLocaleComplex	
Long CharPosture	
Long CharPostureAsian	
Long CharPostureComplex	
Integer CharRelief	
Boolean CharShadowed	Schattierung der Zeichen
Integer CharStrikeout	
Integer CharUnderline	Unterstreichung der Zeichen
Long CharUnderlineColor	Farbe der Unterstreichung

Boolean CharUnderlineHasColor	Unterstreichung in Farbe (TRUE/FALSE)
Single CharWeight	Zeichendicke
Single CharWeightAsian	
Single CharWeightComplex	

6.3 com.sun.star.style.ParagraphProperties

Eigenschaft	mögliche Werte
Integer ParaAdjust	com.sun.star.style.ParagraphAdjust.LEFT com.sun.star.style.ParagraphAdjust.RIGHT com.sun.star.style.ParagraphAdjust.BLOCK com.sun.star.style.ParagraphAdjust.CENTER com.sun.star.style.ParagraphAdjust.STRETCH
Integer ParaIndent	
Integer ParaLastLineAdjust	
Long ParaBottomMargin	Abstand nach unten in 1/100 Millimeter
Long ParaLeftMargin	Abstand links in 1/100 Millimeter
Long ParaRightMargin	Abstand nach rechts in 1/100 Millimeter
Long ParaTopMargin	Abstand nach oben in 1/100 Millimeter

6.4 Formate für Zahlen, Datum und Zeit

6.5 Bedingte Formatierung

Einer Zelle oder einem Zellbereich können bedingte Formatierungen angebracht werden, die wirksam werden, wenn die Bedingungen zutreffen. Diese praktische Eigenschaft kann auch über Basic genutzt werden.

Die Services `com.sun.star.table.XCell` und `com.sun.star.sheet.XSheetCellRange` erlauben über die Eigenschaft `ConditionalFormat` bzw. `ConditionalFormatLocal` Zugriff auf ein Objekt, welches das Interface `com.sun.star.sheet.XSheetConditionalEntries` unterstützt und einen `XIndexAccess` bzw. `XElementAccess` auf die bedingten Formatierungen einer Zelle oder eines Zellbereiches anbietet.

Die in `XSheetConditionalEntries` enthaltenen Objekte unterstützen den Service `com.sun.star.sheet.TableConditionalEntry`, welche die Interfaces `com.sun.star.sheet.XSheetConditionalEntry` und `com.sun.star.sheet.XSheetCondition` exportiert:

Eigenschaften von <code>com.sun.star.sheet.XSheetConditionalEntry</code>	
String StyleName	Name der Formatvorlage

Eigenschaften von `com.sun.star.sheet.XSheetCondition`

Long Operator	logischer Operator der Bedingung NONE Keine Bedingung definiert EQUAL Wert muss gleich dem spezifizierten Wert in Formula1 sein NOT_EQUAL Wert muss ungleich dem spezifizierten Wert in Formula1 sein GREATER Wert muss grösser als der spezifizierte Wert in Formula1 sein GREATER_EQUAL Wert muss gleich oder grösser als der spezifizierte Wert in Formula1 sein LESS Wert muss kleiner als der spezifizierte Wert in Formula1 sein LESS_EQUAL Wert muss gleich oder kleiner als der spezifizierte Wert in Formula1 sein BETWEEN Wert muss zwischen den spezifizierten Werten in Formula1 und Formula2 sein NOT_BETWEEN Der Wert muss ausserhalb der beiden spezifizierten Werte in Formula1 und Formula2 sein FORMULA Der in Formula1 spezifizierte Ausdruck muss ein Resultat ergeben, das nicht gleich 0 ist.
String Formula1	Erste Formel als Zeichenkette, die den auszuwertenden Ausdruck enthält. Kann z.B. ?10? sein oder aber auch ?SUM(A1:A20) > 100?
String Formula2	Zweite Formel als Zeichenkette, die den auszuwertenden Ausdruck enthält. Kann z.B. ?10? sein oder aber auch ?SUM(A1:A20) > 100?
Object SourcePosition	Die Zelle oder der Zellbereich, zu dem die Bedingung gehört

Mit diesem Wissen sind wir in der Lage die bedingten Formatierungen über Basic zu prüfen, zu entfernen oder zu erstellen.

```

Sub bedingteFormateEinerZelleAnzeigen
  ' Variablen deklarieren
  Dim oZelle as Object
  Dim oBedingtesFormat as Object
  Dim nCounter as Integer
  Dim sAusgabe as String

  ' Zelle A1 holen
  oZelle = ThisComponent.Sheets(0).getCellByPosition( 0, 0 )

  ' Schleife durch alle bedingten Formate
  For nCounter = 0 To oZelle.ConditionalFormat.Count-1

    ' bedingtes Formate holen
    oBedingtesFormat = oZelle.ConditionalFormat(nCounter)
    ' Ausgabe der Eigenschaften
    sAusgabe = (nCounter+1) & ". Bedingtes Format:" & chr(13)
    sAusgabe = sAusgabe & "Operator: " & _
      oBedingtesFormat.Operator & chr(13)
    sAusgabe = sAusgabe & "Formel 1: " & _
      oBedingtesFormat.Formula1 & chr(13)
    sAusgabe = sAusgabe & "Formel 2: " & _
      oBedingtesFormat.Formula2 & chr(13)
    sAusgabe = sAusgabe & "Formatvorlage: " & _

```

```

                oBedingtesFormat.StyleName & chr(13)
            ' Ausgabe anzeigen
            MsgBox sAusgabe
        Next nCounter
End Sub

```

Folgendes Code-Beispiel fügt ein neues bedingtes Format zu den bedingten Formaten der Zelle A1 hinzu. Während über Format ? Bedingte Formatierung im Dialog Bedingte Formatierung nur drei bedingte Formatierungen definiert werden können (wirklich?), scheint dies über Basic keiner Beschränkung zu unterliegen. In meinen Versuchen wurden jedenfalls auch Formatierungen an zehnter Stelle ausgewertet und angewendet.

```

Sub bedingteFormateEinerZelleHinzufuegen
    ' Variablen deklarieren
    Dim oZelle as Object
    ' die bedingten Formate als XIndexAccess, XElementAccess
    Dim oBedingteFormate as Object

    ' Zelle A1 holen
    oZelle = ThisComponent.Sheets(0).getCellByPosition( 0, 0 )

    ' bedingte Formate der Zelle holen
    oBedingteFormate = oZelle.ConditionalFormat

    ' neues bedingtes Format als PropertyValue
    Dim oBedingung(3) as New com.sun.star.beans.PropertyValue
    ' Eigenschaften des Formates definieren
    oBedingung(0).Name = "Operator"
    oBedingung(0).Value = com.sun.star.sheet.ConditionOperator.FORMULA
    oBedingung(1).Name = "Formulal"
    oBedingung(1).Value = "SUM(A1:A10) > 100"
    oBedingung(2).Name = "StyleName"
    oBedingung(2).Value = "Heading1"

    ' das neue Format zu den bereits bestehenden hinzufügen
    oBedingteFormate.addNew( oBedingung() )
    oBedingteFormate.clear()

    ' wichtig! Bedingte Formate in zelle setzen
    oZelle.ConditionalFormat = oBedingteFormate
End Sub

```

6.6 Rahmen von Zellen und Zellbereichen

Um den Rahmen einer Zelle oder eines Zellbereiches zu manipulieren, können die Eigenschaften TopBorder, BottomBorder, LeftBorder und RightBorder verwendet werden. Sowohl Zellen als auch Zellbereiche bieten diese Eigenschaften an, über welche die entsprechenden Rahmenlinien bestimmt werden können.

Die Definition der Eigenschaften einer Linie erfolgt über das Struct `com.sun.star.table.BorderLine`, welches vier Eigenschaften besitzt:

Eigenschaften des Struct `com.sun.star.table.BorderLine`

Long Color	Farbe der Linie. Z.B. RGB(0,255,0) für ein hässliches Blau
Short/Integer InnerLineWidth	Breite der inneren Linie der Doppellinie (in 1/100 mm)
Short/Integer OuterLineWidth	Breite der äusseren Linie der Doppellinie (in 1/100 mm)
Short/Integer LineDistance	Distanz zwischen den beiden Linien (in 1/100 mm)

Der Rahmen eine Zelle oder eines Zellbereiches besteht aus einer Doppellinie. Die Stärken beider Linien und der Abstand zwischen den beiden können unabhängig definiert werden. Ist aber InnerLine gleich 0, so wird auch die äussere Linie nicht angebracht (Ooo1.lrc1).

```
Sub zellRahmenDirekt
  ' Variable deklarieren
  Dim oZellbereich as Object
  Dim oRahmenLinie as Object

  oZellbereich = ThisComponent.Sheets(0).getCellRangeByName( "B2:B8" )
  ' Rahmenlinie erstellen
  oRahmenLinie = CreateUnoStruct("com.sun.star.table.BorderLine")
  ' Eigenschaften der Linie definieren
  With oRahmenLinie
    .Color = RGB( 255, 0, 0 )
    .InnerLineWidth = 0
    .OuterLineWidth = 8
    .LineDistance = 0
  End With

  ' einzelne Rahmen setzen
  oZellbereich.TopBorder = oRahmenLinie
  oZellbereich.BottomBorder = oRahmenLinie
  oZellbereich.LeftBorder = oRahmenLinie
  oZellbereich.RightBorder = oRahmenLinie
End Sub
```

Mit dem direkten Zugriff die Rahmenlinien der Zellen können jedoch nur die einzelnen Zellen angesteuert werden. Einen Zellbereich nur als ganzes zu umrahmen ist mit diesem direkten Zugriff nicht möglich. Die Rahmenformatierung wird immer auf alle Zellen angewendet, so dass ein Gitternetz entsteht. Soll nur der äussere Rahmen eines ganze Zellbereiches gesetzt werden, so muss ein Struct vom Typ `com.sun.star.table.TableBorder` erstellt und über das Property `TableBorder` einer Zelle oder eines Zellbereiches zugewiesen werden.

Das Struct `com.sun.star.table.TableBorder`

Das Struct `com.sun.star.table.TableBorder` beschreibt den Rahmen eines Zellbereiches. Jeder Zelle oder Zellbereich kann über die Eigenschaft `TableBorder` ein Rahmen zugewiesen werden, der den äusseren Rahmen, sowie die Trennlinien zwischen den einzelnen Zellen beschreibt:

Struct <code>com.sun.star.table.TableBorder</code>	
BorderLine TopLine	oberer Linienstil als <code>com.sun.star.table.BorderLine</code>
boolean IsTopLineValid	TopLine-Eigenschaft verwenden oder nicht
BorderLine BottomLine	unterer Linienstil als <code>com.sun.star.table.BorderLine</code>

boolean IsBottomLineValid	BottomLine-Eigenschaft verwenden oder nicht
BorderLine LeftLine	linker Linienstil als com.sun.star.table.BorderLine
boolean IsLeftLineValid	LeftLine-Eigenschaft verwenden oder nicht
BorderLine RightLine	rechter Linienstil als com.sun.star.table.BorderLine
boolean IsRightLineValid	RightLine-Eigenschaft verwenden oder nicht
BorderLine HorizontalLine	Linienstil der horizontalen Trennlinien zwischen den einzelnen Zellen als com.sun.star.table.BorderLine
boolean IsHorizontalLineValid	HorizontalLine-Eigenschaft verwenden oder nicht
BorderLine VerticalLine	Linienstil der vertikalen Trennlinien zwischen den einzelnen Zellen als com.sun.star.table.BorderLine
boolean IsVerticalLineValid	VerticalLine-Eigenschaft verwenden oder nicht
Short/Integer Distance	Distanz zwischen der Linie und dem Inhalt
boolean IsDistanceValid	Distance-Eigenschaft verwenden oder nicht

Die Linien-Eigenschaften sind alle ein Struct vom Typ com.sun.star.table.BorderLine. Wird ein Rahmen eines Zellbereiches gesetzt, so entscheiden die horizontale und vertikale Linie, ob ein Rahmen bloss um den ganzen Zellbereich oder um jede Zelle (Gitternetz) gezogen wird. Horizontale und vertikale Linien definieren folglich die Trennlinien zwischen den Zellen eines Zellbereiches.

Im folgenden Beispiel wird der Zellbereich B2:D4 mit einem dünnen, roten Rahmen umgeben. Den Eigenschaften HorizontalLine und VerticalLine werden leere BorderLines zugeordnet, was bewirkt, dass keine Linie gezeichnet wird und auch einfach weggelassen werden kann.

Sub zellRahmen

```

' Variable deklarieren
Dim oZellBereich as Object
Dim oRoteLinie as Object
Dim oTableBorder as Object
' Zellbereich über Name holen
oZellBereich = ThisComponent.Sheets(0).getCellRangeByName( "B2:D4" )

' BorderLine erzeugen
oRoteLinie = CreateUnoStruct("com.sun.star.table.BorderLine")
' Linien-Eigenschaften setzen
With oRoteLinie
    .Color = RGB( 255, 0, 0 )
    .InnerLineWidth = 0
    .OuterLineWidth = 8
    .LineDistance = 0
End With

' Tabellenrahmen erzeugen
oTableBorder = createUnoStruct("com.sun.star.table.TableBorder")
' Eigenschaften setzen
With oTableBorder
    .TopLine = oRoteLinie
    .IsTopLineValid = True
    .BottomLine = oRoteLinie
    .IsBottomLineValid = True

```

```

        .LeftLine = oRoteLinie
        .IsLeftLineValid = True
        .RightLine = oRoteLinie
        .IsRightLineValid = True
        .HorizontalLine = _
            CreateUnoStruct("com.sun.star.table.BorderLine")
        .IsHorizontalLineValid = True
        .VerticalLine = _
            CreateUnoStruct("com.sun.star.table.BorderLine")
        .IsVerticalLineValid = True
        .Distance = 0
        .IsDistanceValid = True
    End With

    ' Rahmen zuweisen
    oZellBereich.TableBorder = oTableBorder
End Sub

```

7 Selektionen in Calc

7.1 Selektionen behandeln

Um an die aktuelle Selektion in einem Calc-Dokument zu erhalten, benutzt man die Funktion `getCurrentSelection()` des Tabellendokumentes:

```

Dim calcDokument as Object
Dim selektion as Object

' aktuelles Dokument holen
calcDokument = ThisComponent
' aktuelle Selektion holen
selektion = calcDokument.getCurrentSelection()

```

In Calc-Dokumenten können drei verschiedene Selektionen unterschieden werden:

Selektion	Rückgabewert von <code>getCurrentSelection()</code>
eine einzelne Zelle	<code>com.sun.star.sheet.SheetCell</code>
ein einzelner Zellbereich	<code>com.sun.star.sheet.SheetCellRange</code>
mehrere Zellbereiche	<code>com.sun.star.sheet.SheetCellRanges</code>

Natürlich ist es manchmal notwendig den Typ der Selektion zu kennen. Dies erreichen Sie mit einem Test nach den unterstützten Services:

```

Sub selektierteZellen
    Dim calcDokument as Object
    Dim selektion as Object

    ' aktuelles Dokument holen
    calcDokument = ThisComponent
    ' aktuelle Selektion holen

```

```

selektion = calcDokument.getCurrentSelection()

' Selektion kann null(?), eine einzelne Zelle,
' ein einzelner oder mehrere Zellbereiche sein
If IsNull( selektion ) then
    ' keine Selektion (überhaupt möglich? wie?)
    MsgBox "Keine Selektion"

Elseif selektion.supportsService( _
    "com.sun.star.sheet.SheetCell" ) then
    ' einzelne Zelle selektiert
    MsgBox "Einzelne Zelle selektiert"

Elseif selektion.supportsService( _
    "com.sun.star.sheet.SheetCellRange" ) then
    ' einzelner Zellbereich selektiert
    MsgBox "Einzelner Zellbereich selektiert"

Elseif selektion.supportsService( _
    "com.sun.star.sheet.SheetCellRanges" ) then
    ' mehrere Zellbereiche selektiert
    MsgBox "Mehrere Zellbereiche selektiert"

End If
End Sub

```

Ist der Typ der Selektion einmal bestimmt, kann auf die Interfaces und Eigenschaften der entsprechenden Objekt-Typen zugegriffen werden. In Tabelle 2 sind die Objekt-Typen aufgeführt, welche bei den verschiedenen Selektionen zurückgegeben werden. In den beiden ersten Fällen der SheetCell- und SheetCellRange-Objekte kann mit diesen analog zu den Beispielen in Kapitel zu den Zellen bzw. Zellbereichen verfahren werden.

Ein Spezialfall ist die Selektion von mehreren Zellbereichen mittels der Ctrl-Taste. Das Container-Objekt SheetCellRanges enthält dabei die einzelnen Zellbereiche. Im folgenden soll nur der Fall der Mehrfachselektion mit einem Codebeispiel ausgeführt werden. Für die anderen Fälle sei auf die entsprechenden Kapitel über Calc-Zellen und Zellbereiche verwiesen.

CellRanges unterstützen die Interfaces com.sun.star.container.XIndexAccess und com.sun.star.container.XEnumerationAccess, welche Zugriff auf die enthaltenen Zellbereiche erlauben. Im folgenden Beispiel wird über das XIndexAccess-Interface auf die Zellbereiche zugegriffen:

```

Sub mehrfachSelektion
    Dim selektion as Object
    Dim ausgabe as String
    Dim zellBereichAdresse as Object

    selektion = ThisComponent.getCurrentSelection()

    If Not( selektion.supportsService( _
        "com.sun.star.sheet.SheetCellRanges" ) ) then
        MsgBox "keine Mehrfachselektion"
        Exit Sub
    End If

    ausgabe = "Mehrere Zellbereiche ausgewählt" & chr(13)

    ' Anzahl der Zellbereiche

```

```

ausgabe = ausgabe & "Anzahl Bereiche: " & _
           selektion.getCount() & chr(13)

' Schleife durch alle Zellbereiche
for i=0 To selektion.getCount()-1
  ' Adresse des Zellbereiches holen
  zellBereichAdresse = selektion._
                       getByIndex(i).getRangeAddress()
  ' Adressinfo an Ausgabe anfügen
  ausgabe = ausgabe & _
            zellBereichAdresse.StartColumn & " " & _
            zellBereichAdresse.StartRow & "->" & _
            zellBereichAdresse.EndColumn & " " & _
            zellBereichAdresse.EndRow & chr(13)
Next i

' Ausgabe der Informationen über MessageBox
MsgBox ausgabe
End Sub

```

7.2 Zellbereiche selektieren

Um einen Zellbereich zu selektieren, dient die Methode `select(cellRange)` des `currentControllers`. Für den direkten Aufruf bedient man sich demnach `ThisComponent.currentController.select(cellRange)`.

```

Sub zellbereichSelektieren
  ' Variable deklarieren
  Dim oZellbereich as Object

  ' Zellbereich über Name holen ...
  oZellbereich = ThisComponent.Sheets(0).getCellRangeByName( "A1:B10" )
  ' ... und selektieren
  ThisComponent.currentController.select( oZellbereich )

  ' warten bis OK gedrückt wird
  MsgBox "weiter"

  ' einzelne Zelle über Bereichsname holen
  oZellbereich = ThisComponent.Sheets(0).getCellRangeByName( "A1" )
  ' ... und selektieren
  ThisComponent.currentController.select( oZellbereich )
End Sub

```

8 Notizen in Calc-Dokumenten

Ein Calc-Dokument kann Notizen (Annotations) enthalten, die einer Zelle zugeordnet sind und über `oZelle.getAnnotation()` angesprochen werden kann. Zudem bietet ein Tabellenblatt (Sheet) über das Feld `Annotations` einen `com.sun.star.container.XIndexAccess` über alle Notizen eines Blattes.

Eine Notiz bzw. das Interface `com.sun.star.sheet.XSheetAnnotation` bietet folgende Methoden an:

com.sun.star.sheet.XSheetAnnotation	
Position	Die com.sun.star.table.CellAddress der Zelle, zu welcher die Notiz zugeordnet ist.
Author (nur lesen)	Der Autor der Notiz
Date (nur lesen)	Das Datum als Zeichenkette
IsVisible	Gibt TRUE zurück wenn sichtbar, andernfalls FALSE
setVisible(boolean)	Methode um Notiz anzuzeigen bzw. auszublenden.
getParent()	gibt die Zelle zurück, in der die Notiz enthalten ist.

8.1 Zugriff auf Notizen über Tabellenblatt

Das folgende Beispiel iteriert über alle Notizen des ersten Tabellenblattes eines Calc-Dokumentes:

```

Sub notizenEinesTabelleblattes
    ' Variable deklarieren
    Dim oNotizen as Object
    Dim oNotiz as Object
    Dim oZelle as Object
    Dim nCounter as Integer
    Dim sAusgabe as String

    ' XIndexAccess für Notizen (Annotations) des Tabellenblattes
    oNotizen = ThisComponent.Sheets(0).Annotations

    for nCounter = 0 To oNotizen.Count - 1
        ' Notiz über Index holen
        oNotiz = oNotizen.getByIndex( nCounter )

        ' Notizangaben ausgeben
        sAusgabe = "Notiz: " & _
            oNotiz.String & chr(13)
        sAusgabe = sAusgabe & "Autor: " & _
            oNotiz.Author & chr(13)
        sAusgabe = sAusgabe & "Datum: " & _
            oNotiz.Date & chr(13)

        ' die Zelle, in der die Notiz enthalten ist
        oZelle = oNotiz.getParent()
        ' Zelladresse ermitteln
        sAusgabe = sAusgabe & "Zelle: " & _
            oZelle.CellAddress.Column & "/" & _
            oZelle.CellAddress.Row

        ' Notiz einblenden
        oNotiz.setVisible( TRUE )

        ' anzeige der Notizinfo an Benutzer
        MsgBox sAusgabe

        ' Notiz ausblenden
        oNotiz.setVisible( FALSE )
    
```

```

Next nCounter

If nCounter = 0 then
    MsgBox "Keine Notiz in diesem Dokument"
End if
End Sub

```

Während `getPosition()` eine Zelladresse liefert kann über die Methode `getParent()`, die über das unterstützte Interface `com.sun.star.container.XChild` zur Verfügung gestellt wird, direkt auf das Zell-Objekt zugegriffen werden.

8.2 Notizen hinzufügen und Zugriff über Zellen

Neben dem Zugriff auf die Notizen über das Tabellenblatt, lässt sich eine Notiz auch über die Zelle erreichen. `oZelle.Annotation` liefert die Notiz. Um eine neue Notiz zu erstellen reicht es aus die Eigenschaft von `oZelle.Annotation` zu setzen. Auch wenn einer Zelle noch keine Notiz zugeordnet wurde, erhält man über `oZelle.Annotation` eine Notiz. Es ist nicht nötig eine neue Instanz zu erzeugen.

```

Sub NotizHinzufuegen
    ' Variable deklarieren
    Dim oZelle as Object

    ' Zelle A1 holen
    oZelle = ThisComponent.Sheets(0).getCellByPosition(0,0)

    ' Notiz einfügen oder überschreiben
    oZelle.Annotation.String = "Eine Notiz"

    ' Notiz einblenden..
    oZelle.Annotation.IsVisible = TRUE
    ' warten bis OK gedrückt wird
    MsgBox "weiter"
    ' ... und wieder ausblenden
    oZelle.Annotation.IsVisible = FALSE
End Sub

```

9 Suchen und Ersetzen

9.1 SearchDescriptor und ReplaceDescriptor

Das Suchen und/oder Ersetzen von Text in einem Calc-Dokument erfolgt über den Service `com.sun.star.util.SearchDescriptor` bzw. `com.sun.star.util.ReplaceDescriptor`, welche so konfiguriert werden können, wie das auch über den Suchen und Ersetzen-Dialog von Calc gemacht werden könnte.

Eigenschaften des Services <code>com.sun.star.util.XSearchDescriptor</code>	
String SearchString	Die Zeichenkette, die das Suchmuster enthält (Text, Format oder Regular Expression)

Eigenschaften des Services <code>com.sun.star.util.SearchDescriptor</code>	
Boolean SearchBackwards	Sucht rückwärts, wenn TRUE

Boolean SearchCaseSensitive	Gross-/Kleinschreibung wird beachtet, wenn TRUE
Boolean SearchWords	Nur ganze Wörter, wenn TRUE
Boolean SearchRegularExpression	SearchString wird als Regulärer Ausdruck behandelt, wenn TRUE
Boolean SearchStyles	Wenn TRUE, wird nicht nach Text sondern nach dem Namen einer Absatzvorlage gesucht, das auf den SearchString passt
Boolean SearchSimilarity	Ähnlichkeitsuche, wenn TRUE
Boolean SearchSimilarityRelax	Alle Ähnlichkeitsregeln miteinander anwenden, wenn TRUE
short/Integer SearchSimilarityRemove	Anzahl der Zeichen, welche ignoriert werden sollen (Ähnlichkeitssuche)
short/Integer SearchSimilarityAdd	Anzahl der Zeichen die hinzugefügt werden sollen (Ähnlichkeitssuche)
short/Integer SearchSimilarityExchange	Anzahl der Zeichen, die ersetzt werden sollen (Ähnlichkeitssuche)

Eigenschaften des Services com.sun.star.util.XReplaceDescriptor	
String ReplaceString	Die Zeichenkette die als Ersetzung dienen soll

9.2 Die Interfaces XSearchable und XReplaceable

Ein Search- bzw. ReplaceDescriptor kann auf alle Objekte angewandt werden, die die Interfaces com.sun.star.util.XReplaceable bzw. com.sun.star.util.XSearchable exportieren. Das sind insbesondere Zellen, Zellbereiche und Tabellenblätter, welche demnach folgende Methoden aufweisen:

Methoden des Interfaces com.sun.star.util.XSearchable	
createSearchDescriptor()	erzeugt einen SearchDescriptor
findAll(oSearchDescription)	durchsucht den enthaltenen Text nach allen Entsprechungen des spezifizierten Suchanforderung
findFirst(oSearchDescription)	durchsucht den enthaltenen Text nach der ersten Entsprechungen des spezifizierten Suchanforderung
findNext(nPosition, oSearchDescription)	durchsucht den enthaltenen Text nach der nächsten Entsprechungen des spezifizierten Suchanforderung

```
Sub textInZellebereichSuchen
    ' Variablen deklarieren
    Dim oZelleOderBereichOderBlatt as Object
    Dim oSuchBeschreibung as Object
    Dim oTrefferZelle as Object

    ' Zellbereich holen in dem ersetzt werden soll
    oZelleOderBereichOderBlatt = _
        ThisComponent.Sheets(0).getCellRangeByName( "A1:D40" )
```

```

' SearchDescriptor erzeugen
oSuchBeschreibung = oZelleOderBereichOderBlatt._
    createSearchDescriptor()

' SearchDescriptor konfigurieren
With oSuchBeschreibung
    .SearchString = "hallo"
    .SearchBackwards = False
    .SearchCaseSensitive = True
    .SearchWords = True
    .SearchRegularExpression = False
    .SearchStyles = False
    .SearchSimilarity = False
    .SearchSimilarityRelax = True
    .SearchSimilarityRemove = 2
    .SearchSimilarityAdd = 2
    .SearchSimilarityExchange = 2
End With

' nach erster Trefferzelle suchen
oTrefferZelle = oZelleOderBereichOderBlatt._
    findFirst( oSuchBeschreibung )

' Loop durch alle Trefferzellen
Do While Not IsNull ( oTrefferZelle )

    ' Treffer vermelden
    oTrefferZelle.CellBackColor = RGB( 200, 200, 200 )
    MsgBox "Treffer in Zelle " & _
        oTrefferZelle.CellAddress.Column & "/" & _
        oTrefferZelle.CellAddress.Row

    ' weitersuchen
    oTrefferZelle = oZelleOderBereichOderBlatt._
        findNext( oTrefferZelle , oSuchBeschreibung )
Loop
End Sub

```

Methoden des Interfaces com.sun.star.util.XReplaceable	
createReplaceDescriptor()	Erzeugt einen ReplaceDescriptor
replaceAll(SearchDescription)	Ersetzt allen Text, der der Suchanforderung entspricht

```

Sub SuchenUndErsetzen
    ' Variablen deklarieren
    Dim oZelleOderBereichOderBlatt as Object
    Dim oErsetzungsBeschreibung as Object

    ' Zellbereich holen indem ersetzt werden soll
    oZelleOderBereichOderBlatt = _
        ThisComponent.Sheets(0).getCellRangeByName( "A1:B10" )

    ' ReplaceDescriptor erzeugen
    oErsetzungsBeschreibung = _

```

```
oZelleOderBereichOderBlatt.createReplaceDescriptor()
```

```
' ReplaceDescriptor konfigurieren
oErsetzungsBeschreibung.SearchString = "hallo"
oErsetzungsBeschreibung.ReplaceString = "adieux"
oErsetzungsBeschreibung.SearchBackwards = False
oErsetzungsBeschreibung.SearchCaseSensitive = True
oErsetzungsBeschreibung.SearchWords = True
oErsetzungsBeschreibung.SearchRegularExpression = False
oErsetzungsBeschreibung.SearchStyles = False
oErsetzungsBeschreibung.SearchSimilarity = False
oErsetzungsBeschreibung.SearchSimilarityRelax = True
oErsetzungsBeschreibung.SearchSimilarityRemove = 2
oErsetzungsBeschreibung.SearchSimilarityAdd = 2
oErsetzungsBeschreibung.SearchSimilarityExchange = 2

' Ersetzung im Zellbereich durchführen
oZelleOderBereichOderBlatt.replaceAll( oErsetzungsBeschreibung )
```

```
End Sub
```

10 Calc-Funktionen

10.1 Aufruf von eingebauten Calc-Funktionen

Die Funktionen, welche in Calc zur Verfügung stehen, können auch über Ooo-Basic genutzt werden. Zum einen besteht die Möglichkeit, eine Funktion aufzurufen, die Argumente der Funktion in einem Array mitzugeben und das Resultat der ausgeführten Funktion z.B. in eine Basic-Variable zu schreiben oder als Wert zurückzugeben:

Das folgende Beispiel ruft die SUM() Funktion auf (immer englisch als Standard-Sprache verwenden). Die numerischen Summanden werden in einem Array mitgegeben.

```
Function funktionAufrufen ( value as Double ) As Double
' Variablen deklarieren
Dim oFunktion as Object

' com.sun.star.sheet.FunctionAccess holen
oFunktion = createUnoService("com.sun.star.sheet.FunctionAccess")

' als Argumente ein Array mit numerischen Werten.
' Sinnloserweise wird als Summanden einfach zweimal
' das Argument value verwendet
Dim aArgumente(1) As Variant
aArgumente(0) = value
aArgumente(1) = value

' Aufruf der SUM-Funktion mit Übergabe des Arrays
' mit den Werten die zu addieren sind
funktionAufrufen = oFunktion.callFunction( "SUM", aArgumente() )
End Function
```

Die Argumente einer Funktion werden in einen Array gelegt und beim Aufruf der Methode oFunktion.callFunction(sName, aArgumentArray()) als zweiter Parameter mitgegeben. Die Argumente, die in einem Array an die Funktion

übergeben werden, können folgende Datentypen aufweisen:

Datentyp	Funktions-Argumente
Long oder Double	für einen numerischen Wert
String	für eine Zeichenkette
Long[]» der Double[]	ein 2D-Array mit numerischen Werten
String[]» ein 2D-Array mit Zeichenketten	
Any[]	ein 2D-Array mit gemischten Datentypen

10.2 Eigene Calc-Funktionen in Basic

In Calc-Zellen lassen sich praktische Funktionen einfügen, welche Berechnungen mit Zellwerten anstellen und das Ergebnis als Wert in die Zelle schreiben. Wem die verfügbaren Funktionen nicht genug sind, kann seine eigenen Funktionen in OpenOffice.org-Basic schreiben und sie in der bekannten Form =meinefunktion(arg1 ; agr2) in einer Zelle verwenden.

Grundsätzlich können Argumente einer Funktion in einer Calc-Zelle drei Ausprägungen haben:

Argumententyp	Beschreibung	Werte in Basic
=meineFunktion("hallo ooo",23.55)	eingetippte Zeichenketten und numerische Werte	Zeichenkette oder numerischer Wert
=meineFunktion(A1)	Referenzen auf einzelne Zellen	Zeichenkette oder numerischer Wert
=meineFunktion(A1:B10)	Referenzen auf Zellbereiche	2D-Array

Im folgenden Beispiel wird die Übergabe des Argumentes vArgument der Funktion Argument in der Zelle an die Basic-Funktion Argument demonstriert. Ein Argument wird wie in der Tabelle aufgeführt immer als numerischer Wert, Zeichenkette oder zweidimensionaler Array übergeben.

```
' argument muss as Variant sein !!
Function Argument ( vArgument as Variant )
    ' Variablen deklarieren
    Dim sRueckgabeWert As String

    if IsMissing(vArgument) then
        sRueckgabeWert = "Argument fehlt"

    ElseIf IsArray(vArgument) then
        sRueckgabeWert = "Ein Array mit " & UBound(vArgument, 1) & _
            " Zeilen und " & UBound(vArgument, 2) & " Spalten"

    ElseIf IsNumeric( vArgument ) then
        sRueckgabeWert = "Numerisch: " & vArgument
    else
        sRueckgabeWert = "Zeichenkette: " & vArgument
    End if
    Argument = sRueckgabeWert
End Function
```

Wird ein Argument eingetippt oder handelt es sich um eine Referenz auf eine Zelle, so wird der Wert nach meinen Versuchen immer in eine Zeichenkette oder einen numerischen Wert transformiert. Zellinhalte wie WAHR und FALSCH oder Zellen mit Datumsformat werden als numerische Werte repräsentiert und nicht mit den Datentypen Boolean bzw. einem Datums-Objekt übergeben:

Wert in der Calc-Zelle	Typ der Basic-Variable	Wert
Text	String	z.B. ?Elefanten sind grau?
Numerisch	Numerisch	z.B. 2234 oder 0.222
Datum	Numerisch	31.12.1899-> 1 1.1.1900 -> 2 1.1.1970 -> 25569 2.1.1970 -> 25570
FALSCH/WAHR	Numerisch	0 oder 1
Leere Zelle	Numerisch	0

Folgendes Code-Beispiel demonstriert die Konvertierung eines Long-Wertes in ein Date-Objekt:

```
Sub ZumDatumKonvertieren ( nDatum as Long )
    ' Variable deklarieren
    Dim dDatum as Date
    ' Datum aus Long-Variable konvertieren
    dDatum = CDate( nDatum )
    ' Ausgabe an Benutzer
    MsgBox dDatum
End Sub
```

Als Konsequenz dieses pass-by-value ergibt sich, dass in Calc-Funktionen, die in Basic implementiert werden, keine Möglichkeit besteht, auf das Zellobjekt zuzugreifen. Der Wert wird ohne Information auf seine Quelle übergeben, womit keine Manipulation der Zelleigenschaften (Hintergrund, Textformatierung, etc.) möglich sind. Der Rückgabewert der Basic-Funktion (der in die Zelle eingetragen wird) ist die einzige Möglichkeit auf die Zelle Einfluss zu nehmen, in der die Funktion aufgerufen wurde.

Ein möglicher Workaround wäre natürlich die Zell- oder Zellbereichsadresse als Zeichenkette mitzugeben =meineFunktion("A1:B10"). So lässt sich der Zellbereich ermitteln und auf seine Eigenschaften zugreifen. Einen impliziten Mechanismus für einen solchen Zugriff existiert jedoch in Basic nicht, während dies meines Wissens mittels eines Calc-Addins in Java oder C++ realisierbar ist.

Das folgende Code-Beispiel stellt ein funktionales Äquivalent der =SUMME()-Funktion dar, die als Basic-Funktion implementiert ist. Sie demonstriert insbesondere die Übergabe von Werten eines ganzen Zellbereiches (z.B. A1:G10), die als zweidimensionaler Array (Matrix) übergeben werden.

```
Function summenTest ( vArgument as Variant )
    ' ist das argument ein Array (Zellbereich)
    if Not IsArray(vArgument) then
        ' ist der einzelne Wert numerisch
        if IsNumeric( vArgument ) then
            ' Rückgabewert ist gleich Eingabewert
            summenTest = vArgument
        else
            ' Summe ist gleich 0 wenn kein numerischer Wert
            ' übergeben wurde
            summenTest = 0
        end if
    end if
End Function
```

```

        end if
        ' funktion beenden
        exit function
End If

Dim nSumme as Long
Dim i as Integer
Dim j as Integer
' der spätere Rückgabewert
nSumme = 0
' Schleife durch 1 Dimension (Zeilen)
for i=1 to UBound(vArgument,1)
    ' Schleife durch zweite Dimension (Spalten)
    for j=1 to UBound(vArgument,2)
        ' numerisch?
        if IsNumeric( vArgument(i,j) ) then
            ' Wert dazuzählen
            nSumme = nSumme + vArgument(i,j)
        end if
    next j
next i
' Rückgabewert in Zelle
summenTest = nSumme
End Function

```

11 Formulare in Calc-Dokumenten

In Bezug auf Formulare weisen Calc-Dokumente gegenüber den anderen OpenOffice.org-Dokumenten eine Besonderheit auf. Formular befinden sich auf der DrawPage eines Office-Dokumentes. Während in Dokumenten von Writer, Impress und Draw nur eine einzige DrawPage existiert, verfügt ein Calc-Dokument über eine DrawPage für jedes Tabellenblatt. Die Formulare und deren Elemente sind in Calc-Dokumenten somit an ein Tabellenblatt und dessen DrawPage gebunden.

```

Sub FormulareInCalc
    ' Variablen deklarieren
    Dim oTabellenblatt as Object
    Dim nCounter as Integer
    Dim nCounter2 as Integer

    ' Schleife durch alle Tabellenblätter
    For nCounter=0 To ThisComponent.Sheets.Count-1

        ' Tabellenblatt holen
        oTabellenblatt = ThisComponent.Sheets(nCounter)

        If oTabellenblatt.DrawPage.Forms.Count > 0 Then
            ' Schleife durch alle Formulare des Tabelleblattes
            For nCounter2=0 To oTabellenblatt.DrawPage.Forms.Count-1
                ' gefundenes Formular melden
                MsgBox "Formular " & oTabellenblatt._
                    DrawPage.Forms(nCounter2).Name & _
                    " in Tabellenblatt " & (nCounter+1)
            Next nCounter2
        End If
    Next nCounter
End Sub

```

```
Else
    ' kein Formular in diesem Tabellenblatt
    MsgBox "kein Formular im Tabellenblatt " & (nCounter+1)
End If
Next nCounter
End Sub
```