

OFFICE TALK

Skript

Version 5.3

Stand der Dokumentation

Ersteller:	JOOPS Informationstechnik GmbH
Autoren	Manuela Springer, Josef Springer
Geprüft	3.11.2016
Revision	4.11.2016, 7.10.2016, 03.06.2016, 21.5.2014, 2.10.2013, 25.7.2013, 5.7.2013, 20.5.2013, 15.5.2013, 2.5.2013, 7.3.2013, 8.2.2013, 3.1.2013, 29.9.2012, 20.9.2012, 23.7.2012, 11.7.2012, 30.5.2012, 22.5.2012, 10.5.2012, 26.4.2012, 24.4.2012, 19.4.2012, 28.3.2012, 20.2.2012, 3.2.2012, 30.1.2011, 8.12.2011, 28.11.2011, 18.11.2011, 3.11.2011, 24.10.2011, 11.10.2011, 5.10.2011, 28.9.2011, 21.9.2011, 12.9.2011, 7.9.2011, 2.9.2011, 3.8.2011, 26.7.2011, 21.7.2011, 15.9.2011, 12.9.2011, 7.9.2011, 2.9.2011, 3.8.2011, 26.7.2011, 21.7.2011, 4.7.2011, 9.6.2011, 6.6.2011, 29.5.2011, 10.5.2011, 4.5.2011, 19.4.2011, 1.3.2011, 1.2.2011, 1.1.2011, 15.10.2010, 19.7.2010, 1.6.2010, 1.5.2010, 1.4.2010, 1.3.2010, 25.1.2010, 1.3.2010, 18.11.2009, 15.10.2009, 21.8.2009, 11.7.2009, 11.5.2009, 20.4.2009, 18.3.2009, 21.2.2009, 18.11.2008, 10.8.2008, 3.5.2008, 25.3.2008, 31.1.2008, 11.12.2007, 16.10.2007, 5.9.2007, 18.6.2007, 17.5.2007, 27.4.2007, 26.3.2007, 8.1.2007
Änderungsverweis	Siehe Datei changes.html
Version	5.30
Freigabe	5.11.2016

Inhaltsverzeichnis

Einleitung.....	1
Allgemeine Syntax.....	1
Skript.....	1
Makro.....	1
Skriptmakro Syntax.....	1
Aufbau eines Makros.....	2
Makroname.....	2
Syntax.....	2
Beispiel.....	2
Bemerkung.....	2
Direktive.....	2
Syntax.....	2
Library.....	3
Syntax.....	3
Beispiel.....	3
Interface.....	3
Syntax.....	3
Beispiel.....	3
Bemerkung.....	4
Assembly.....	4
Syntax.....	4
Beispiel.....	4
Variablendeklaration.....	4
VisualBasic Datentypen.....	5
Globale Variable.....	10
Lokale Variable.....	11
Stringsyntax von Variablen.....	12
Stringsyntax für Integer/Long/Float/Double.....	12
Stringsyntax für Date.....	12
Stringsyntax für Time.....	12
Stringsyntax für Boolean.....	13
Anweisung.....	13
Syntax.....	13
Systemvariable.....	13
Die Namen der Systemvariablen.....	13
Globale Systemvariable-Typen.....	14
Globale Systemvariablen.....	15
Syntax der globalen Systemvariablendienste.....	15
action.....	15
Syntax.....	15
ActionHistory.....	17
Syntax.....	17
Error.....	18
Syntax.....	18
Beispiel.....	18
FileSystemObject.....	19
Syntax.....	19
Beispiel.....	25
FileDialog.....	25
Syntax.....	25
Beispiel.....	26
HTTPClient.....	27
Syntax.....	27

Beispiel 1	32
Beispiel 2	32
Beispiel 3	32
Mail	34
Syntax	34
Besonderheiten	39
Beispiel	39
process	41
Syntax	41
processdata	48
Syntax	48
Beispiel-1	50
Beispiel-2	51
Beispiel-3	51
ProcessHistory	52
Syntax	52
Beispiel	52
Question	54
Syntax	54
Beispiel	54
Resource	55
Syntax	55
scheduledata	56
Syntax	56
SystemData	56
Syntax	56
Beispiel	56
ScriptDialog	57
Syntax	57
Beispiel eines nicht modalen Dialoges	114
Beispiel eines Dialoges mit Aktionen zum Start	116
step	118
Syntax	118
Beispiel	120
StepHistory	122
Syntax	122
stepscheduler	124
Syntax	124
Beispiel-1	134
Beispiel-2	134
Beispiel-3	134
Beispiel-4	135
Beispiel-5	135
Beispiel-6	136
worker	137
Syntax	137
WSDLClient	141
Syntax	141
Beispiel	143
WSDLStruct	144
Syntax	144
Besonderheiten	144
Beispiel	144
Dynamisch geladene Variablentypen	146
Kommunikationsbibliothek	146
Automatisation	146
Anweisungen	147
= (Zuweisung)	147

Syntax.....	147
Besonderheiten	147
Beispiel	148
Close	148
Syntax.....	148
Beispiel	148
Do Until.....	148
Syntax.....	148
Beispiel	148
Do While.....	149
Syntax.....	149
Beispiel	149
Exit Do	149
Syntax.....	149
Beispiel	149
Exit For	150
Syntax.....	150
Beispiel	150
For Next	150
Syntax.....	150
Beispiel	150
Halt.....	151
Syntax.....	151
Beispiel	151
If	151
Syntax.....	151
Beispiel	151
Input.....	152
Syntax.....	152
Beispiel	152
Line Input	152
Syntax.....	152
Beispiel	152
New.....	153
Syntax.....	153
Besonderheiten	153
Beispiel	153
Open	153
Syntax.....	153
Beispiel	154
Print	154
Syntax.....	154
Besonderheiten	154
Beispiel	154
Return	155
Syntax.....	155
Beispiel	155
Select Case.....	155
Syntax.....	155
Beispiel	156
Try-Catch.....	156
Syntax.....	156
Beispiel	156
While	157
Syntax.....	157
Besonderheiten	157
Beispiel	157
Write.....	157

Syntax.....	157
Beispiel	157
Funktionen	158
Array	158
Syntax.....	158
Beispiel	158
Zugriffe auf Arrayelemente	158
Syntax.....	158
Beispiel	158
Bemerkung	158
Call-Dienst	159
Behandlung fehlerhafter Argumente	159
Verletzung von Ausführungsbedingungen.....	159
Syntax.....	159
Beispiel	160
Bemerkung	160
Call-Makro	160
Syntax.....	160
Beispiel	160
Bemerkung	160
Call-Supermakro	161
Syntax.....	161
Beispiel	161
ChrB	161
Syntax.....	161
Beispiel	161
Chr	161
Syntax.....	161
Beispiel	162
CurDir	162
Syntax.....	162
Beispiel	162
Date	162
Syntax.....	162
Beispiel	162
Delay	162
Syntax.....	162
Beispiel	162
Bemerkung	162
Enum.....	163
Syntax.....	163
Beispiel	163
Filter.....	163
Syntax.....	163
InputBox	164
Syntax.....	164
Beispiel	164
IsDate	164
Syntax.....	164
Beispiel	164
IsEmpty	165
Syntax.....	165
Beispiel	165
IsNull.....	165
Syntax.....	165
Beispiel	165
IsNumeric	165
Syntax.....	165

Beispiel	165
InStr	166
Syntax.....	166
Ergebnis.....	166
Beispiel	166
InStrRev	166
Syntax.....	166
Join	167
Syntax.....	167
Beispiel	167
LCase.....	167
Syntax.....	167
Left	168
Syntax.....	168
Beispiel	168
Len.....	168
Syntax.....	168
Beispiel	168
LTrim	168
Syntax.....	168
Mid.....	169
Syntax.....	169
Beispiel	169
MousePointer	169
Syntax.....	169
Beispiel	169
MsgBox.....	170
Syntax.....	170
Beispiel	171
Refresh	171
Syntax.....	171
Beispiel	171
Replace	171
Syntax.....	172
Right.....	172
Syntax.....	172
Beispiel	172
RTrim	172
Syntax.....	173
Shell.....	173
Syntax.....	173
Beispiel	173
Bemerkung.....	174
Shell Wait	174
Syntax.....	174
Beispiel	174
Space	175
Syntax.....	175
Split	175
Syntax.....	175
Start	175
Syntax.....	176
Beispiel	176
Bemerkung.....	176
StrComp	177
Syntax.....	177
Ergebnis.....	177
Beispiel	177

StrConv	177
Syntax.....	177
String.....	178
Syntax.....	178
StrReverse.....	178
Syntax.....	178
Time.....	178
Syntax.....	178
Beispiel	178
UBound	179
Syntax.....	179
Beispiel	179
Bemerkung.....	179
UCase	179
Syntax.....	179
Die Makrosyntax im Überblick.....	180
Ausdrücke.....	180
Kaskadierung	180
Bearbeitungsreihenfolge	180
Binäre Ausdrücke	181
Boolsche Operatoren	181
Operator Is	181
Arithmetische Operatoren	181
Beispiel	181
Operator &	181
Linker Operand ist Array.....	181
Linker Operand ist kein Array	182
Beispiel	182
Verbinden mehrerer Zeichenketten	182
Beispiel	182
Hinweise für die Verwendung von Variablen	182
Die Syntax des Makros.....	183
Die Syntax des Ausdrucks.....	183

Einleitung

Diese Dokumentation beschreibt die Syntax von Skripts und Skriptmakros. Skripts sind die flexible Art, OfficeTalk an die variablen Bedürfnisse eines Unternehmens anzupassen. Wie Sie Skripts und Skriptmakros erstellen, ändern und löschen, lesen Sie in der Dokumentation Business-Process-Management, Kapitel *Register Skripts*.

Allgemeine Syntax

In dem Kapitel wird der Aufbau und die allgemeine Syntax des Skripts und des Skriptmakros beschrieben. Den größeren Teil dieser Dokumentation belegt das Kapitel *Skriptmakro Syntax*.

Skript

Das Skript besteht aus einem Namen und einem oder mehreren Makros (auch Methoden genannt), die in Aktionen verwendet werden. Ein Skript kann maximal ein Makro mit der Eigenschaft **Startmakro** enthalten. Bei der Ausführung eines Skripts durch eine Aktion wird das Skriptmakro mit dieser Eigenschaft ausgeführt. Das Skript wird im Bearbeiter abgelegt. Das Skript wird im Bearbeiterdialog im Register Skript erstellt.

Makro

Ein Makro besteht aus einem Namen, mit dem es auch gestartet wird. Die erste Zeile des Makrotextes ist zugleich auch sein Name. Es enthält Anweisungen, die ausgeführt werden. Die Syntax ist weitgehend an Microsoft® VisualBasic angelehnt. In einem Skriptmakro kann auf den aktuellen Bearbeiter, den laufenden Vorgang sowie den daraus aktuellen Arbeitsschritt und die Aktion zugegriffen werden. Jedes Makro ist einem Skript zugeordnet, und wird durch den Skriptnamen in Verbindung mit seinem Makronamen (`<Skriptname>.<Macroname>`) eindeutig identifiziert. Es wird im Bearbeiterdialog im Register Skript erstellt.

Skriptmakro Syntax

In diesem Kapitel wird die Syntax des Skriptmakros beschrieben. Die Syntax entspricht weitgehend der Syntax von Microsoft® VisualBasic. Nachfolgend sind Anweisungen oder Anweisungsteile im Gegensatz zu den übrigen Texten in der Schriftart *Courier* geschrieben. Die Schreibweise der folgenden Syntax:

- `<` und `>` umschließen Begriffe, die durch den aktuellen Namen, Bezeichner oder ähnlichen zu ersetzen sind und werden, falls nicht anders vermerkt, nicht geschrieben.
- `[` und `]` umschließen optionale Angaben und werden nicht geschrieben.
- `...` kennzeichnen weitere gleichartige Angaben und werden nicht geschrieben.
- Mehrere Angaben durch `|` getrennt sind alternativ und mit `{ }` umschlossen.
- Wörter ohne obige Kennzeichnung sind unverändert zu übernehmen.

Aufbau eines Makros

```
<Makroname>[(Argument[, ...])  
[<Direktive...>]  
[<Variablendeklaration...>]  
[<Anweisung...>]
```

Makroname

Der Makroname benennt das Makro. Mit diesem Namen kann es auch gestartet werden. Dem Namen können optional Argumente folgen.

Syntax

```
<Makroname>[(Argument[, ...])  
Makroname  
Alphanumerischer Name bis maximal 64 Zeichen  
Argument  
<Variable> As <Typ>  
Variable  
Alphanumerischer Name bis maximal 64 Zeichen  
Typ  
{String|SQLString|Date|Time|Array|ByteString|Character|Integer|Long|  
Float|Double|Boolean|Worker|Desk|Machine|Department|Team|Company|  
Office|Process|ProcessHistory|Processdata|Resource|Question|  
Scheduledata|Step|StepHistory|Action|ScriptDialog|Object}
```

Beispiel

```
Berechnen(kosten As Double, faktor As Integer)
```

Bemerkung

Die Argumente eines Makros sind aus Gründen der Schnittstellensicherheit nicht änderbar (readOnly). Der Compiler bricht die Übersetzung mit einer entsprechenden Fehlermeldung ab, wenn auf ein Argument in ändernder Weise zugegriffen wird. Als einzige Ausnahme davon gelten Argumente vom Type Array. Auf die Elemente eines Arguments vom Typ Array darf in ändernder Weise zugegriffen werden. Ein kleines Beispiel soll das verdeutlichen:

```
WegstreckeErmitteln( straßen As Array, ort As String )  
.....  
straßen( 1 ) = "Orlando-di-Lasso Str. 2"     'zulässig Zuweisung  
ort = "Putzbrunn"                             'unzulässige Zuweisung  
...
```

Direktive

Mit einer Direktive werden für ein Makro allgemeingültige Vorgaben gemacht

Syntax

```
<Direktivename> <Direktiveargument>
```

Library

Die Direktive lädt Schnittstellen (Bibliotheken) für die Kommunikation mit externen Systemen. Diese Schnittstellen werden in Makros benötigt. Damit ein Makro mit Variablentypen externer Systeme arbeiten kann, müssen diese zuerst mit der Direktive `Library` geladen werden. Die Direktive wird vom Register **Bibliotheken** der Workbench verwaltet. (siehe auch Kapitel *Register Bibliotheken* in der Dokumentation [Business-Process-Management](#)).

Syntax

Library "<Dateiname>"

Der `Dateiname` der zu ladenden Bibliothek. Dies muss ein Dateiname mit der Namens-erweiterung `.pcl` sein. Die Pfadangabe kann relativ sein, wenn das System mit einem Startpfad gestartet wurde. Der Pfad kann einen symbolischen Namensteil enthalten (siehe Kapitel *Datei/URL* in der Dokumentation [Business-Process-Management](#)).

Beispiel

Um mit MS-Word zu arbeiten, müssen folgende Direktiven verwendet werden.

```
Library "..\Library\Microsoft Office.pcl"
```

```
Library "..\Library\Microsoft VBIDE.pcl"
```

```
Library "..\Library\Microsoft Word.pcl"
```

Oder alternativ, wenn Sie nur eine vorbereitete Vorlage ausfüllen und drucken wollen

```
Library "..\Library\Microsoft Word Small.pcl"
```

Interface

Die Direktive ist nur auf Windows-Plattformen verfügbar und lädt dynamisch die Schnittstellen (Interface) eines externen Systems für die Kommunikation. Das externe System muss dazu die Automatisierungsschnittstelle unterstützen. Ob ein System die Automatisierungsschnittstelle unterstützt und somit dynamisch eingebunden werden kann, erfragen sie vom Hersteller des Systems. Die Direktive wird vom Register **Interfaces** der Workbench verwaltet. (siehe auch Kapitel *Register Interfaces* in der Dokumentation [Business-Process-Management](#)).

Syntax

Interface "<Interfacename>"

`Interfacename` ist die Signatur der Typenbibliothek mit der sie in der Registry im Kapitel **HKEY_CLASSES_ROOT** registriert ist.

Beispiel

Ein kleines Beispiel für die dynamische Integration des Systems Microsoft-Excel:

```
Interface "Microsoft Excel 9.0 Object Library"
```

```
Dim excel as Excel.Application
```

```
excel = New Excel.Application
```

```
excel.Visible = True
```

```
excel.Workbooks.Add
```

```
... weitere Anweisungen ...  
excel.Quit
```

Bemerkung

Mit der Anweisung `Dim` und `New` kann nur eine **CoClass** der Automatisierungsschnittstelle verwendet werden. Bei allen weiteren Interfaces muss für Zuweisungen eine Variable vom Typ `Object` verwendet werden.

Assembly

Die Direktive ist nur auf Windows-Plattformen verfügbar und lädt dynamisch die Schnittstellen (Assembly) eines externen .NET-Systems für die Kommunikation. Das Assembly muss installiert sein, oder in Dateiform vorhanden sein. Die Direktive wird vom Register **Assemblies** der Workbench verwaltet. (siehe auch Kapitel *Register Assemblies* in der Dokumentation Business-Process-Management).

Syntax

Assembly "<Assemblyname>"

Das Assembly kann mit seinem Dateinamen oder mit seiner Signatur angegeben werden. (siehe auch Kapitel *Register Assemblies* in der Dokumentation Business-Process-Management).

Beispiel einer Signatur

```
"System.Printing, Version=3.0.0.0, Culture=neutral,  
PublicKeyToken=31bf3856ad364e35"
```

Beispiel eines Dateinamens

```
"C:\Windows\assembly\GAC_32\System.Printing\3.0.0.0__31bf3856ad364e35  
\System.Printing.dll"
```

Beispiel

Ein kleines Beispiel für die dynamische Integration der Druckdienste:

```
Assembly "System.Printing, Version=3.0.0.0, Culture=neutral,  
PublicKeyToken=31bf3856ad364e35"
```

```
Dim printer as System.Printing.PrinterServer
```

```
printer = New System.Printing.PrinterServer
```

```
MsgBox("Printrname: " & printer.getName)
```

Variablendeklaration

Mit der Variablendeklaration werden Variablen bekanntgegeben, um sie anschließend zu benutzen. Der Skripteditor bietet Ihnen bei Eingabe von `Dim`, gefolgt von zwei Leerzeichen, eine Liste der möglichen Variablentypen zur Auswahl an.

VisualBasic Datentypen

In der Variablendeklaration können neben den OfficeTalk spezifischen *Systemvariable* auch Datentypen der VisualBasic-Syntax verwendet werden.

Array

Der Datentyp Array wird für Listen mit Variablen verwendet. Ein Array kann selbst wiederum ein Array enthalten (mehrdimensionales Array).

Beispiel

```
Dim x As Array  
x = Array("erster", 10, worker, Array(process, "Ausführung"))
```

Dienste des Datentyps Array

Der Datentyp Array unterstützt, neben den *Zugriffe auf Arrayelemente*, zur einfacheren Verwaltung seines Inhalts, einige Dienste.

add (<Variable>, at: <Zahl>)

Der Dienst fügt <Variable> an der Position <Zahl> im Array, ein und liefert die eingefügte Variable.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Die einzufügende Variable.
2. Zahl
{Variabel|Literal|Ausdruck|Funktion}
Die Position für die Einfügung. Die Zählbasis ist 0.

addAll (<Variable>)

Der Dienst fügt <Variable> am Ende des Arrays, ein und liefert die eingefügte Array. <Variable> kann ein Array oder ein String sein. Wenn <Variable> ein String ist, wird jedes Zeichen des Strings als Arrayelement eingefügt.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Das einzufügende Array.

addFirst (<Variable>)

Der Dienst fügt <Variable> am Anfang des Arrays ein und liefert die eingefügte Variable.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Die einzufügende Variable.

addLast (<Variable>)

Der Dienst fügt <Variable> am Ende des Arrays, ein und liefert die eingefügte Variable.

2. Variable
{Variabel|Literal|Ausdruck|Funktion}
Die einzufügende Variable.

asAssociations

Der Dienst wandelt das Array als neues Array mit Associationen. Dazu muss das Array wiederum Arrays mit jeweils zwei Elementen enthalten. Diese Zwei-Element Arrays werden in Associationen gewandelt. Eine Association besteht aus einem Schlüssel und seinem Wert. Das erste Element des inneren Arrays wird der Schlüssel, und das zweite Element des inneren Arrays wird der korrespondierende Wert. Dieses Association-Array kann für die Dialogelemente Auswahlliste und Liste eines Vorgangsdialoges verwendet werden, um für eine Auswahl nicht dessen Wert, sondern dessen Schlüssel zu erhalten.

Beispiel

```
Dim pri As Integer
Dim name As String
dialog = ScriptDialog new

dialog.value("Priorität",
             put: Array(Array(1,Huber),Array(2,Meier),
                        Array(3,Müller)).asAssociations)

dialog.open                                'Anwender wählt den Eintrag Meier
pri = dialog.key("Priorität")              'pri enthält 2
name = dialog.value("Priorität")          'name enthält Meier
```

join

Der Dienst liefert, ähnlich wie die Funktion *Join*, den Inhalt des Arrays als einen String, eingeschlossen in die Zeichen (und). Die einzelnen Elemente des Arrays werden durch das Zeichen , getrennt.

join(<Variable>)

Der Dienst liefert, ähnlich wie die Funktion *Join*, den Inhalt des Arrays als einen String, eingeschlossen in die Zeichen (und). Die einzelnen Elemente des Arrays werden jedoch durch <Variable> getrennt.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Das Zeichen, um die Elemente des Arrays zu trennen. <Variable> muss ein Datum vom Typ Character oder String bezeichnen.

includes (<Variable>)

Der Dienst liefert die Variable aus dem Array, die mit der angegebenen <Variable> übereinstimmt. Der Vergleich wird mit dem Operator = durchgeführt.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Die gesuchte Variable.

identityIncludes (<Variable>)

Der Dienst liefert die Variable aus dem Array, die mit der angegebenen <Variable> übereinstimmt. Der Vergleich wird mit dem Operator *Is* durchgeführt.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Die gesuchte Variable.

identityRemove (<Variable>)

Der Dienst entfernt die Variable aus dem Array, die mit der angegebenen <Variable> übereinstimmt. Der Vergleich wird mit dem Operator *Is* durchgeführt. Das Ergebnis des Dienstes ist die entfernte Variable oder Null, wenn die Variable im Array nicht entfernt wurde.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Die zu entfernende Variable.

remove (<Variable>)

Der Dienst entfernt die Variable aus dem Array, die mit der angegebenen <Variable> übereinstimmt. Der Vergleich wird mit dem Operator *=* durchgeführt. Das Ergebnis des Dienstes ist die entfernte Variable oder Null, wenn die Variable im Array nicht entfernt wurde.

1. Variable
{Variabel|Literal|Ausdruck|Funktion}
Die zu entfernende Variable.

removeAt(<Zahl>)

Der Dienst entfernt die Variable an der mit <Zahl> angegebenen Position und liefert die entfernte Variable

1. Zahl
{Variabel|Literal|Ausdruck|Funktion}
Die Position der zu entfernenden Variablen. Die Zählbasis ist 0.

removeFirst

Der Dienst entfernt die erste Variable in dem Array und liefert die entfernte Variable.

removeLast

Der Dienst entfernt die letzte Variable in dem Array und liefert die entfernte Variable.

reverse

Der Dienst liefert ein neues Array, mit der umgedrehten Reihenfolge der Variablen.

Boolean

Der Datentyp Boolean wird für die Literale True und False Verwendet.

Beispiel

```
Dim x As Boolean
x = True
```

Character

Der Datentyp Character wird für einzelne Zeichen des Zeichensatzes verwendet.

Beispiel

```
Dim x As Character
x = ChrB(10) 'Ein Zeilenvorschub
```

Date

Der Datentyp `Date` wird für ein Tagesdatum verwendet.

Beispiel

```
Dim x As Date
x = Date           'Das aktuelle Tagesdatum
```

Double

Der Datentyp `Double` wird für Zahlen mit Bruch im Bereich $4.9406564584125 \times 10^{-324}$ bis $1.7976931348623 \times 10^{308}$ verwendet. Das Zeichen `.` ist der Dezimaltrenner. Das Zeichen `,` ist der Tausendertrenner.

Beispiel

```
Dim x As Double
x = 123,456.12
```

File

Der Datentyp `File` ist nicht Bestandteil der VisualBasic-Datentypen, sondern eine Office-Talk-Erweiterung. Mit dem Datentyp können Dateien gelesen und geschrieben werden. Einzelheiten zu und diesen Aufgaben finden Sie in den Kapiteln *Close*, *Input*, *Line Input*, *Open*, *Print* und *Write*.

Beispiel

```
Dim x As File
x = Open "c:\temp\josef.txt" For Output
...
Close x
```

Float

Der Datentyp `Float` wird für Zahlen mit Bruch im Bereich 1.4013×10^{-45} bis 3.40282×10^{38} verwendet. Das Zeichen `.` ist der Dezimaltrenner. Das Zeichen `,` ist der Tausendertrenner.

Beispiel

```
Dim x As Float
x = 123,456.12
```

Integer

Der Datentyp `Integer` wird für ganze Zahlen im Bereich $2^{29} - 1$ (536,870,911) bis -2^{29} verwendet.

Beispiel

```
Dim x As Integer
x = 1234
```


Long

Der Datentyp `Long` wird für ganze Zahlen im Bereich $2^{29} - 1$ (536,870,911) bis -2^{29} verwendet.

Beispiel

```
Dim x As Long
x = 123456
```

String

Der Datentyp `String` wird für Zeichenfolgen in `"` und `'` eingeschlossen, verwendet.

Beispiel

```
Dim x As String
x = "das ist mein name"
```

Dienste des Datentyps String

Der Datentyp `String` unterstützt den Spezialdienst `evaluate`. Mit diesem Dienst können Dienste des zugrunde liegenden Basissystems ausgeführt werden. Der `String` muss ein Ausdruck in Smalltalk-Syntax sein. Im Ausdruck können Argumentvariable (siehe Kapitel *Makroname*), Variable (siehe Kapitel *VisualBasic Datentypen*) und globale Systemvariable (siehe Kapitel *Globale Systemvariablen*) verwendet werden.

Hinweis: Inhaltsänderungen von Variablen durch den Smalltalkausdruck sind lokaler Natur und gelten nur innerhalb des Smalltalkausdruckes. Der Inhalt einer Variablen des Makros kann nur durch eine Ergebniszuzuweisung des Dienstes geändert werden (siehe Beispiel *Dienste des Basissystems verwenden*).

Beispiele

```
Dim smalltalk As String
Dim ergebnis As String
Dim zahl As Integer
```

'Dienste des Basissystems verwenden

```
smalltalk = "Locale current languageID"
ergebnis = smalltalk.evaluate
MsgBox("Systemländerkennung: " & ergebnis)
```

'Globale Variablen verwenden

```
smalltalk = "worker printName"
MsgBox("Der angemeldete Barbeiters heißt " & smalltalk.evaluate)
```

'Locale Variablen verwenden

```
zahl = 99
smalltalk = "zahl sqrt"
MsgBox("Wurzel aus " & zahl & " = " & smalltalk.evaluate)
```

SQLString

Der Datentyp `SQLString` ist nicht Bestandteil der VisualBasic-Datentypen, sondern eine OfficeTalk-Erweiterung, und entspricht dem Datentyp `String`. Jedoch kann der Inhalt der Variable mit dem Inhalt anderer Variablen gebildet werden. Besonders für SQL-Anweisungen ist dieser Datentyp sehr hilfreich, da damit die unübersichtliche Aneinanderreihung mehrerer Variablen (SQL-Anweisungsteile und Datenwerte) entfällt. Die im `SQLString` einzufügenden Variableninhalte werden durch den Variablennamen mit führendem Platzhalterzeichen `$` geschrieben.

Hinweis: Die Ersetzung der Variablenplatzhalter erfolgt nur, wenn die `SQLString`-Variable als Argument eines Dienste verwendet wird, oder wenn die Variable einer Variablen vom Typ `String` zugewiesen wird !

Natürlich ist dieser Ersetzungsmechanismus nicht auf SQL-Einsatzgebiete beschränkt. Der Datentyp `SQLString` kann auch in anderen Anwendungsbereichen eingesetzt werden.

Beispiel

```
Library "..\Library\OracleForOfficeTalk.pcl"

Dim session As SmallCOM.OracleForOfficeTalk.OfficeTalkSession
Dim befehl As SQLString
Dim arg1 As String
Dim arg2 As String

session = New SmallCOM.OracleForOfficeTalk.OfficeTalkSession
session.openDatabase("OFFICETA", connect: "OfficeTalk/OfficeTalk",
connectType: 0)
arg1 = "Müller"
arg2 = "M"
befehl = "Select from tabelle where name='$arg1' and ort like '$arg2%'"
session.createDynaset(befehl, queryOption: 0)
```

Bemerkung

Der Datentyp kann alternativ auch mit dem Namensraum `Joops.Scripting` verwendet werden. Z.B. `Dim zeile As Joops.Scripting.SQLString`

Time

Der Datentyp `Time` wird für eine Uhrzeit verwendet.

Beispiel

```
Dim x As Time
x = Time            'Die aktuelle Uhrzeit
```

Globale Variable

Globale Variable sind ab der Ausführung des Makros, das die Deklaration enthält, bis zum Ausführungsende der laufenden Aktion gültig. Globale Variable müssen vor den lokalen Variablen deklariert werden. Die erste `Public Dim`-Anweisung eines Makros ist zugleich auch die Definition der globalen Variablen. Weitere `Public Dim`-Anweisung im Aktions-

ablauf gelten nur als Deklaration der bereits definierten Variable. Um im Makro eine globale Variable zu verwenden, muss sie deklariert werden.

Hinweis: Globale Variablen mit Ressourcen, die eine spezielle Beendigung erfordern (z.B. parallele Vorgangsdialoge, geöffnete Dateien, usw.), müssen spätestens im Startmakro der Aktion freigegeben werden.

Syntax

```
Public Dim <Variable> As <Typ>
```

Variable

alphanumerischer Name bis maximal 64 Zeichen. Siehe dazu auch Abschnitt *Systemvariable*.

Typ

Mögliche Datentypen siehe Kapitel *Syntax*. Die Schreibweise der Namen ist nicht case sensitiv. Siehe dazu auch Kapitel *Systemvariable*. Weiter Syntaxregeln finden Sie im Kapitel *Lokale Variable*

Lokale Variable

Lokale Variable gelten nur innerhalb des Makros, in dem sie deklariert sind. Lokale Variable müssen nach globalen Variablen deklariert werden.

Syntax

```
Dim <Variable> As <Typ>
```

Variable

alphanumerischer Name bis maximal 64 Zeichen. Siehe dazu auch Abschnitt *Systemvariable*.

Typ

Mögliche Datentypen siehe Kapitel *Syntax*. Die Schreibweise der Namen ist nicht case sensitiv. Siehe dazu auch Kapitel *Systemvariable*. Variable vom Typ `Date`, `Time`, `Integer`, `Float`, `Double` und `Boolean` können auch über *Stringvariable* belegt werden. Jedoch müssen diese Stringvariablen der Stringsyntax des Datentyps entsprechen. Addition und Subtraktion mit `Date`-Datentypen wird als Addition und Subtraktion von Tagen interpretiert. Addition und Subtraktion mit `Time`-Datentypen wird als Addition und Subtraktion von Sekunden interpretiert. Jedoch sind `Date`- und `Time`-Datentypen in arithmetischen Ausdrücken nur als linker Operand erlaubt. Ein Beispiel:

```
Dim morgen As Date
```

```
Dim nächsteStunde As Time
```

```
morgen = Date + 1
```

```
nächsteStunde = Time + (60 * 60)
```

```
MsgBox(Join(Array("Morgen ist der:", morgen, "nächste Stunde ist: ",  
nächsteStunde), " "))
```

Bemerkung

In der nachfolgenden Beschreibung wird für den Variablentyp `String` auch der Ausdruck **Zeichenkette** verwendet. Für den Variablentyp `Character` wird auch der Ausdruck **Zeichen** verwendet.

Beispiel

```
Dim zahl As Integer
```

```
zahl = 123.45 * 56
```

Stringsyntax von Variablen

Variable vom Typ Date, Time, Integer, Float, Double und Boolean können auch mit Zeichenketten (String) erstellt werden. Jedoch müssen diese Zeichenketten der Syntax zur Erzeugung aus einer Zeichenkette, der Stringsyntax entsprechen.

Stringsyntax für Integer/Long/Float/Double

[+ | -]Ziffer[.]... [,]<Ziffer>...

, ist der Dezimalpunkt und . ist der Tausendertrenner.

Beispiel

```
Dim variable As Integer
Dim variable2 As Double
variable = "200"
variable = "-350"
variable2 = "456,99"
variable2 = "299.455,66"
```

Stringsyntax für Date

TT.MM.[JJ]JJ

TT: 1 – 32 (Der Tag als ein- oder zweistellige Zahl)

MM: 1 - 12 | Januar – Dezember

[JJ]JJ: Das Jahr mit oder ohne Jahrhundert

Beispiel

```
Dim variable As Date
variable = "1.1.1980"
variable = "12.02.07"
variable = "1. Januar 2008"
variable = "1. Januar 08"
```

Stringsyntax für Time

HH:MM[:SS]

HH: 1 – 24 (Die Stunden als ein- oder zweistellige Zahl)

MM: 1 – 60 (Die Minuten als ein- oder zweistellige Zahl)

SS: 1 – 60 (Die Sekunden als ein- oder zweistellige Zahl)

Beispiel

```
Dim variable As Time
variable = "10:10:10"
variable = "23:10:12"
variable = "11:2"
```

Stringsyntax für Boolean

#TRUE# | True | #FALSE# | False

Beispiel

```
Dim variable As Boolean
variable = "#true#"
variable = "#TRUE#"
variable = "false"
variable = "True"
variable = "FALSE"
```

Anweisung

Hier werden die auszuführenden Anweisungen geschrieben. Die Syntax ist von der jeweiligen Anweisung oder Funktion abhängig.

Syntax

```
<Anweisung, ...>
{ Anweisungen | Funktionen }
```

Systemvariable

OfficeTalk-Scripting stellt Systemvariablen zur Verfügung. Systemvariablen können Sie voll qualifiziert mit dem vorangestellten Namensraum (z.B. `Joops.OfficeTalk.Desk`) und teilqualifiziert ohne Namensraum (z.B. `Desk`) angeben. Die Schreibweise der Datentypen ist nicht case sensitiv. Datentypen aus dem Namensraum `SmallCOM` müssen Sie immer voll qualifiziert schreiben. Wenn Sie einen unbekannten Variablentyp eingetippt haben, erhalten Sie beim Formatieren oder Kompilieren des Makros eine Auswahlliste mit den gültigen Typen. Daraus können Sie durch Auswahl den Variablentyp korrigieren. Es gibt zwei Arten von Systemvariablen:

- Der Variablentyp ist bekannt, aber die Variable muss zuerst mit der `Dim`-Anweisung deklariert werden, um benutzt werden zu können.
- Der Variablentyp und der Variablenname ist bekannt. D.h. Die Variable ist automatisch deklariert, die `Dim` Anweisung entfällt. Diese Variablen werden **globale Systemvariable** genannt.

Die Namen der Systemvariablen

Da das zugrunde liegenden Basissystem und OfficeTalk gleich benannte Klassen enthält, ist es empfehlenswert, um Mehrdeutigkeiten zu vermeiden, Klassen aus den Namensräumen OfficeTalk und Skripting unter Nennung ihres vollständigen Namens zu verwenden.

Klassen aus dem Namensbereich	vollständiger Name inkl. Namensraum
OfficeTalk	Joops.OfficeTalk (z.B.: Joops.OfficeTalk.Worker)
Skripting	Joops.Scripting (z.B.: Joops.Scripting.SQLString)

Globale Systemvariable-Typen

Folgende Typen von Variablen sind in Makros bekannt:

- **Action** oder **Joops.OfficeTalk.Action**
Datentyp einer Aktion (siehe Kapitel *action*)
- **ActionHistory** oder **Joops.OfficeTalk.ActionHistory**
Datentyp der Aktionshistorie (siehe Kapitel *ActionHistory*)
- **Company** oder **Joops.OfficeTalk.Company**
der Bearbeiter als Firma (siehe Kapitel *worker*)
- **Department** oder **Joops.OfficeTalk.Department**
der Bearbeiter als Abteilung (siehe Kapitel *worker*)
- **Desk** oder **Joops.OfficeTalk.Desk**
der Bearbeiter Schreibtisch (siehe Kapitel *worker*)
- **Machine** oder **Joops.OfficeTalk.Machine**
eine Maschine als der Bearbeiter (siehe Kapitel *worker*)
- **Mail** oder **Joops.Scripting.Mail**
Datentyp zum Versenden von eMails (siehe Kapitel *Mail*)
- **Office** oder **Joops.OfficeTalk.Office**
der Bearbeiter als Büro (siehe Kapitel *worker*)
- **Process** oder **Joops.OfficeTalk.Process**
Datentyp des Vorgangs (siehe Kapitel *process*)
- **Processdata** oder **Joops.OfficeTalk.ProcessData**
Datentyp der permanenten Vorgangsdaten (siehe Kapitel *processdata*)
- **ProcessHistory** oder **Joops.OfficeTalk.ProcessHistory**
Datentyp der Vorgangshistorie (siehe Kapitel *Beispiel-3*)
- **Question** oder **Joops.OfficeTalk.Question**
Datentyp der Fragen und Antworten für Vorgänge (siehe Kapitel *Question*)
- **Resource** oder **Joops.OfficeTalk.Resource**
Datentyp der Ressource (siehe Kapitel *Resource*)
- **Scheduledata** oder **Joops.OfficeTalk.ScheduleData**
Datentyp der temporären Vorgangsdaten (siehe Kapitel *scheduledata*)
- **ScriptDialog** oder **Joops.Scripting.ScriptDialog**
Datentyp eines Vorgangsdialoges (siehe Kapitel *ScriptDialog*)
- **Step** oder **Joops.OfficeTalk.Step**
Datentyp eines Arbeitsschrittes (siehe Kapitel *step*)
- **StepHistory** oder **Joops.OfficeTalk.StepHistory**
Datentyp der Arbeitsschritthistorie (siehe Kapitel *StepHistory*)
- **Team** oder **Joops.OfficeTalk.Team**
der Bearbeiter als Team (siehe Kapitel *worker*)
- **WSDLClient** oder **Joops.Scripting.WSDLClient**
Der Client zur Senden und Empfangen von Webservices (siehe Kapitel *WSDLClient*)
- **WSDLStruct** oder **Joops.Scripting.WSDLStruct**
Die Struktur der Ergebnisse eines Webservices (siehe Kapitel *WSDLStruct*)
- **<Systemname>.<CoClass>**
Für jedes dynamisch geladenes Interface ein Eintrag. Dabei steht <Systemname> für

den registrierten Namen des Systems und `<CoClass>` für eine CoClass der Automatisationsschnittstelle.

Globale Systemvariablen

Folgende Variablen und deren Namen sind in allen Makros ohne Deklaration bekannt:

- **action**
Die aktuelle Aktion unter der das Skriptmakro abläuft (siehe Kapitel *action*).
- **Error**
Die Fehlervariable. Die Variable liefert Informationen über den zuletzt aufgetretenen Fehler (siehe Kapitel *Error*).
- **process**
Der aktuelle Vorgang unter dem das Skriptmakro abläuft (siehe Kapitel *process*).
- **processdata**
Die permanenten Daten der Vorgangs (siehe Kapitel *processdata*).
- **scheduledata**
Die temporären Daten der Vorgangs (siehe Kapitel *scheduledata*).
- **step**
Der aktuelle Arbeitsschritt unter dem das Skriptmakro abläuft (siehe Kapitel *step*).
- **stepscheduler**
Er überwacht und führt die Aktionen eines Arbeitsschrittes aus (siehe Kapitel *StepHistory*).
- **worker**
Der aktuelle Bearbeiter, unter dem das Skriptmakro abläuft (siehe Kapitel *worker*).

Syntax der globalen Systemvariablendienste

Die generelle Syntax der Dienste aller Systemvariablen finden Sie im Kapitel *Call-Dienst*.

action

Anweisungen des Skripts können über diese globale Variable auf die laufende Aktion, in dem das Skript abläuft, zugreifen. Die Namen der Dienste beginnen mit `action..`

Syntax

defaultResultName

Der Dienst liefert den Namen des Standardergebnisses. Dieses Ergebnis wird verwendet, wenn das ausgeführte Skriptmakro kein Ergebnis liefert.

execution

Der Dienst liefert die Anzahl der bisherigen Ausführungen als Zahl. Die aktuelle Ausführung wird dabei nicht mitgezählt.

symbolNameString

Der Dienst liefert den Namen des Aktionssymbols in druckbarer Form.

isMandatory

Der Dienst liefert `True`, wenn die Aktion vorgeschrieben ist, ansonsten `False`.

isOptional

Der Dienst liefert `True`, wenn die Aktion optional ist, ansonsten `False`.

isExecuted

Der Dienst liefert `True`, wenn die Aktion bereits einmal ausgeführt wurde, ansonsten `False`.

processTime

Der Dienst liefert die geplante Bearbeitungszeit der Aktion.

scriptName

Der Dienst liefert den Namen des Skripts, das beim Start der Aktion ausgeführt wird. Der Dienst `step.scriptName.displayString` liefert den Namen in druckbarer Form.

symbolNames

Der Dienst liefert die Begriffe der möglichen Aktionssymbole. Die Ablaufhistorie einer Aktion kann z.B. mit einem Symbol, das durch diese Begriffe bezeichnet wird, belegt werden (siehe Dienst *actionSymbol* (<Zeichenkette>)). Die möglichen Begriffe lauten:

- **etwas Schreiben** für das Symbol des Schreibens
- **Verhandeln** für das Symbol des Verhandels
- **etwas Suchen** für das Symbol des Suchens
- **etwas berechnen** für das Symbol etwas zu berechnen
- **etwas delegieren** für das Symbol die Delegation eines Vorgangs
- **etwas Drucken** für das Symbol etwas auszudrucken
- **etwas Versenden** für das Symbol etwas per Post oder E-Mail zu versenden
- **etwas Geben** für das Symbol etwas zu geben
- **etwas Entgegennehmen** für das Symbol etwas entgegenzunehmen
- **etwas Entscheiden** für das Symbol eine Entscheidung zu treffen
- **etwas Fragen** für das Symbol etwas zu erfragen
- **jemanden Kontaktieren** für das Symbol für das Kontaktieren einer Person
- **Dialog Erfassung** für das Symbol per Dialog Daten oder Informationen zu erfassen
- **Vorgang starten** für das Symbol einen neuen Vorgang zu starten
- **Kommunikation mit externem System** für das Symbol mit einer anderen Applikation zu kommunizieren oder eine andere Applikation zu steuern
- **externe Datenbankaktion** für das Symbol aus einer Datenbank Informationen zu lesen oder in eine Datenbank Informationen zu schreiben
- **WebService verwenden** für das Symbol einen Webservice zu nutzen
- **eine Aktion** für das Symbol eine Aktion auszuführen. Dieses Symbol ist das Standardsymbol.

ActionHistory

Eine Aktionshistorie beinhaltet Informationen zu einer ausgeführten Aktion. Der Dienst *actions* der Systemvariablen *StepHistory* liefert die Historien der ausgeführten Aktionen.

Syntax**comment**

Der Dienst liefert den im Skriptmakro erzeugten Kommentar zur Aktion. Siehe Dienst *comment*(<Zeichenkette>) der Systemvariablen *step*.

consumed

Der Dienst liefert die Kosten der verbrauchten Ressourcen.

description

Der Dienst liefert die Beschreibung der Aktion.

endedAt

Der Dienst liefert Endedatum und Uhrzeit der Aktion

mandatory

Der Dienst liefert `True`, wenn die Ausführung der Aktion vorgeschrieben ist.

startingWorker

Der Dienst liefert den Namen des Bearbeiters, der die Aktion ausgeführt hat.

startedAt

Der Dienst liefert Startdatum und Uhrzeit der Aktion

Error

Die globale Variable `Error` der gleichnamigen Systemvariablen liefert Informationen über den gegenwärtigen Ausführungszustand des Makros. Die Variable ist automatisch deklariert, und somit im Makro immer verfügbar. Ein möglicher Anwendungsort ist der *Catch*-Anweisungsteil in der *Try-Catch*-Anweisung.

Syntax**Clear**

Löscht den Inhalt (`Description` und `Source`) der Fehlervariablen.

Description

Liefert den Fehlertext als `String`. Das ist der Text der Ausnahme wie er ohne eine *Try-Catch*-Anweisung von der Makroüberwachung gemeldet würde.

Hinweis: Wenn die Variable in einem *Try-Catch*-Blockes bei der Ausführung der `Shell (Wait)` Funktion verwendet wird, liefert der Dienst als Ergebnis `Array(<Fehlercode>, <Fehlertext>)`. `<Fehlercode>` ist dabei die Nummer des Fehlers als `Integer` und `<Fehlertext>` ist dabei die Beschreibung des Fehlers als `String`. Auf Windows-Plattformen entspricht `<Fehlercode>` der Umgebungsvariablen `ERRORLEVEL`. Auf Linux-Plattformen entspricht `<Fehlercode>` der Variablen `$?`.

Wenn die Variable in einem *Try-Catch*-Blockes bei der Ausführung des Dienstes `compileMessages(...)` der Systemvariablen `FileSystemObject` verwendet wird, liefert der Dienst als Ergebnis `Array(<Meldungsschlüssel>, <Fehlertext>)`. `<Meldungsschlüssel>` ist dabei der Schlüssel der fehlerhaften Meldung als `String` und `<Fehlertext>` ist dabei die Beschreibung des erkannten Fehlers als `String`.

Source

Liefert den Namen des Makros, in dem der Fehler erkannt wurde, als `String` im Format `<Skriptname>.<Makroname>` (siehe Kapitel Makro).

Beispiel

```
Dim zahl as Integer
```

```
Try
```

```
    zahl = zahl + 100
```

```
End Try
```

```
Catch
    MsgBox ("Additionsfehler: " & Error.Description)
End Catch
```

FileSystemObject

Mit einer Variablen vom Typ `FileSystemObject` können grundsätzliche Dateioperationen mit Verzeichnissen und Dateien durchgeführt werden. Die Variable muss im Deklarationsteil mit

```
Dim <Variable> As Joops.Scripting.FileSystemObject
```

deklariert werden.

Hinweis: Ein Dateiname kann als Präfix einen symbolischen Verzeichnisnamen enthalten. Z. B.: `$(temp)\Planung.txt`. Dabei wird `$(temp)` in das eigentliche Verzeichnis übersetzt. Einzelheiten dazu finden Sie im Kapitel *resolve (<Variable>)*.

Syntax

```
Dim fso As Joops.Scripting.FileSystemObject
```

actualizeMessages (<Arbeitsmeldungen>, with: <Originalmeldungen>)

Der Dienst aktualisiert die Meldungen in `<Arbeitsmeldungen>` mit den Meldungen in `<Originalmeldungen>`. Beide Argumente müssen ein Array sein. Zur Beschreibung des Aufbaus eines Array siehe Kapitel *readMessages (<Dateiname>)*. In `<Arbeitsmeldungen>` fehlende Meldungen werden aus `<Originalmeldungen>` übernommen. Meldungen in `<Arbeitsmeldungen>` werden entfernt, wenn sie in `<Originalmeldungen>` fehlen. Wurden `<Arbeitsmeldungen>` durch die Aktualisierung verändert, liefert der Dienst als Ergebnis die veränderten `<Arbeitsmeldungen>`. Wurden `<Arbeitsmeldungen>` nicht verändert, liefert der Dienst `Null`.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

compileMessages (<Dateiname>)

Der Dienst erstellt für die Meldungsdatei `<Dateiname>` die Indexdatei. Die Namenserverweiterung der Meldungsdatei muss `.lbl` lauten. Der Dienst liefert bei erfolgreicher Kompilierung `True`. Wurde ein Syntaxfehler erkannt, wirft der Dienst eine Ausnahme, die mit der Anweisung *Try-Catch* aufgefangen werden kann.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

copyFolder (<Source> to: <Destination>)

Der Dienst kopiert das mit `<Source>` angegebene Verzeichnis in das mit `<Destination>` angegebene Zielverzeichnis und liefert das Ergebnis als `Boolean` (`True`: Verzeichnis wurde kopiert, `False`: `<Source>` bezeichnet eine Datei oder das Verzeichnis wurde nicht kopiert). Derzeit wird dieser Dienst noch nicht unterstützt und liefert als Ergebnis `False`.

copyFile (<Source> to: <Destination>)

Der Dienst kopiert die mit `<Source>` angegebene Datei in die mit `<Destination>` angegebene Zielfile und liefert das Ergebnis als `Boolean` (`True`: Datei ist kopiert, `False`: `<Source>` bezeichnet ein Verzeichnis oder die Datei wurde nicht kopiert).

createFile (<Variable>)

Der Dienst legt die mit <Variable> angegebene Datei ohne Inhalt an und liefert das Ergebnis als Boolean (True: Datei wurde angelegt, False: <Variable> bezeichnet ein Verzeichnis oder die Datei wurde nicht angelegt).

createFolder (<Variable>)

Der Dienst legt das mit <Variable> angegebene Verzeichnis an und liefert das Ergebnis als Boolean (True: Verzeichnis wurde angelegt, False: Verzeichnis wurde nicht angelegt).

dateCreated(<Variable>)

Der Dienst liefert Erstellungsdatum und Uhrzeit der mit <Variable> angegebenen Datei oder des Verzeichnisses als Date und Time in einem Array.

Hinweis: Abweichend zur VB-Syntax muss für den Dienst nicht zuerst ein GetFile erfolgen.

dateLastAccessed(<Variable>)

Der Dienst liefert Datum und Uhrzeit des letzten Zugriffs der mit <Variable> angegebenen Datei oder des Verzeichnisses als Date und Time in einem Array.

Hinweis: Abweichend zur VB-Syntax muss für den Dienst nicht zuerst ein GetFile erfolgen.

dateLastModified(<Variable>)

Der Dienst liefert Datum und Uhrzeit der letzten Änderung der mit <Variable> angegebenen Datei oder des Verzeichnisses als Date und Time in einem Array.

Hinweis: Abweichend zur VB-Syntax muss für den Dienst nicht zuerst ein GetFile erfolgen.

deleteFile (<Variable>)

Der Dienst löscht die mit <Variable> angegebene Datei und liefert das Ergebnis als Boolean (True: Datei wurde entfernt, False: <Variable> bezeichnet ein Verzeichnis oder die Datei wurde nicht entfernt).

deleteFolder (<Variable>)

Der Dienst löscht das mit <Variable> angegebene Verzeichnis und liefert das Ergebnis als Boolean (True: Verzeichnis wurde entfernt, False: <Variable> bezeichnet eine Datei oder das Verzeichnis wurde nicht entfernt).

fileExist (<Variable>)

Der Dienst prüft, ob die mit <Variable> angegebene Datei existiert und liefert das Ergebnis als Boolean (True: Datei existiert, False: <Variable> bezeichnet ein Verzeichnis oder die Datei existiert nicht).

folderContents (<Variable>)

Der Dienst liefert den Inhalt des mit <Variable> angegebene Verzeichnisses als Array.
Der Dienst liefert Null wenn <Variable> kein Verzeichnis bezeichnet.

folderExist (<Variable>)

Der Dienst prüft, ob das mit <Variable> angegebene Verzeichnis existiert und liefert das Ergebnis als Boolean (True: Verzeichnis existiert, False: <Variable> bezeichnet eine Datei oder Verzeichnis existiert nicht).

getBaseName (<Variable>)

Der Dienst liefert den reinen Dateinamen ohne Verzeichnis und Namenerweiterung aus dem mit(<Variable>) benannten Dateinamen. Z.B.:

```
getBaseName ( c:\OfficeTalk\Bin\OfficeTalk.exe )  
ergibt  
OfficeTalk
```

getDriveName (<Variable>)

Der Dienst liefert den Laufwerksnamen aus dem mit(<Variable>) benannten Dateinamen. Z.B.:

```
getDriveName ( c:\OfficeTalk\Bin\OfficeTalk.exe )  
ergibt  
c:\
```

getExtensionName (<Variable>)

Der Dienst liefert die Namenserverweiterung aus dem mit(<Variable>) benannten Dateinamen. Z.B.:

```
getExtensionName ( c:\OfficeTalk\Bin\OfficeTalk.exe )  
ergibt  
exe
```

getFileName (<Variable>)

Der Dienst liefert den Dateinamen ohne den Laufwerksteil aus dem mit(<Variable>) benannten Dateinamen. Z.B.:

```
getFileName ( c:\OfficeTalk\Bin\OfficeTalk.exe )  
ergibt  
OfficeTalk\Bin\OfficeTalk.exe
```

getParentFolderName (<Variable>)

Der Dienst liefert den Namen der Elternverzeichnis aus dem mit(<Variable>) benannten Dateinamen. Z.B.:

```
getParentFolderName ( c:\OfficeTalk\Bin\OfficeTalk.exe )  
ergibt  
c:\OfficeTalk
```

getTempName

Der Dienst liefert einen temporären Dateinamen im benutzerspezifischen Verzeichnis. Das Verzeichnis ist abhängig vom Systemlogin (siehe Kapitel *getTempDirectoryName*).

getTempDirectoryName

Der Dienst liefert das benutzerspezifische temporäre Verzeichnis. Auf Linux-Plattformen verweist die Umgebungsvariable \$HOME auf das Verzeichnis. Auf Windows-Plattformen verweist die Umgebungsvariable \$TEMP auf das Verzeichnis.

moveFolder (<Source> to: <Destination>)

Der Dienst verschiebt das mit <Source> angegebene Verzeichnis in das mit <Destination> angegebene Zielverzeichnis und liefert das Ergebnis als Boolean (True: Verzeichnis wurde verschoben, False: <Source> bezeichnet eine Datei oder das Verzeichnis wurde nicht verschoben). Derzeit wird dieser Dienst noch nicht unterstützt und liefert als Ergebnis False.

moveFile (<Source> to: <Destination>)

Der Dienst verschiebt die mit <Source> angegebene Datei in die mit <Destination> angegebene Zielfeile und liefert das Ergebnis als Boolean (True: Datei wurde verschoben, False: <Source> bezeichnet ein Verzeichnis oder die Datei wurde nicht verschoben).

readMessages (<Dateiname>)

Der Dienst liest die mit <Dateiname> benannte Meldungsdatei. Die Namensweiterung der Meldungsdatei muss **.lbi** lauten. Der Dienst liefert das Ergebnis in einem Array. Dabei ist jede einzelne Meldung wiederum als Schlüssel/Meldungstext-Paar in ein Array gepackt. Der Zeichensatz des Betriebssystems wird beim Lesen verwendet.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

readMessages (<Dateiname>, encoding: <Zeichensatz>)

Siehe *readMessages (<Dateiname>)*. Als Zeichensatz beim Lesen wird <Zeichensatz> verwendet. Erlaubte Zeichensätze sind **Default**, **UTF-8** und **UTF-16**. Bei Zeichensätzen mit 2-Byte-Code muss **UTF-8** oder **UTF-16** verwendet werden. Das Argument <Zeichensatz> muss als String (Zeichenkette) angegeben werden.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

resolve (<Variable>)

Der Dienst löst einen Dateinamen mit einem symbolischen Verzeichnis auf und liefert den aufgelösten Dateinamen. Ein symbolisches Verzeichnis **\$(<Name>)** bezeichnet über eine OfficeTalk-Einstellung oder über eine Umgebungsvariable der Systemplattform den Namen des Verzeichnisses. Weitere Einzelheiten dazu finden Sie im Kapitel *Dokument einfügen* der Dokumentation Business-Process-Management.

size(<Variable>)

Der Dienst liefert die Dateigröße der mit <Variable> angegebenen Datei oder des Verzeichnisses als Integer. Wenn <Variable> ein Verzeichnis bezeichnet, liefert der Dienst die Größe aller Dateien in dem Verzeichnis.

Hinweis: Abweichend zur VB-Syntax muss für den Dienst nicht zuerst ein `GetFile` erfolgen.

translations (<Meldungen>, proofings: <Korrekturlesungen>)

Der Dienst übernimmt die Kennzeichen der Korrekturlesungen aus dem Argument `Korrekturlesungen` in das Argument `Meldungen`. Das Array `Meldungen` muss für jede Meldung ein Array(<String>, <String>, <Boolean>) enthalten, wobei das erste Element der Name der Meldung, das zweite Element der Meldungstext, und das dritte Element das Kennzeichen für die Korrekturlesung ist. Das Array `Korrekturlesungen` muss für jede Korrektur gelesene Meldung ein Array(<String>, <Boolean>) enthalten, wobei das erste Element der Name der Meldung, das zweite Element das Kennzeichen für die Korrekturlesung ist.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

validateMessages (<Meldungen>)

Der Dienst prüft die Meldungstexte im Argument `Meldungen` auf syntaktische Korrektheit. Das Ergebnis des Dienstes ist `Null`, wenn alle Meldungstexte syntaktisch korrekt sind, oder der Name der ersten syntaktisch fehlerhaften Meldung. Das Array `Meldungen` muss

für jede Meldung ein Array(<String>, <String>) enthalten, wobei das erste Element der Name der Meldung und, das zweite Element der Meldungstext ist. Die Syntaxregeln einer Meldung sind im Übersetzungsprozess beschrieben.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

writeMessages (<Meldungen>, to: <Dateiname>)

Der Dienst schreibt die Meldungszeilen in dem Array <Meldungen> in die Datei <Dateiname>. Die Namensweiterung der Meldungsdatei muss **.lbl** lauten. <Meldungen> muss für jede Meldung ein Array mit Schlüssel und Meldungstext enthalten. Der Zeichensatz des Betriebssystems wird beim Schreiben verwendet.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

writeMessages (<Meldungen>, to: <Dateiname>, encoding: <Zeichensatz>)

Siehe *writeMessages (<Meldungen>, to: <Dateiname>)*. Als Zeichensatz beim Schreiben wird <Zeichensatz> verwendet. Erlaubte Zeichensätze sind **Default**, **UTF-8** und **UTF-16**. Bei Zeichensätzen mit 2-Byte-Code muss **UTF-8** oder **UTF-16** verwendet werden. Das Argument <Zeichensatz> muss als String (Zeichenkette) angegeben werden.

Hinweis: Der Dienst ist nur für Sonderanwendungen im OEM-Bereich erforderlich.

Beispiel

```
Dim dialog As FileSystemObject
fso = New Joops.Scripting.FileSystemObject
MsgBox( fso.copyFile( "c:\temp\Test.txt" to: "c:\temp\Copytest.txt" ))
MsgBox( fso.deleteFile( "c:\temp\Test.txt" ))
MsgBox( fso.folderContents( "c:\temp" ))
```

FileDialog

Mit einer Variablen vom Typ `FileDialog` wird ein Auswahldialog geöffnet, der eine Datei zum Lesen oder Schreiben auswählt. Die Variable muss im Deklarationsteil mit

```
Dim <Variable> As Joops.Scripting.FileDialog
```

deklariert werden.

Syntax

```
Dim dialog As Joops.Scripting.FileDialog
```

directory (<Variable>, title: <Titel>)

Der Dialog zur Auswahl eines Verzeichnisses wird mit dem angegebenen Titel geöffnet. Der Dialog ermöglicht auch das Erstellen von neuen Verzeichnissen. <Variable> gibt die Voreinstellung für das initiale Verzeichnis an. Der Dienst liefert das gewählte Verzeichnis als String, oder Null, wenn kein Verzeichnis ausgewählt wurde.

open

Der Dienst öffnet den Dateiauswahldialog zum Lesen einer Datei mit dem Dateisuchmuster `*.*`. Der Dienst liefert die gewählte Datei als String, oder Null, wenn keine Datei ausgewählt wurde.

open (<Variable>)

Der Dienst öffnet den Dateiauswahldialog zum Lesen einer Datei mit dem Dateisuchmuster aus der Variablen. Der Dienst liefert die gewählte Datei als String, oder Null, wenn keine Datei ausgewählt wurde.

open (<Variable>, title: <Titel>)

Wie oben, der Dateiauswahldialog zum Lesen einer Datei erhält zusätzlich den angegebenen Titel.

openTitle(<Titel>)

Der Dienst öffnet den Dateiauswahldialog zum Lesen einer Datei mit dem Dateisuchmuster `*.*` und dem angegebenen Titel. Der Dienst liefert die gewählte Datei als `String`, oder `Null`, wenn keine Datei ausgewählt wurde.

save

Der Dienst öffnet den Dateiauswahldialog zum Speichern einer Datei mit dem Dateisuchmuster `*.*`. Der Dienst liefert die gewählte Datei als `String`, oder `Null`, wenn keine Datei ausgewählt wurde.

save (<Variable>)

Der Dienst öffnet den Dateiauswahldialog zum Speichern einer Datei mit dem Dateisuchmuster aus der Variablen. Der Dienst liefert die gewählte Datei als `String`, oder `Null`, wenn keine Datei ausgewählt wurde.

save (<Variable>, title: <Titel>)

Wie oben, der Dateiauswahldialog erhält zusätzlich den angegebenen Titel.

saveTitle(<Titel>)

Der Dienst öffnet den Dateiauswahldialog zum Speichern einer Datei mit dem Dateisuchmuster `*.*` und dem angegebenen Titel. Der Dienst liefert die gewählte Datei als `String`, oder `Null`, wenn keine Datei ausgewählt wurde.

Beispiel

```
Dim dialog As FileDialog
Dim fileName As String
dialog = New Joops.Scripting.FileDialog
fileName = dialog.open
MsgBox( fileName )
```

HTTPClient

Mit der Systemvariablen `HTTPClient` können Anweisung an Internetapplikationen, die das XML-, oder REST-Request-Protokoll unterstützen, gesandt werden. Die Anweisungsarten werden unterteilt in **DELETE**, **GET**, **POST** und **PUT**. Die zu verwendende Anweisungsart hängt von der Aufgabe und von der Internetapplikation ab. Der Empfänger der Anweisung ist eine Internetadresse (`http://...` oder `https://...`). Die Antwort der Internetapplikation erfolgt im XML-Format und untergliedert sich in Kopfteil (Header) und Nutzdaten (Body). Die Nutzdaten liegen, abhängig von der Arbeitsweise der Internetapplikation im XML- oder im JSON-Format vor. Kopfteil und Nutzdaten sind ihrerseits in Felder, auch Attribute genannt, unterteilt. Um durch HTTP-Anweisungen mit einer Internetapplikation zu kommunizieren, bedarf es genauer Kenntnis der zu verwendenden HTTP-Anweisungssyntax! Der generelle Ablauf der Bearbeitung einer Anweisungen ist:

1. Die Anweisung wird vorbereitet oder sofort an den Empfänger gesandt (z.B.: `delete()` oder `executeGet(...)`).
2. Der Empfänger verarbeitet die Anweisung.
3. Die Antwort des Empfängers auf die Ausführung wird angefordert. Sie kann sich im Kopfteil (siehe Kapitel *header(<Variable>)*) oder in den Nutzdaten (siehe Kapitel *body(<Variable>)*) befinden. Nutzdaten liegen, abhängig von der Arbeitsweise des Webrowsers, im XML- oder im JSON-Format vor. Die entsprechenden Dienste erkennen das selbstständig.

Die Variable muss im Deklarationsteil mit

```
Dim <Variable> As Joops.Scripting.HTTPClient
```

deklariert werden.

Hinweis: Beim Absenden einer Anforderung erfolgt mit den Angaben **Benutzer** und **Passwort** aus der Registerseite **HTTP** der allgemeinen Einstellungen in OfficeTalk eine Anmeldung beim Verbindungsprovider. Falls eine der Angaben in der Registerseite fehlt, werden Sie mit dem Dialog **HTTP-Anmeldung** danach gefragt (siehe Dokumentation Business-Process-Management Kapitel *Register HTTP*).

Der Verbindungsaufbau erfolgt; falls in der Registerseite **HTTP** der allgemeinen Einstellungen in OfficeTalk nichts anderes angegeben ist, bei der Internetadresse `http://...` über den Port **80**, und bei der Internetadresse `https://...` über den gesicherten Port **443**.

Werden in einer Anweisung (z.B. `delete(...)`, `execute`, `executeGet(...)`, usw.) dem Empfänger Zeichenketten mit Sonderzeichen übergeben, so müssen diese Sonderzeichen, **HTML**-konform codiert sein (siehe dazu Kapitel *asHTMLString(<Variable>)*).

Vom Hersteller des Webserver erfahren Sie ob die Nutzdaten im XML- oder im JSON-Format geliefert werden. Das CRM-System **vtiger** liefert z.B. die Nutzdaten JSON-Format. Das Projektverwaltungssystem **CoP** liefert z.B. die Nutzdaten im XML-Format.

Syntax

```
Dim client As Joops.Scripting.HTTPClient
```

asJSON(<Array>)

Der Dienst erstellt aus dem Inhalt von `<Array>` eine JSON-konforme Zeichenkette. Diese Zeichenkette muss eventuell noch um die schließenden Klammern `{` und `}` ergänzt werden, und kann dann in Diensten mit dem Argument `contents:` (z.B. `executePost(..., contents: ...)`) als Wert für das Argument `contents:` verwendet werden. `<Array>` enthält für jedes Datum ein Array mit zwei Einträgen. Der erste Eintrag ist dessen Name. Der zweite Eintrag ist dessen Wert. Ein Beispiel:

Aus `Array(Array("betrag", 220.00), Array("ort", "München"))` wird die JSON-Zeichenkette `"betrag":200.00,"ort":"München"`.

Hinweis: Das JSON-Format muss bei Webservern, die das JSON-Datenprotokoll voraussetzen, für die zu sendenden Daten verwendet werden.

body

Der Dienst liefert aus der Antwort einer Anweisung alle Nutzdaten, auch Body genannt, als Zeichenkette.

body(<Variable>)

Der Dienst liefert aus den Nutzdaten der Antwort, auch Body genannt, den Wert des Attributes mit dem Namen <Variable>. <Variable> kann einen Abschnittspfad, in dem sich das Attribut befindet, als `Array`, oder nur den Namen des Attributes enthalten. (Beispiel siehe Kapitel *Beispiel 2*). I.d.R. enthält das Ergebnis einen einzelnen Attributwert.

Nutzdaten im XML-Format

Wenn sich das Attribut in dem Abschnitt und den untergeordneten Abschnitten befindet, liefert der Dienst alle Attributwerte als `Array`.

Nutzdaten im JSON-Format

Der Wert des Attributes aus der ersten Fundstelle wird geliefert. Wenn <Variable> eine Attributgruppe bezeichnet, liefert der Dienst ein `Array` für die Attributgruppe mit je einem `Array` pro Attribut. Der erste Eintrag des Attributarrays ist der Attributname. Der zweite Eintrag des Attributarrays ist der Attributwert.

Beispiele

```
body("begin")
```

Liegen die Nutzdaten im XML-Format vor, wird das Attribut `begin` aus allen Abschnitten geliefert. Liegen die Nutzdaten im JSON-Format vor, wird das Attribut `begin` aus dem ersten gefundenen Abschnitt geliefert.

```
body(Array("assignments-and-accountings", "assignments", assignment, "begin"))
```

Liegen die Nutzdaten im XML-Format vor, wird das Attribut `begin` aus allen `assignment`-Abschnitten geliefert, wenn der Abschnitt (`"assignments-and-accountings"`, `"assignments"`) mehrere Abschnitte mit dem Namen `assignment` enthält. Liegen die Nutzdaten im JSON-Format vor, wird das Attribut `begin` aus dem ersten gefundenen Abschnitt geliefert.

body(<Variable>, in: <Abschnittspfad>)

Der Dienst liefert aus den Nutzdaten der Antwort, auch Body genannt, den Wert des Attributes mit dem Namen <Variable> aus dem Abschnitt mit dem Namen <Abschnittspfad>. Sowohl <Variable> als auch <Abschnittspfad> können einen Pfad innerhalb den Nutzdaten zum Attribut beschreiben.

Nutzdaten im XML-Format

Ein `Array` mit je einem `Array` pro Attributgruppe wird geliefert. <Variable> kann ein oder mehrere Attributnamen als `Array` benennen.

Enthalten die Nutzdaten einen Abschnitt mit mehreren gleich benannten untergeordneten Abschnitten, ähnlich einer Liste, und soll der Dienst die gesamte Liste liefern, muss `<Abschnittspfad>` mit dem übergeordneten Abschnittsnamen enden und jedem Attributnamen in `<Variable>` muss sein letzter Abschnitt vorangestellt werden (Beispiel siehe auch Kapitel *Beispiel 3*).

Nutzdaten im JSON-Format

Der Wert des Attributes aus der ersten Fundstelle wird geliefert. Wenn `<Variable>` eine Attributgruppe bezeichnet, liefert der Dienst ein `Array` für die Attributgruppe mit je einem `Array` pro Attribut. Der erste Eintrag des Attributarrays ist der Attributname. Der zweite Eintrag des Attributarrays ist der Attributwert.

Alle Attribute als Liste (XML- Nutzdaten)

Der Abschnitt `Array('assignments-and-accountings', 'assignments')` enthält den Abschnitt `'assignment'` mit mehreren Einträgen, ähnlich einer Liste. Um diese Liste zu erhalten, muss `<Abschnittspfad>` mit dem Abschnitt `assignments` enden und jedem Attribut muss der Abschnitt `assignment` vorangestellt werden:

```
body(Array(Array('assignment', 'id'), Array('assignment', 'account-type'), Array('assignment', 'project-name'), Array('assignment', 'begin')), in: Array('assignments-and-accountings', 'assignments'))
```

Nur die Attribute der ersten Zeile der Liste (XML- Nutzdaten)

Wird der Abschnittsname `assignment` statt dessen im Argument `<Abschnittspfad>` angegeben, liefert der Dienst maximal eine Zeile als Ergebnis:

```
body(Array('id', 'account-type', 'project-name', 'begin'), in: Array('assignments-and-accountings', 'assignments', 'assignment'))
```

connect

Der Dienst baut die Verbindung zum Host auf. Mit den Diensten `host(<Variable>)` und `port(<Variable>)` wird der Host und der Port für die Verbindung bestimmt. Wenn die Verbindung aufgebaut werden konnte, liefert der Dienst `True`, ansonsten `False`.

delete(<Variable>)

Der Dienst bereitet eine **DELETE**-Anweisung vor. Die `<Variable>` spezifiziert den Empfänger der Anweisung.

disconnect

Der Dienst trennt eine bestehende Verbindung.

execute

Mit dem Dienst wird eine Anweisung, die mit dem Dienst `get(<Variable>)`, `delete(<Variable>)`, `post(<Variable>, contents: <Anweisung>)` oder `put(<Variable>, contents: <Anweisung>)` vorbereitete wurde, an den dort angegebenen Empfänger gesandt. Falls bei der Ausführung des Dienstes ein Fehler auftritt, wird eine Ausnahme geworfen, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann der Fehler statt dessen programmatisch bearbeitet werden.

executeDelete(<Variable>)

Der Dienst sendet an den mit `<Variable>` bezeichneten Empfänger eine **DELETE**-Anweisung. Falls bei der Ausführung des Dienstes ein Fehler auftritt, wird eine Ausnahme ge-

worfen, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann der Fehler statt dessen programmatisch bearbeitet werden.

executeGet(<Variable>)

Der Dienst sendet an den mit <Variable> bezeichneten Empfänger eine **GET**-Anweisung. Falls bei der Ausführung des Dienstes ein Fehler auftritt, wird eine Ausnahme geworfen, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann der Fehler statt dessen programmatisch bearbeitet werden.

executePost(<Variable>, contents: <Anweisung>)

Der Dienst sendet an den mit <Variable> bezeichneten Empfänger eine **POST**-Anweisung. Die Variable <Anweisung> enthält die Anweisungsdetails im HTML-Format. Falls bei der Ausführung des Dienstes ein Fehler auftritt, wird eine Ausnahme geworfen, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann der Fehler statt dessen programmatisch bearbeitet werden.

executePut(<Variable>, contents: <Anweisung>)

Der Dienst sendet an den mit <Variable> bezeichneten Empfänger eine **PUT**-Anweisung. Die Variable <Anweisung> enthält die Anweisungsdetails im HTML-Format. Falls bei der Ausführung des Dienstes ein Fehler auftritt, wird eine Ausnahme geworfen, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann der Fehler statt dessen programmatisch bearbeitet werden.

get(<Variable>)

Der Dienst bereitet eine **GET**-Anweisung vor. Die <Variable> spezifiziert den Empfänger der Anweisung.

header

Der Dienst liefert alle Kopfteile der Antwort, auch Header genannt, auf die letzte Anweisung als `Array`. Dabei ist das Format der meisten Antwortfelder `Array(<Attributname>, <Attributwert>)`.

header(<Variable>)

Der Dienst liefert aus dem Kopfteil der Antwort auf die letzte Anweisung, auch Header genannt, den Inhalt des Attributes mit dem Namen <Variable> als `Array`. <Variable> kann dabei einen Bereich, in dem sich das Attribut befindet, oder nur den Namen des Attributes enthalten. Das Ergebnis ist ein `Array` der Attributwerte. Beispiel siehe Kapitel *Beispiel 1*. I.d.R. enthält das `Array` einen einzelnen Attributwert. Wenn das Attribut im angegebenen Bereich der Kopfdaten jedoch mehrfach enthalten ist, enthält der `Array` mehrerer Einträge.

header(<Variable>, put: <Inhalt>)

Der Dienst belegt in der Anweisung, die mit dem Dienst `delete(<Variable>)`, `get(<Variable>)`, `post(<Variable>, contents: <Anweisung>)` oder `put(<Variable>, contents: <Anweisung>)` vorbereitet wurde, im Kopfteil das Attribut mit dem Namen <Variable> mit der Zeichenkette <Inhalt>. Mit dem Dienst kann eine Anweisung schrittweise erstellt, und mit dem Dienst `execute` versandt werden.

hasResponse

Der Dienst liefert `True`, wenn ein Ergebnis der letzten Anweisung vorliegt.

host(<Variable>)

Mit dem Dienst wird der Host für die Verbindung bestimmt. <Variable> bezeichnet den Host im URL-Format.

isConnected

Der Dienst liefert `True`, wenn der die Verbindung zum Host aufgebaut ist, ansonsten `False`.

isValid

Der Dienst liefert `True`, wenn das Ergebnis der letzten Anweisung gültig ist.

keepAlive (<Variable>)

Der Dienst bestimmt, ob die Verbindung zum Host nach der Ausführung des HTTP-Kommando (z.B. *execute*) beibehalten (`True`), oder getrennt (`False`) werden soll. `False` ist vorgelegt.

post (<Variable>, contents: <Anweisung>)

Der Dienst bereitet eine **POST**-Anweisung vor. Die <Variable> spezifiziert den Empfänger der Anweisung. Die Variable <Anweisung> enthält die Anweisungsdetails im HTML-Format.

port(<Variable>)

Mit dem Dienst wird der Port für die Verbindung bestimmt. <Variable> bezeichnet den Port numerisch. Der Port **80** ist vorgelegt.

protocol

Der Dienst liefert den Namen und die Version des verwendeten HTTP-Protokolls.

put (<Variable>, contents: <Anweisung>)

Der Dienst bereitet eine **PUT**-Anweisung vor. Die <Variable> spezifiziert den Empfänger der Anweisung. Die Variable <Anweisung> enthält die Anweisungsdetails im HTML-Format.

responseString

Der Dienst liefert das Ergebnis der letzten Anweisung als Zeichenkette.

status

Der Dienst liefert den Status der letzten Anweisung als `Integer`. Wobei Werte > 99 und < 600 für eine ordnungsgemäße Ausführung stehen. Alle anderen Werte stehen für einen Ausführungsfehler.

statusString

Der Dienst liefert den Status der letzten Anweisung als Zeichenkette.

version

Der Dienst liefert die aktuelle HTTP-Version.

Beispiel 1

```
Dim client As HTTPClient
Dim ergebnis As Array

client = New Joops.Scripting.HTTPClient
Try
    client.executeGet("http://www.yahoo.com")
    ergebnis = client.header("date")
    If UBound(ergebnis) >= 1 Then
        MsgBox("Kopfdatenergebnis für date: " & ergebnis(0))
    Else
        MsgBox("Kein Kopfdatenergebnis für date !")
    End If
End Try
Catch
    MsgBox("HTTP-Fehler: " & client.lastError,,vbCVritical,"HTTP-Fehler")
End Catch
```

Beispiel 2

```
Dim client As HTTPClient
Dim ergebnis As Array

Try
    client.executeGet("http://217.146.139.126/coptrack/mobileAccess?
        user=springerJ&pwd=&command=doGetAssignmentsAndAccountings")
    ergebnis = client.body( Array ("assignments-and-accoutings",
        "assignments","begin") )
    If UBound(ergebnis) >= 1 Then
        MsgBox("Nutzdatenergebnis für begin: " & Join(ergebnis, ","))
    Else
        MsgBox("Kein Nutzdatenergebnis für begin !")
    End If
End Try
Catch
    MsgBox("HTTP-Fehler: " & error.Description,,vbCVritical,
        "HTTP-Fehler")
End Catch
```

Beispiel 3

```
Dim client As HTTPClient
Dim ergebnis As Array
Dim zeile As Array
```



```
Dim i As Array

Try
    client.executeGet("http://217.146.139.126/coptrack/mobileAccess?
        user=springerJ&pwd=&command=doGetAssignmentsAndAccountings"
    ergebnis = client.body(Array(Array('assignment', 'id'),
        Array('assignment', 'account-type'),
        Array('assignment', 'project-name'),
        Array('assignment', 'begin'))
        in: Array('assignments-and-accountings', 'assignments'))
    For i = 1 To UBound(ergebnis) Step 1
        zeile = ergebnis(i - 1)
        MsgBox("Nutzdatenergebnis für " & i & "-te Zeile: " &
            Join(zeile, ","))
    Next
End Try
Catch
    MsgBox("HTTP-Fehler: " & error.Description,,vbCVritical,
        "HTTP-Fehler")
End Catch
```

Mail

Mit einer Variablen vom Typ **Mail** werden E-Mails auf der Basis von SMTP versandt. Die Variable muss im Deklarationsteil mit `Dim <Variable> As Joops.Scripting.Mail` deklariert werden. Abhängig von der Einstellung in OfficeTalk Environment **Mail** wird die E-Mail mit oder ohne SSL versandt.

Syntax

```
Dim mail As Joops.Scripting.Mail
```

addAttachment(<Variable>)

Mit dem Dienst wird eine Datei, die als Anhang verwendet werden soll, angegeben. Es können mehrere Dateien mit wiederholten **addAttachment**-Aufrufen angehängt werden.

authenticate(<Variable>)

Mit **True** für <Variable> sind die Angaben *user* (<Variable>) und *password* (<Variable>) für die Anmeldung am Rechner erforderlich. Mit **False** für <Variable> erfolgt die Anmeldung am Rechner anonym. Die Angabe übersteuert die allgemeine Einstellung im Register **Mail** (siehe Dokumentation [Business-Process-Management](#) Kapitel *Register Mail*).

cc(<Variable>)

Mit <Variable> wird der CC-Empfänger der E-Mail benannt. <Variable> kann eine einzelne E-Mailadresse als **String** oder ein **Array** mit mehreren E-Mailadressen sein.

delete (<Variable>)

Der Dienst löscht die E-Mails mit den im **Array** angegebenen ID's. <Variable> ist ein **Array** mit den ID's. Die ID stammt dem ersten Ergebniseintrag einer E-Mail der Dienste *receiveAndRemove* (<Variable>), *receiveFrom* (<Sender>, subject: <Betreff>, text: <Text>, host: <Rechner>, user: <Benutzer>, password: <Passwort>, remove: < Variable >), *receiveFrom* (<Sender>, subject: <Betreff>, host: <Rechner>, user: <Benutzer>, password: <Passwort>, remove: <Variable >) oder *receiveFrom* (<Sender>, host: <Rechner>, user: <Benutzer>, password: <Passwort>, remove: < Variable >). Das Ergebnis des Dienstes ist die Anzahl der gelöschten E-Mails oder -1 für einen Fehler.

Hinweis: Mit den Diensten *host* (<Variable>), *user* (<Variable>) und *password* (<Variable>) muss das Login für die Mailbox festgelegt werden, falls es von den Einstellungen im Register **Mail** (siehe Dokumentation [Business-Process-Management](#) Kapitel *Register Mail*) abweicht.

from(<Variable>)

Mit <Variable> wird der Absender der E-Mail benannt.

host (<Variable>)

Mit <Variable> wird der Internetname des Rechners, der das Versenden der E-Mail übernimmt, benannt. Als Voreinstellung wird die Angabe **Host** aus den allgemeinen OfficeTalk Einstellungen **Mail** verwendet.

mailFrom(<Sender>, to: <Receiver>, host: <Host>, user: <Username>, subject: <Subject>, text: <Text>, attachment: <Filename>)

Der Dienst ist ein Alias für den gleichnamigen Dienst *sendFrom*(<Sender>, to: <Receiver>, host: <Host>, user: <Username>, subject: <Subject>, text: <Text>, attachment: <Filename>) für und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

mailFrom(<Sender>, to: <Receiver>, host: <Host>, user: <Username>, subject: <Subject>, text: <Text>)

Der Dienst ist ein Alias für den gleichnamigen Dienst *sendFrom* (<Sender>, to: <Receiver>, host: <Host>, user: <Username>, subject: <Subject>, text: <Text>) für und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

mailFrom(<Sender>, to: <Receiver>, host: <Host>, user: <Username>, subject: <Subject>, attachment: <Filename>)

Der Dienst ist ein Alias für den gleichnamigen Dienst *sendFrom* (<Sender>, to: <Receiver>, host: <Host>, user: <Username>, subject: <Subject>, attachment: <Filename>) für und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

mailFrom(<Sender>, receiver: <Receiver>, host: <Host>, username: <Username>, ...)

Der Dienst ist ein Alias für den gleichnamigen Dienst *sendFrom* (<Sender>, receiver: <Receiver>, host: <Host>, username: <Username>, ...) für und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

password (<Variable>)

Mit <Variable> wird das Passwort für die Anmeldung des Benutzers *user* auf dem mit *host* benannten Rechner angegeben. Als Voreinstellung wird die Angabe **Passwort** aus den allgemeinen OfficeTalk Einstellungen **Mail** verwendet.

receiveAndRemove (<Variable>)

Beispiel für E-Mailtext

Mit dem Dienst werden aus den Postfach alle E-Mails, die mit den Auswahlkriterien übereinstimmen, abgeholt. Die Auswahlkriterien werden mit den Diensten *from*(<Variable>), *subject*(<Variable>) und *text*(<Variable>) gesetzt und können die Wildcards * und ? enthalten. Keine Angabe (Null) für den Dienst *from*(<Variable>) entspricht der Angabe * für alle Absender. Wenn das Argument <Variable> True ist, werden die abgeholten E-Mails auch im Postfach gelöscht. Das Ergebnis des Dienstes ist

```
Array(Array(<ID>, <Absender>, <Betreff>, <Text>, <Anhänge>, <Am>,
           <HTMLText>), ...)
```

<ID>: Die eindeutige ID der E-Mail im Postfach als String

<Absender>: Array (<E-Mailadresse>, <Name>)

<E-Mailadresse> E-Mailadresse des Absenders als String

<Name> Name des Absenders als String (leer falls nicht angegeben)

Falls der Absender nicht ermittelt werden kann, ist das Array leer.

<Betreff>: Die Betreffzeile der E-Mail als String

<Text> : Der Text der E-Mail aus dem Tag **<body>** als String exclusive HTML-Tags

Im Text enthalten URL's (z.B.: Tags **<a href...>**) sind statt dem Tag **<a href...>** mit { { und } } umschlossen.

<Anhänge>: Die Dateinamen der Anlagen der E-Mail als Array.

<Am>: Array(<Datum>, <Uhrzeit>)

<Datum> Das Empfangsdatum der E-Mail

<Uhrzeit> Die Empfangsuhrzeit der E-Mail

<HTMLText>: Der Text der E-Mail aus dem Tag **<body>** als String inklusive HTML-Tags

Anlagen werden im allgemeinen Verzeichnis für E-Mail-Anhänge gespeichert (siehe Dokumentation Business-Process-Management Kapitel *Register Mail*, Abschnitt *Anlagen in*).

Hinweis: Falls die Anmeldung am Postfach des Mailservers nicht mit den Diensten *host* (<Variable>), *user* (<Variable>) und *password* (<Variable>) benannt wurde, wird die Anmeldung aus den generellen Einstellungen verwendet (siehe Dokumentation Business-Process-Management Kapitel *Register Mail*). In jedem Fall müssen diese Anmeldedaten vorhanden sein. Andernfalls wird der Dienst mit einer entsprechenden Fehlermeldung quittiert.

Beispiel für <body> einer E-Mailtext

Der **<body>** der Beispielsmail ohne HTML-Tags im vierten Arrayeintrag <Text>

Sehr geehrte Damen und Herren,

besuchen Sie {{www.joops.com/OfficeTalkSuite.htm}}OfficeTalk,
um Ihr Unternehmen vorwärts zu bringen !

mit freundlichen Grüßen / best regards,
Josef Springer
(Geschäftsleitung/Management)

Der **<body>** der Beispielsmail mit HTML-Tags im sechsten Arrayeintrag <HTMLText>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
  </head>
  <body text="#000000" bgcolor="#ffffff">
    Sehr geehrte Damen und Herren,<br>
    <br>
    besuchen Sie <a
href="www.joops.com/OfficeTalkSuite.htm">OfficeTalk</a>,
    um Ihr Unternehmen vorw&auml;rts zu bringen !<br>
    <div class="moz-signature">
      <title>Signature</title>
      <p>mit freundlichen Gr&uuml;&szlig;en / best regards,<br>
        Josef Springer<br>
        (Gesch&auml;ftsleitung/Management)</p>
    <br>
    </div>
  </body>
</html>
```

receiveFrom(<Sender>, subject: <Betreff>, text: <Text>, host: <Rechner>, user: <Benutzer>, password: <Passwort>, remove: < Variable >)

Mit dem Dienst wird aus den Eingangspostfach die erste E-Mail, die mit den optionalen Auswahlkriterien <Sender>, <Betreff> und <Text> übereinstimmen, abgeholt. Zu den Auswahlkriterien, dem Argument *remove*: und den Anhängen siehe *receiveAndRemove* (<Variable>).

receiveFrom(<Sender>, subject: <Betreff>, host: <Rechner>, user: <Benutzer>, password: <Passwort>, remove: <Variable>)

Der Dienst entspricht dem oben beschriebenen Dienst, jedoch ohne `<Text>`.

receiveFrom(`<Sender>`, host: `<Rechner>`, user: `<Benutzer>`, password: `<Passwort>`, remove: `< Variable >`)

Der Dienst entspricht dem oben beschriebenen Dienst, jedoch ohne `<Betreff>` und `<Text>`.

receiver (`<Variable>`)

Der Dienst ist ein Alias für den Dienst `to (<Variable>)` und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

replyTo(`<Variable>`)

Mit `<Variable>` wird die Antwortadresse bestimmt. Wenn der Empfänger die E-Mail direkt beantwortet, wird die Antwort an diese Adresse gesandt. Wenn `replyTo` nicht angegeben ist, wird als Antwortadresse die Senderadresse verwandt.

sender(`<Variable>`)

Der Dienst ist ein Alias für den Dienst `from(<Variable>)` und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

send

Der Dienst ist ein Alias für den Dienst `sendText`. Das Ergebnis des Dienstes ist `True`, wenn die E-Mail versandt wurde, in allen anderen Fällen `False`.

sendFrom (`<Sender>`, to: `<Receiver>`, host: `<Host>`, user: `<Username>`, subject: `<Subject>`, text: `<Text>`)

Der Dienst entspricht dem oben beschriebenen Dienst, jedoch ohne Anhang.

sendFrom (`<Sender>`, to: `<Receiver>`, host: `<Host>`, user: `<Username>`, subject: `<Subject>`, attachment: `<Filename>`)

Der Dienst entspricht dem gleichnamigen Dienst, jedoch ohne Text. Wenn nur ein Anhang ohne Text versendet werden soll, empfiehlt sich diese Form.

sendFrom(`<Sender>`, to: `<Receiver>`, host: `<Host>`, user: `<Username>`, subject: `<Subject>`, text: `<Text>`, attachment: `<Filename>`)

Der Dienst sendet die durch die Parameter beschriebene E-Mail. Die einzelnen Parameter sind in den gleichnamigen Diensten beschrieben.

sendFrom (`<Sender>`, receiver: `<Receiver>`, host: `<Host>`, username: `<Username>`, ...)

Der Dienst ist als Alias für den jeweiligen Dienst `sendFrom(<Sender>, to: <Receiver>, host: <Host>, user: <Username>, ...)` vorhanden. Er sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

sendHTML

Der Dienst versendet die E-Mail als HTML-Text. Die E-Mail muss vorher durch die o.g. Dienste beschrieben worden sein. Im Gegensatz zum Dienst `sendText` muss hier jedoch der zu sendende Text im HTML-Format vorliegen. D.h.: Mit dem Dienst `text` muss HTML formatierter Text übergeben worden sein. Das Ergebnis des Dienstes ist `True`, wenn die E-Mail versandt wurde, in allen anderen Fällen `False`.

sendText

Der Dienst versendet die Mail als plain Text. Die E-Mail muss vorher durch die o.g. Dienste beschrieben worden sein. Das Ergebnis des Dienstes ist `True`, wenn die E-Mail versandt wurde, in allen anderen Fällen `False`.

subject(<Variable>)

Mit <Variable> wird der Titel der E-Mail benannt.

ssl

Der Dienst meldet anhand eines Boolean-Wertes, ob der sichere SSL-Port für den E-Mailversand verwendet wird. Wenn der Dienst `ssl (<Variable>)` nicht benutzt wurde, ist das Ergebnis die allgemeine E-Maileinstellung (siehe Dokumentation Business-Process-Management, Kapitel *Einstellungen, Mail*).

ssl (<Variable>)

Wird `True` für <Variable> verwendet, wird der sichere SSL-Port für den E-Mailversand verwendet. Wird `False` für <Variable> verwendet, wird der ungesicherte Port für den E-Mailversand verwendet. Wird der Dienst nicht benutzt, wird anhand der allgemeinen E-Maileinstellungen (siehe Dokumentation Business-Process-Management, Kapitel *Einstellungen, Mail*) entschieden, ob der sichere SSL-Port für den E-Mailversand verwendet wird.

text(<Variable>)

Mit <Variable> wird der zu sendende Text der E-Mail benannt.

to (<Variable>)

Mit <Variable> wird der Empfänger der E-Mail benannt.

useIMAP

Der Dienst liefert `True` für die Verwendung eines IMAP-Mailservers und `False` für die Verwendung eines POP3-Mailservers. Siehe auch allgemeine Einstellung im Register **Mail** (siehe Dokumentation Business-Process-Management Kapitel *Register Mail*).

useIMAP(<Variable>)

Mit `True` für <Variable> wird abweichend von den allgemeinen Einstellung im Register **Mail** (siehe Dokumentation Business-Process-Management Kapitel *Register Mail*) ein IMAP-Mailserver für die weiteren Dienste verwendet. Der Dienst muss direkt nach der Erzeugung der Mail-Variable (`New Scripting.Mail`) erfolgen. Mit `False` für <Variable> wird ein POP3-Mailserver für die weiteren Dienste verwendet.

Hinweis: Bei Bedarf müssen mit den Diensten `host (<Variable>)`, `ssl (<Variable>)`, `user (<Variable>)`, `password (<Variable>)` und `authenticate(<Variable>)` die weiteren Parameter für die Arbeit mit dem Mailserver aus den allgemeinen Einstellungen übersteuert werden.

usePOP3

Der Dienst liefert `True` für die Verwendung eines POP3-Mailservers und `False` für die Verwendung eines IMAP-Mailservers. Siehe auch allgemeine Einstellung im Register **Mail** (siehe Dokumentation Business-Process-Management Kapitel *Register Mail*).

usePOP3(<Variable>)

Mit True für <Variable> wird abweichend von den allgemeine Einstellung im Register **Mail** (siehe Dokumentation Business-Process-Management Kapitel *Register Mail*) ein POP3-Mailserver für die weiteren Dienste verwendet. Der Dienst muss direkt nach der Erzeugung der Mail-Variable (New Scripting.Mail) erfolgen. Mit False für <Variable> wird ein IMAP-Mailserver für die weiteren Dienste verwendet.

Hinweis: Bei Bedarf müssen mit den Diensten *host* (<Variable>), *ssl* (<Variable>), *user* (<Variable>), *password* (<Variable>) und *authenticate*(<Variable>) die weiteren Parameter für die Arbeit mit dem Mailserver aus den allgemeinen Einstellungen übersteuert werden.

user (<Variable>)

Mit <Variable> wird der Benutzername auf dem mit *host* benannten Rechner angegeben. Der Absender der E-Mail ist unter dem Benutzernamen auf dem mit *host* angegebenen Rechner bekannt. Als Voreinstellung wird die Angabe **Benutzer** aus den allgemeinen OfficeTalk Einstellungen **Mail** verwendet.

username (<Variable>)

Der Dienst ist ein Alias für den Dienst *user*(. . .) und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

Besonderheiten

Das Attribut *replyTo*(<Variable>) wird, falls es nicht angegeben ist, mit dem Attribute *to* (<Variable>) besetzt. Die Attribute *from*(<Variable>), *to* (<Variable>), *host* (<Variable>) und *subject*(<Variable>) sind Pflichtangaben.

Beispiel

```
Dim mail As Mail
Dim fd As FileDialog
Dim sendNoFile As Boolean
Dim filename As String
fd = New Joops.Scripting.FileDialog
filename = fd.open
mail = New Joops.Scripting.Mail
mail.sender( "josef.springer@joops.com" )
mail.receiver( "Manuela.Springer@joops.com" )
mail.host( "joops2" )
mail.username( "jos@joops2" )
mail.password( "xyz" )
mail.subject( "test-ST" )
mail.text( "hallo" )
sendNoFile = IsEmpty( filename )
If sendNoFile = False
    Then mail.addAttachment( filename )
End If
```

```
If mail.send = True  
    Then MsgBox("Die Mail wurde versandt")  
    Else MsgBox("Die Mail konnte nicht versandt werden")  
End If
```


process

Anweisungen des Skripts können über diese globale Variable auf den laufenden Vorgang, in dem das Skript abläuft, zugreifen. Nachfolgend sind die wichtigsten Dienste des Vorgangs beschrieben. Die Namen der Dienste beginnen mit `process..`

Syntax

activeHistories

Der Dienst liefert die Historie aller laufenden Vorgänge, die aus der Vorgangsvorlage, von der dieser Dienst angefordert wird, erstellt wurden (siehe auch Kapitel *Beispiel-3*). Wenn bei der Makroausführung die Vorgangshistorie nicht eingeschaltet ist (siehe Dokumentation *Business-Process-Management*, Kapitel *Einstellungen, Aufgaben*), wird eine Ausnahme geworfen (siehe Kapitel *Behandlung fehlerhafter Argumente*).

activeHistory

Der Dienst liefert die Historie des laufenden Vorgangs (siehe auch Kapitel *Beispiel-3*). Wenn bei der Makroausführung die Vorgangshistorie nicht eingeschaltet ist (siehe Dokumentation *Business-Process-Management*, Kapitel *Einstellungen, Aufgaben*), wird eine Ausnahme geworfen (siehe Kapitel *Behandlung fehlerhafter Argumente*).

answer

Der Dienst liefert die mit dem Dienst `ask(...)` oder `askWorker(...)` zuletzt gestellte und beantwortete Frage. Der Dienst liefert als Ergebnis eine Systemvariable `Question` (siehe Kapitel *Question*) oder `Null`, wenn keine beantwortete Frage vorliegt.

answerFrom(<Variable>)

Der Dienst liefert die mit dem Dienst `ask(...)` oder `askWorker(...)` zuletzt gestellte Frage, die von der E-Mail Adresse <Variable> beantwortet wurde. Die E-Mail-Adressangabe in <Variable> ist nicht case sensitive. Der Dienst liefert als Ergebnis eine Systemvariable `Question` (siehe Kapitel *Question*). oder `Null`, wenn keine von <Variable> beantwortete Frage vorliegt.

answers

Der Dienst liefert alle mit dem Dienst `ask(<Adresse>, subject: <Betreff>, question: <Text>, attachments: <Anhänge>)` oder `askWorker(<Bearbeiter>, subject: <Betreff>, question: <Text>, attachments: <Anhänge>)` gestellten, und bisher beantworteten Fragen. Der Dienst liefert als Ergebnis ein Array von Systemvariablen `Question`. Wenn keine Antworten vorliegen, ist das Ergebnis des Dienstes ein leeres Array. Das Beispiel ermittelt und gibt die Antwort der zuletzt gestellten Frage aus:

```
Dim beantworteteFragen As Array
Dim frage As Question
beantworteteFragen = process answers
If UBound(beantworteteFragen) > 0
    Then frage = beantworteteFragen(UBound(beantworteteFragen) - 1)
    MsgBox("Die Antwort der zuletzt beantworteten Frage: " &
        frage.answer)
End If
```

ask(<Adresse>, subject: <Betreff>, question: <Text>, attachments: <Anhänge>)

Mit dem Dienst wird an den Empfänger mit der E-Mail-Adresse <Adresse> eine E-Mail mit dem <Betreff> und dem Inhalt <Text> gesendet. Die Anhänge in dem Array <An-

`hänge`> werden an die E-Mail angehängt. Sollen keine Anhänge versandt werden, muss hier ein leeres `Array` oder `Null` als Argument verwendet werden. Der Vorgang wird dadurch in eine Warteschleife versetzt. Erst wenn die Antwort auf die E-Mail eingegangen ist, kann er weitergeführt werden. Vorgänge, die sich im Wartezustand befinden, werden in der Aufgabenliste mit *kursivem Namen* angezeigt. Wenn ein solcher Vorgang bearbeitet oder beendet werden soll, erscheint zuerst ein Dialog mit den eingegangenen und ausstehenden E-Mails. Nur wenn der Dialog mit **Ausführen** beantwortet wird, kann der Vorgang bearbeitet werden. Es sind auch mehrere `ask(<Adresse>, subject: <Betreff>, question: <Text>, attachments: <Anhänge>)` Anweisungen für einen einzelnen Vorgang erlaubt.

`ask(<Adresse>, subject: <Betreff>, question: <Text>,)`

Siehe Dienst `ask(<Adresse>, subject: <Betreff>, question: <Text>, attachments: <Anhänge>)`.

`askWorker(<Bearbeiter>, subject: <Betreff>, question: <Text>, attachments: <Anhänge>)`

Inhaltlich gilt das selbe, wie beim zuvor beschriebenen Dienst `ask(...)`. Mit Ausnahme des ersten Arguments `<Bearbeiter>`. Hier geben Sie den Bearbeiter, an den die E-Mail gesendet werden soll, an. Dieser Bearbeiter muss eine E-Mail-Adresse enthalten !

`askWorker(<Bearbeiter>, subject: <Betreff>, question: <Text>)`

Siehe Dienst `askWorker(<Bearbeiter>, subject: <Betreff>, question: <Text>, attachments: <Anhänge>)`.

`asynchronWaitFor(<Vorgang>)`

Mit dem Dienst wird der Vorgang bei dem Arbeitsschritt, der mit der Option **Beendigung gestarteter Vorgänge abwarten** versehen ist, angehalten. D.h. Der Vorgang kann weiter bis zu diesem Arbeitsschritt ausgeführt werden. Wenn dieser Arbeitsschrittes zur Ausführung an der Reihe ist, geht der Vorgang automatisch in den Wartezustand. Erst wenn der angegebene Vorgang `<Vorgang>` beendet ist, kann er weitergeführt werden. Vorgänge, die sich im Wartezustand befinden, werden in der Aufgabenliste mit *kursivem Namen* angezeigt. Um einen solchen Vorgang trotzdem zu bearbeiten, muss der Bearbeiter das Vorgangsrechte **warten ignorieren** (siehe Kapitel Business-Process-Management, Kapitel *Rechte*) besitzen. Ist das der Fall, erscheint zuerst einem Hinweisdialog auf den oder die wartenden Vorgänge. Nur wenn dieser Hinweis mit **Ja** beantwortet wird, wird der Vorgang auch bearbeitet. Mehrere `asynchronWaitFor(...)` Anweisungen für einen einzelnen Vorgang sind erlaubt.

Hinweis: Nach dem Abschluss des aktuellen Arbeitsschrittes wird der Vorgang, auf dessen Beendigung gewartet wird, automatisch ausgeführt, wenn die dazu erforderlichen Bedingungen erfüllt sind. Siehe Dokumentation Workflow, Kapitel *Automatischer Start der Ausführung*.

Beispiel

Dim vorgang As Process

```
vorgang = Start "Akquisition" "Kunde"
```

```
vorgang.processdata.addItem ( "Receiver", with: "max.huber@xxx.de",  
                                in: "Mail" )
```

```
vorgang.asynchronWaitFor( gestartet, comment: "Das ist der  
                                Wartekommentar" )
```

asynchronWaitFor(<Vorgang>, comment: <Zeichenkette>)

Funktionalität siehe oben. Zusätzlich wird mit <Zeichenkette> ein Kommentar mit gegeben.

asynchronWaitings

Der Dienst liefert in einem `Array` alle Vorgänge, die mit `asynchronWaitFor(<Vorgang>, comment: <Zeichenkette>)` angewiesen wurden, zu warten.

currentStep

Der Dienst liefert den aktuell ausgeführten Arbeitsschritt eines Vorgangs, oder den als nächstes auszuführenden Arbeitsschritt eines Vorgangs. Der Dienst ist für Sonderfälle vorgesehen.

description

Der Dienst liefert die Beschreibung des Vorgangs.

description(<Zeichenkette>)

Mit dem Dienst belegen Sie die Beschreibung dieses Vorgangs mit <Zeichenkette>.

executionColor

Der Dienst liefert die Ampelfarbe des Vorgangs als String. Green für Vorgänge deren Bearbeitungszeit im vorgegebenen Zeitrahmen liegt. Yellow deren Bearbeitungszeit geringfügig überschritten ist, und red, wenn die Bearbeitungszeit beträchtlich überschritten ist.

finishedHistories

Der Dienst liefert die Historien aller abgeschlossenen Vorgänge, die aus der Vorgangsvorlage, von der dieser Dienst angefordert wird, erstellt wurden (siehe auch Kapitel *Beispiel-3*). Wenn bei der Makroausführung die Vorgangshistorie nicht eingeschaltet ist (siehe Dokumentation Business-Process-Management, Kapitel *Einstellungen, Aufgaben*), wird eine Ausnahme geworfen (siehe Kapitel *Behandlung fehlerhafter Argumente*).

goTo(<Bearbeiter>)

Mit dem Dienst delegieren Sie die Bearbeitung des Vorgangs an den Bearbeiter <Bearbeiter>. Mögliche Bearbeiter erhalten Sie u.a. mit dem Dienst `workers` oder `substitutes`. <Bearbeiter> kann ein passiver (Abteilung, Büro, usw.), oder ein aktiver Bearbeiter (Schreibtisch, Maschine) sein. <Bearbeiter> muss jedoch die erforderlichen Vorgangsrechte besitzen. Einzelheiten zu den Vorgangsrechten lesen Sie in der Dokumentation Business-Process-Management, Kapitel *Rechte*. Der Dienst beendet mit `True`, wenn die Delegation ausgeführt wurde, ansonsten mit `False`.

Hinweis: Mit dem Dienst kann nicht der aktuelle Vorgang delegiert werden. Um den aktuellen Vorgang zu delegieren siehe Kapitel `goTo(<Bearbeiter>)`.

goTo(<Bearbeiter>, comment: <Kommentar>)

In der Historie und in der Delegationsbenachrichtigung per E-Mail wird der angegebene Kommentartext eingetragen. Die weiteren Einzelheiten zum Dienst finden Sie im Kapitel `goTo(<Bearbeiter>)`.

identity

Der Dienst liefert die eindeutige Identität des Vorgangs in der Datenbank als `Integer`.

lock

Der Dienst sperrt den Vorgang gegen Zugriff durch andere Bearbeiter. Das Ergebnis des Dienstes ist bei erfolgreicher Sperrung `True`, andernfalls `False`. Der Dienst ist nicht für Vorgänge der eigenen Aufgabenliste erlaubt, und sollte mit Bedacht verwendet werden, da Sie damit die Bearbeitung des Vorgangs durch seinen Eigentümer behindern ! Befindet sich der Vorgang in der Ausführung, liefert der Dienst als Ergebnis `False`. Der Sperre des Vorgangs wird beim Ende des laufenden Arbeitsschrittes aufgehoben.

Hinweis: Da der der Vorgang, solange er gesperrt ist, in der Aufgabenliste seines Bearbeiters nicht ausgeführt werden kann, sollte der Arbeitsschritt möglichst schnell beendet werden, um eine mögliche Ausführung durch seinen Eigentümer nicht unnötig lange zu verzögern !

logicalName

Der Dienst liefert den zusätzlich vergebenen Vorgangsnamen. Wenn ein neuer Vorgang gestartet wird, kann dieser Name über eine Dialogbox eingegeben werden.

logicalName(<Name>)

Mit dem Dienst kann der zusätzliche Vorgangsname vergeben werden. <Name> ist dabei der zu vergebende Name.

maxCharLogicalName

Der Dienst liefert die maximale Zeichenanzahl des zusätzlichen Vorgangsnamens.

minMaxDueDate

Der Dienst liefert in einem `Array` das geplante früheste und das geplante späteste Endedatum des Vorgangs. Ein Beispiel:

```
theDueDates = process.minMaxDueDate
```

```
MsgBox("Frühestes Endedatum: " & theDueDates(0).date & " - " &  
theDueDates(0).time & " Spätestes Endedatum: " & theDueDates(1).date &  
" - " & theDueDates(1).time)
```

Hinweis: Bei einem Vorgang aus der Aufgabenliste wird für die Berechnung das Startdatum des aktuellen Arbeitsschrittes verwendet. Bei einer Vorgangsvorlage wird für die Berechnung das aktuelle Datum und Uhrzeit als Startdatum des ersten Arbeitsschrittes angenommen.

Bei der Berechnung des Endedatums wird ein Arbeitstag mit 8 Stunden angenommen. D.h. Die geplante Bearbeitungszeit eines Arbeitsschrittes wird auf einen Kalendertag mit 24 Stunden hochgerechnet. Die Wartezeit eines Arbeitsschrittergebnisses hingegen wird als Zeitraum auf Basis von 24 Stunden verwendet.

Für die Berechnung eines Endedatums werden Ablaufpfade, die im Register **Arbeitsschrittergebnisse** die Option **Keine Berechnung der Zeiten für nachfolgende Arbeitsschritte** gesetzt haben, nicht berücksichtigt (siehe Business-Process-Management, Kapitel *Arbeitsschrittergebnis*).

minMaxDueDate(<Pfadbegrenzung>, stopTimes: <Array>)

Der Dienst liefert in einem `Array` das geplante früheste und das geplante späteste Enddatum des Vorgangs. Mit einer Zahl für das Argument `<Pfadbegrenzung>` wird die zu berücksichtigende Anzahl der Ablauffpade bestimmt. Bei der Angabe `Null` oder `0` wird die Einstellung von OfficeTalk verwendet. Mit dem Argument `stopTime:` kann bestimmt werden, welche oder welche Teile eines Ablauffpades ab welchem Arbeitsschritt nicht berücksichtigt werden soll. Das Format der Arguments:

- `Null`: Die Angaben in den Arbeitsschrittergebnissen des Vorgangs werden verwendet.
- `Array()`: Die Angaben in den Arbeitsschrittergebnissen des Vorgangs werden ignoriert.
- `Array(Array(<Arbeitsschrittergebnisname>, <Arbeitsschrittname>), ...)`

`<Arbeitsschrittergebnisname>`: `String`

`<Arbeitsschrittname>`: `<Kategorie>-<Bezeichnung>`

`<Kategorie>`: `String`

`<Bezeichnung>`: `String`

Für die generelle Beschreibung siehe Dienst *minMaxDueDate*.

priority

Der Dienst liefert die Priorität des Vorgangs als Zahl.

processByIdentity(<Identität>)

Der Dienst liefert den Vorgang mit der angegebenen Identität (siehe Kapitel *identity*). Der Vorgang kann auch aus der Aufgabenliste eines anderen Bearbeiters stammen. Deshalb dürfen bei diesem Vorgang nur abfragende Dienste verwendet werden !

Hinweis: Falls erforderlich, wird der Inhalt des gefundenen Vorgangs mit seinen Daten aus der Datenbank aktualisiert (siehe Kapitel *refresh*).

processByID(<Identität>)

Siehe *processByIdentity(<Identität>)*. Der Dienst wird in der nächsten Hauptversion entfernt.

processData

Der Dienst liefert die permanenten Daten des Vorgangs. Von diesen Vorgangsdaten können Sie weitere Dienste ausführen lassen (siehe Kapitel *processdata*) oder Sie können diese Vorgangsdaten einem gestarteten Vorgang zur weiteren Bearbeitung übergeben (siehe Kapitel *processdata*).

processData({ <Vorgang> | <Vorgangsdaten> })

Mit dem Dienst übergeben Sie die Vorgangsdaten eines Vorgangs an einen anderen Vorgang. Als Argument können ein `<Vorgang>` vom Typ `process` oder die `<Vorgangsdaten>` eines Vorgangs vom Typ `processdata` verwendet werden. Der Dienst ist für die Übergabe der aktuellen Vorgangsdaten an einen gestarteten Vorgang hilfreich (siehe Kapitel *processdata*). Die Vorgangsdaten werden als Kopie übergeben.

refresh

Der Dienst aktualisiert den Inhalt des Vorgangs mit seinem Datenbankinhalt. Der Dienst sollte nur für Vorgänge anderer Bearbeiter verwendet werden und keinesfalls für Vorgänge des angemeldeten Bearbeiters !

scheduleAt(<Datum>, time: <Zeit>, comment: <Kommentar>)

Mit dem Dienst kann die Startzeit des nächsten Arbeitsschrittes eines Vorgangs geändert werden. Der Dienst liefert das Ergebnis der Änderung als Boolean. <Datum> ist das Startdatum, <Zeit> ist die Startzeit und <Kommentar> ist eine optionale Zeichenkette, mit der die Änderung erklärt wird. Diese Erklärung findet sich auch in der Historie wieder. Der Dienst ist für Sonderfälle, in denen z.B. die Startzeit geändert werden soll, vorgesehen. Der ändernde Bearbeiter muss die dafür erforderlichen Rechte (siehe Dokumentation Business-Process-Management Kapitel *Rechte*) besitzen !

Hinweis: Der Dienst darf keinesfalls für den aktuell ausgeführten Vorgang verwendet werden. Er darf auch nicht für einen Vorgang, dessen Bearbeiter in OfficeTalk angemeldet ist, verwendet werden. Die erste Bedingung wird durch den Scheduler geprüft. Die zweite Bedingung muss durch das Makro selbst geprüft werden !

Wird der Dienstes mit einem falschen Argument oder fehlenden Rechten verwendet, wird eine Ausnahme geworfen (siehe Kapitel *Behandlung fehlerhafter Argumente*).

Ein Beispiel:

```
Reschedule( vorgang As Process, eigentümer As Worker )
Dim name As String

name = eigentümer.displayString
If eigentümer.isLoggedIn = False
Then
    Try
        vorgang.scheduleAt(Date + 1, time: Time, comment:
            "Die Ausführung wurde von " & name & " vorgezogen" )
    End Try
    Catch
        MsgBox("Der Vorgang konnte nicht reterminiert werden:<n>" &
            Error.Description)
    End Catch
End If
```

synchronWaitFor(<Vorgang>)

Mit dem Dienst wird der Vorgang nach Abschluss des laufenden Arbeitsschrittes angehalten. Erst wenn der angegebene Vorgang <Vorgang> beendet ist, kann er weitergeführt werden. Vorgänge, die sich im Wartezustand befinden, werden in der Aufgabenliste mit *kursivem Namen* angezeigt. Um einen solchen Vorgang trotzdem zu bearbeiten, muss der Bearbeiter das Vorgangsrechte **warten ignorieren** (siehe Business-Process-Management, Kapitel *Rechte*) besitzen. Ist das der Fall, erscheint zuerst einem Hinweisdialog auf den oder die wartenden Vorgänge. Nur wenn dieser Hinweis mit **Ja** beantwortet wird, wird der Vorgang auch bearbeitet. Mehrere *synchronWaitFor(<Vorgang>)* Anweisungen für einen einzelnen Vorgang sind erlaubt.

Hinweis: Nach dem Abschluss des aktuellen Arbeitsschrittes wird der Vorgang, auf dessen Beendigung gewartet wird, automatisch ausgeführt, wenn die dazu erforderlichen Bedingungen erfüllt sind. Siehe Dokumentation Workflow, Kapitel *Automatischer Start der Ausführung*.

Beispiel

```
Dim vorgang As Process
```

```
vorgang = Start "Akquisition" "Kunde"
```

```
vorgang.processdata.addItem ( "Receiver", with: "max.huber@xxx.de",  
                                in: "Mail" )
```

```
vorgang.synchronWaitFor( gestartet, comment: "Das ist der  
                                Wartekommentar" )
```

synchronWaitFor(<Vorgang>, comment: <Zeichenkette>)

Funktionalität siehe oben. Zusätzlich wird mit <Zeichenkette> ein Kommentar mit gegeben. Dieser Kommentar ist später in der Historie ersichtlich.

synchronWaitings

Der Dienst liefert in einem Array alle Vorgänge, die mit *synchronWaitFor*(<Vorgang>) angewiesen wurden, zu warten.

waitFor(<Vorgang>)

Siehe Kapitel *synchronWaitFor*(<Vorgang>).

waitFor(<Vorgang>, comment: <Zeichenkette>)

Siehe Kapitel *synchronWaitFor*(<Vorgang>, comment: <Zeichenkette>).

waitingProcesses

Der Dienst liefert in einem Array alle Vorgänge, die auf die Beendigung des Vorgangs warten.

processdata

Jedem Vorgang können permanente Daten angehängt werden. Diese Daten können in Rahmen eines Makros erstellt und abgeholt werden. Permanente Vorgangsdaten sind eine flexible Möglichkeit, einen Vorgang allgemein zu beschreiben, aber konkret auszuführen. Die Variable ist global deklariert, und ist somit im Makro immer verfügbar. Die Namen der Dienste beginnen mit `processdata..`

Syntax

add(<Gruppe>)

Der Dienst fügt die Eintragsgruppe mit dem Namen <Gruppe> ein.

addEntry(<Gruppe>)

Der Dienst ist ein Alias für den Dienst `add(<Gruppe>)`, und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

addItem(<Eintrag>, with: <Datum>, inEntry: <Gruppe>)

Der Dienst ist ein Alias für den Dienst `item(<Eintrag>, in: <Gruppe>)` und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

copy(<Gruppe>, from: <Vorgangsdaten>)

Der Dienst kopiert alle Einträge der Vorgangsdatengruppe, benannt durch das Argument <Gruppe>, aus den Vorgangsdaten eines Vorgangs, benannt durch <Vorgangsdaten>, in die Vorgangsdaten eines anderen Vorgangs. <Vorgangsdaten> muss vom Typ `processdata` sein. Der Dienst ist hilfreich, wenn einem neu gestarteten Vorgang Vorgangsdaten des laufenden Vorgangs mitgegeben werden sollen (siehe *Beispiel-2*). Die Vorgangsdaten werden als Kopie übergeben. Der Dienst antwortet mit `True` wenn mindestens ein Eintrag kopiert wurde, ansonsten mit `False`.

copyEntry(<Gruppe>, from: <Vorgangsdaten>)

Der Dienst ist ein Alias für den Dienst `copy(<Gruppe>, from: <Vorgangsdaten>)` und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

copy (<FromGruppe>, to: <ToGruppe>)

Der Dienst kopiert alle Einträge aus der Gruppe <FromGruppe> in die Gruppe <ToGruppe>. Bereits vorhandene gleichnamige Einträge in <ToGruppe> werden überschrieben. Die Vorgangsdaten werden als Kopie übergeben. Der Dienst antwortet mit `True` wenn mindestens ein Eintrag kopiert wurde, ansonsten mit `False`.

entries

Der Dienst liefert die Namen aller Gruppen in den Vorgangsdaten als `Array`. Ein Anwendungsbeispiel dazu finden Sie im Kapitel *Beispiel-3*.

in(<Gruppe>)

Der Dienst liefert alle Einträge unter den Gruppennamen <Gruppe> als Name/Wert-Paar in einem `Array`. Jedes Name/Wert-Paar ist in einem `Array` verpackt.

item(<Eintrag>)

Der Dienst liefert das Datum das unter dem Eintragsnamen <Eintrag> für den aktuellen Vorgang abgelegt wurde. Der Dienst liefert den ersten gefundenen Eintrag aus allen Gruppen. Deshalb ist die Verwendung dieses Dienstes nur dann sinnvoll, wenn der Eintragsname <Eintrag> über alle Gruppen hinweg eindeutig ist. Die Variable, die das Datum aufnehmen soll, muss mit einem übereinstimmenden Typ deklariert werden.

Der Dienst erleichtert die Erstellung eines Makros für mehrer Aufgabengebiete. Z.B. Ein Makro soll für die Aufgabe **Kundenname prüfen** und für die Aufgabe **Interessentennamen prüfen** eingesetzt werden. Beide Aufgabengebiete verwenden unterschiedliche Vorgangsdatengruppen, aber den selben Eintragsnamen für den Kunden- und Interessentennamen. Deshalb liefert der Dienst `processdata.item("name")` immer den Namen.

item(<Eintrag>, in: <Gruppe>)

Der Dienst liefert das Datum das unter den Gruppennamen <Gruppe> und dem Eintragsnamen <Eintrag> für den aktuellen Vorgang abgelegt wurde. Die Variable, die das Datum aufnehmen soll, muss mit einem übereinstimmenden Typ deklariert werden.

items(<Eintragsnamen>, in: >Gruppe>)

Der Dienst liefert die Datums der im Array <Eintragsnamen> angegebenen Eintragsnamen unter den Gruppennamen <Gruppe>, die für den aktuellen Vorgang abgelegt wurden. Enthält <Gruppe> Wildcards (? oder *), werden die Eintragsnamen aller passenden Gruppen geliefert. Das Ergebnis des Dienstes ist

Array (<Eintrag>, ...)

<Eintrag>: Array(<Gruppe>, <Datums>)

<Gruppe>: Der Gruppenname

<Datums>: Array(<Wert>)

<Wert>: Der Eintragswert

itemsIn(<Gruppe>)

Der Dienst liefert die Einträge des angegebenen Gruppennamens <Gruppe>, die für den aktuellen Vorgang abgelegt wurden. Enthält <Gruppe> Wildcards (? oder *), werden die Einträge der passenden Gruppen nach dem Gruppennamen aufsteigend sortiert geliefert. Der Dienst blendet im Ergebnis leere Gruppen und leere Einträge in Gruppen aus. Ein Eintrag wird als leer interpretiert, wenn er einen leeren String oder Null enthält. Das Ergebnis des Dienstes ist

Array (<Gruppe>, ...)

<Gruppe>: Array(<Gruppenname>, <Datums>)

<Gruppenname>: Der Gruppenname

<Datums>: Array(<Eintragsname>, <Wert>)

<Eintragsname>: Der Eintragsname

<Wert>: Der Eintragswert

item(<Eintrag>, inEntry: <Gruppe>)

Der Dienst ist ein Alias für den Dienst `item(<Eintrag>, in: <Gruppe>)` und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

item(<Eintrag>, with: <Datum>, in: <Gruppe>)

Der Dienst fügt den Vorgangsdaten einen Eintrag hinzu. Dabei bezeichnet <Gruppe> die Gruppenbezeichnung und <Eintrag> den Datenbezeichner unter dem <Datum> abgelegt wird. <Datum> muss vom Datentyp String, Date, Time, Character, Integer, Long, Float, Double oder Boolean sein.

item(<Eintrag>, with: <Datum>, inEntry: <Gruppe>)

Der Dienst ist ein Alias für den Dienst *item(<Eintrag>, with: <Datum>, in: <Gruppe>)* und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

namesIn(<Gruppe>)

Der Dienst liefert die Namen aller Einträge in der Gruppe mit dem Namen <Gruppe> als Array.

maxCharString

Der Dienst liefert die maximale Anzahl der Zeichen, die eine Stringvariable enthalten darf. Eine Variable mit mehr Zeichen wird beim Speichern (*processdata.addItem(...)*) automatisch und ohne Warnung abgeschnitten. Sie können im *ScriptDialog* die Eingabelänge mit dem Dienst *limit (<NameOderArrayMitNamen>, maxChar: <Variable>)* begrenzen.

refresh

Der Dienst aktualisiert den Inhalt der Vorgangsdaten mit ihrem Datenbankinhalt. Der Dienst ist erforderlich, wenn andere Applikationen die Vorgangsdaten geändert haben.

remove (<Eintrag>, in: <Gruppe>)

Der Dienst entfernt den mit <Eintrag> benannten Eintrag in der mit <Gruppe> benannten Eintragsgruppe.

remove(<Gruppe>)

Der Dienst entfernt alle Einträge in der mit <Gruppe> benannten Eintragsgruppe.

removeAll

Der Dienst entfernt alle Gruppen in den Vorgangsdaten.

removeEntry(<Gruppe>)

Der Dienst ist ein Alias für den Dienst *remove(<Gruppe>)* und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

removeItem(<Eintrag>, inEntry: <Gruppe>)

Der Dienst ist ein Alias für den Dienst *remove (<Eintrag>, in: <Gruppe>)* und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

valuesIn(<Gruppe>)

Der Dienst liefert die Werte aller Einträge in der Gruppe mit dem Namen <Gruppe> als Array.

Beispiel-1

```
Dim vorname As String
```

```
Dim nachname As String
Dim umsatz As Double
vorname = InputBox("bitte Vorname eingeben" )
nachname = InputBox("bitte Nachname eingeben" )
umsatz = InputBox("bitte Umsatz eingeben" )
processdata.item ( "UMS" with: vorname in: "Vorname")
processdata.item ( "UMS"  with: nachname in: "Nachname")
processdata.item ( "UMS"  with: umsatz in: "Umsatz")
```

In einem späteren Arbeitsschritt des selben Vorgangs können die zuvor eingegebenen Daten weiter verarbeitet werden.

```
Dim vorname As String
Dim nachname As String
Dim umsatz As Double
vorname = processdata.item ( "UMS", in: "Vorname")
nachname = processdata.item ( "UMS", in: "Nachname")
umsatz = processdata.item ( "UMS", in: "Umsatz")
MsgBox( Array (vorname, " ", nachname. " hat Umsatz: ", umsatz )
```

Beispiel-2

```
Dim vorgang As Process

Try
    vorgang = Start "Akquisition" "Kunde"
    vorgang.processdata.copy ("Kunde", from: processdata)
End Try
Catch
End Catch
```

Beispiel-3

```
VorgangsdatenÜbernehmenVon(vorgang As Process)
'Das Makro übernimmt alle Vorgangsdaten aus vorgang
Dim i As Integer
Dim alleGruppennamen As Array

alleGruppennamen = vorgang.entries
For i = 1 To UBound( alleGruppennamen ) Step 1
    processdata.copy( alleGruppennamen(i - 1), from: vorgang.processdata)
Next
```

ProcessHistory

Eine Vorgangshistorie beinhaltet Informationen zu ausgeführten, oder in Bearbeitung befindlichen Vorgängen. Der Dienst `finishedHistories` der Systemvariablen **Process** liefert die Historien aller abgeschlossenen Vorgänge einer Vorgangsvorlage. Der Dienst `activeHistories` der Systemvariablen **Process** liefert die Historien aller laufenden Vorgänge einer Vorgangsvorlage. Der Dienst `activeHistory` der Systemvariablen **Process** liefert die Historie dieses laufenden Vorgangs.

Syntax

abort

Der Dienst liefert `True`, wenn der Vorgang abgebrochen wurde.

comment

Der Dienst liefert den im Skriptmakro erzeugten Kommentar zu dem Vorgang. Siehe Dienst `comment(<Zeichenkette for: <Vorgangselement>)`.

consumed

Der Dienst liefert die Kosten der verbrauchten Ressourcen des Vorgangs.

description

Der Dienst liefert die Beschreibung des Vorgangs.

endedAt

Der Dienst liefert Endedatum und Uhrzeit des Vorgangs.

startedAt

Der Dienst liefert Startdatum und Uhrzeit des Vorgangs.

starter

Der Dienst liefert den Namen des Bearbeiters, der den Vorgang gestartet hat.

steps

Der Dienst liefert die Historien der ausgeführten Arbeitsschritte des Vorgangs.

logicalName

Der Dienst liefert den zusätzlichen Namen des Vorgangs.

processData

Der Dienst liefert die Daten (siehe Kapitel *processdata*), die bei dieser Ausführung des Vorgangs verwendet wurden. Die Daten können mit den Diensten der Systemvariablen *processdata* ausgewertet werden.

Beispiel

```
`Die Gesamtsumme der Ausführungen eines fiktiven Vertriebsvorgangs
`wird ermittelt. Dabei wird hier davon ausgegangen, dass die erste
`Vorgangsvorlage der sichtbaren Vorgangsvorlage der Vertriebsvorgang
`ist und darin der Umsatz in den Vorgangsdaten"Umsatz" / "Auftrag"
`erfasst wird.
```

```
Dim prozesse As Array
Dim historien As Array
Dim i As Integer
Dim summe As Integer

summe = 0
prozesse = stepscheduler.processTemplates
historien = prozesse(0).finishedHistories
For i = 0 To Ubound(historien) Step 1
    historie = historien(i-1)
    summe = summe + historie.processData.item("Umsatz", in: "Auftrag")
Next
MsgBox("Die gesamtsumme der Aufträge beträgt: ", & summe)
```

Question

Bei der Bearbeitung eines Vorgangs können Fragen an eine beliebige E-Mail-Adresse geschickt werden. Die Dienste *answerFrom*(<Variable>), *answer* und *answers* der Systemvariablen *process* liefern jeweils ein oder mehrerer Systemvariablen vom Typ *Question*. Diese Fragen enthalten auch die Antwort darauf.

Syntax

answer

Der Dienst liefert den Antworttext.

asked

Der Dienst liefert die E-Mail-Adresse des Empfängers.

asker

Der Dienst liefert die E-Mail-Adresse des Absenders.

attachments

Der Dienst liefert die Dateinamen der Anlagen zur Antwort. Die Anlagen werden als Dateien im Ordner für Anlagen abgelegt. Zur Bestimmung des Ordners lesen Sie bitte das Kapitel *Einstellungen – Register Mail* in der Dokumentation [Business-Process-Management](#).

question

Der Dienst liefert den Fragetext.

subject

Der Dienst liefert den Betreff der Frage.

Beispiel

```
Dim frage As Joops.OfficeTalk.Question
frage = process.answer 'liefert die zuletzt eingetroffene Antwort
MsgBox("Die Frage lautet: ", & frage.question)
MsgBox("Die Antwort lautet: ", & frage.answer)
```

Resource

In Vorgangsdialogen kann es sinnvoll sein, Informationen einer verwendeten Ressource anzuzeigen. Zu diesem Zweck kann die Systemvariable `Resource` verwendet werden. In diesem Zusammenhang lesen Sie auch die Kapitel *resource(<Name>)* und *resources*.

Syntax

description

Der Dienst liefert die Beschreibung der Ressource als `String`.

displayName

Der Dienst liefert den Namen der Ressource als `String`.

consumed(<Menge>)

Der Dienst liefert die Kosten für die `<Menge>` verbrauchter Ressourcen als `Double`.
`<Menge>` ist ein numerischer Datentyp (`Integer`, `Long`, `Float` oder `Double`) oder ein `String`, der einen numerischen Wert darstellt.

cost

Der Dienst liefert die Kosten pro Einheit der Ressource als `Integer`.

unit

Der Dienst liefert die Einheit der Ressource als `String`.

calculationBase

Der Dienst liefert die Kalkulationsbasis der Ressource als `String`.

faktor

Der Dienst liefert den Mengenfaktor der Ressource als `Integer`.

scheduledata

Bei jedem Vorgang können temporäre Vorgangsdaten verwendet werden. Diese Daten können in Rahmen einer Makroausführung erstellt und für beliebig Anforderungen benutzt werden. Im Gegensatz zu den permanenten Vorgangsdaten (siehe Kapitel *processdata*) sind diese Vorgangsdaten nur verfügbar, solange die Aufgabenliste nicht geschlossen wird. Auch bei einer Delegation des Vorgangs verschwinden diese Vorgangsdaten, weil sie nicht in der Datenbank gespeichert werden. Mit Ausnahme der Dienste `refresh` und `maxCharString` unterstützen die temporären Vorgangsdaten die selben Dienste, wie die permanenten Vorgangsdaten. Temporäre Vorgangsdaten sind eine flexible Möglichkeit, Daten während des Ablaufes eines Vorgangs kurzzeitig für temporäre Zwecke zu nutzen. Die Variable ist global deklariert, und ist somit im Makro immer verfügbar. Die Namen der Dienste beginnen mit `scheduledata..`

Hinweis: Die Limitierung der maximalen Größe von Zeichenketten aus den permanenten Vorgangsdaten (siehe Kapitel *maxCharString*) entfällt.

Syntax

Dienste und Syntax der Dienste siehe Kapitel *processdata*.

SystemData

Mit der Systemvariablen `SystemData` werden, wie mit der Systemvariablen `ProcessData` und `ScheduleData`, Vorgangsdaten verwaltet. Jedoch im Gegensatz zu den Systemvariablen `ProcessData` und `ScheduleData` sind die Daten in `SystemData` vorgangsübergreifend und systemweit verfügbar. Die Daten in `SystemData` sind ab dem Einstellzeitpunkt bis zum Systemende für jeden Vorgangsablauf erreichbar. Die Systemvariable `SystemData` stellt also einen zentralen Schreib-/Lese-Korb von Daten für jeden Vorgangsablauf zur Verfügung.

Besonders Vorgangslandschaften, die eine gemeinsame Datenbasis verwenden, profitieren davon. Vorgangsabläufe können mit der Verwendung der Systemvariablen `SystemData` beschleunigt werden, weil diese Datenstrukturen nur mehr einmal erstellt werden müssen, aber viele Male verwenden können.

Zur Erstverwendung der Systemvariablen `SystemData` muss die Kommunikationsbibliothek **OfficeTalk SystemDate.pcl** geladen werden.

Syntax

Dienste und Syntax der Dienste siehe Kapitel *processdata*.

process(<Vorgang>)

Mit der Anweisung `New SystemData` wird den Vorgangsdaten der aktuelle Vorgang für die weitere Bearbeitung zugewiesen. In Sonderfällen kann mit dem Dienst ein abweichender Vorgang für die weitere Bearbeitung zugewiesen.

Beispiel

Schreiben

```
'Diese Makro schreibt das Datum in die systemweiten Vorgangsdaten
Library "..\Library\OfficeTalk SystemData.pcl"

Dim daten As SystemData
daten = New SystemData
daten.item("eintrag", with: "Kartennamen", in: "gruppe")
```


Lesen

'Diese Makro wird im Ablauf eines anderen Vorgangs verwendet und
'liest das Datum das zuvor im einem anderen Vorgang im Makro Schreiben
'geschrieben wurde

```
Dim daten As SystemData
```

```
Dim datum As String
```

```
daten = New SystemData
```

```
datum = daten.item("eintrag", in: "gruppe")
```

ScriptDialog

Mit Variablen vom Typ `ScriptDialog` können Dialoge erstellt werden. `ScriptDialog` kennt Dienste, für das Einfügen von Eingabefeldern, Buttons, Listen und Checkboxes. Ebenso können darin Daten angezeigt und abgeholt werden. Die Variable muss im Deklarationsteil mit

```
„Dim <Variable> As Joops.Scripting.ScriptDialog“
```

deklariert werden. Die Position der Felder können Sie selbst bestimmen oder vom integrierten Layoutmanager nach vorgegebenen Regeln bestimmen lassen. Die Dienste ohne Positionsangaben, deren Namen mit `add` beginnt, sind für den Layoutmanager vorgesehen. Mit den `add`-Diensten mit Positionsangaben bestimmen Sie selbst die Position des Feldes. Nach dem Sie die Felder dem Layoutmanager übergeben haben, weisen Sie ihn mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` an, den Dialog ein- oder mehrspaltig orientiert zu erstellen. Anschließend öffnen Sie den Dialog mit dem Dienst `open` und übergeben ihn dem Benutzer zur Bearbeitung.

Syntax

Die Dienste gliedern sich in allgemeine Dienste und Dienste zum Einfügen von Dialogelementen für die Texteingabe, Auswahl oder für das Auslösen von Aktionen.

```
Dim dialog As Joops.Scripting. ScriptDialog
```

Allgemeine Dienste

Die Dienste in diesem Kapitel wirken auf Dialog selbst. Dienste für die einzelnen Felder des Dialoges sind in den folgenden Kapiteln beschrieben.

beModeless

Durch Verwendung des Dienstes wird der Dialog, im Gegensatz zum Standard nicht modal geöffnet. Der Dienst muss als erstes nach der Instantiierung durch `New` (z.B. `dialog = New Joops,Scripting.ScriptDialog`) erfolgen ! Der Dienst `open` öffnet den Dialog und kehrt sofort zurück. Folgende Einschränkungen gelten für nicht modale Dialoge:

- Die Dialoge sind nur für Anzeige- und Protokollierungszwecke geeignet.
- Benutzerinteraktion sind nicht möglich.
- In Aktionsblöcken der Dialogelemente *Aktionsschaltfläche*, *Aktionsschaltfläche* und *Aktion bei Doppelklick* können Argumente und lokale Variable nicht verwendet werden.
- Falls für den Dialog ein eigenes Makro verwendet wird, muss das Makro mit der Anweisung `Return <dialogvariable> enden`.
- Das Ergebnis des Dienstes `open` ist kein Boolean und darf nicht verwendet werden.
- Der Dialog muss mit der Anweisung `<dialogvariable>.accept` oder `<dialogvariable>.cancel` explizit geschlossen werden.

- Wird ein Feld des Dialoges mit Inhalt belegt wird, muss die Aktualisierung des Dialogelementes anschließend mit dem Dienst *invalidate* (*<Variable>*) erzwungen werden, wenn der modale Dialog nicht von einem anderen Dialog gestartet wird.
- Auf Linux-Plattformen sollten modale Dialoge mit Vorsicht verwendet werden, da sie nicht zwingend im Vordergrund bleiben.

Das Kapitel *Beispiel eines nicht modalen Dialoges* zeigt das Beispiel der Verwendung eines modalen Dialoges.

title(*<Variable>*)

Der Dienst versieht den Dialog mit einem Titel. *<Variable>* muss vom Typ *String* sein.

open

Der Dienst öffnet den Dialog und liefert als Boolean das Ergebnis der Beendigung. Das Ergebnis ist *True*, wenn der **Accept**-Button gewählt, oder der Dienst *accept* in einer Aktion ausgeführt wurde und *False*, wenn der **Cancel**-Button gewählt, oder einer der Dienste *cancel* und *close* in einer Aktion ausgeführt wurde.

close

Der Dienst schließt den Dialog. Er ist innerhalb Aktionen verwendbar. Das Ergebnis der Dialogausführung ist *False*. Damit können Sie erreichen, dass der Dialog auch mit einem anderen Button als **Cancel** geschlossen wird. z.B.:

```
dialog.addActionButton( "dialog.close", named: "Close", left: 50, top: 400 )
```

Hinweis: Wenn der Dienst *disableCancel* verwendet wurde, wird der Dialog nicht geschlossen.

cancel

Der Dienst schließt den Dialog. Er ist innerhalb Aktionen verwendbar. Das Ergebnis der Dialogausführung ist *False*. Damit können Sie erreichen, dass der Dialog auch mit einem anderen Button als **Cancel**, geschlossen wird. z.B.

```
dialog.addActionButton( "dialog.cancel", named: "Cancel", left: 50, top: 400 )
```

Hinweis: Wenn der Dienst *disableCancel* verwendet wurde, wird der Dialog nicht geschlossen.

checkMenu(*<Menutitel-Variable>*, *item: <Menüeintrag-Variable >*)

Mit dem Dienst wird ein Menüeintrag mit einem Häkchen versehen. Für *<Menütitel-Variable>* schreiben Sie den sichtbaren Namen des Menütitels. Für *<Menüeintrag-Variable>* schreiben Sie den sichtbaren Namen des Menüeintrages.

accept

Der Dienst schließt den Dialog. Er ist innerhalb Aktionen verwendbar. Das Ergebnis der Dialogausführung ist *True*. Damit können Sie erreichen, dass der Dialog auch mit einem anderen Button als **Accept**, geschlossen wird. z.B.

```
dialog.addActionButton( "dialog.accept", named: "Accept", left: 50, top: 400 ). Bei fehlerhaften oder unvollständigen Angaben wird der Dialog nicht geschlossen (siehe Hinweis im Kapitel Schaltfläche für die normale Dialogbeendigung).
```

Hinweis: Wenn der Dienst *disableAccept* verwendet wurde, wird der Dialog nicht geschlossen.

enableAccept

Das Schließen des Dialoges mit dem **Accept**-Button und mit dem Dienst *accept* wird zugelassen. Der mit *addAcceptButtonLeft* (*<Ganzzahl>*, *top: <Ganzzahl>* [,*default: <Bool-Ausdruck>*]) eingesetzte Button wird ebenso enabled. Diese Option ist voreingestellt.

enableCancel

Das Schließen des Dialoges mit dem Fensterschließknopf, mit dem **Cancel**-Button und mit dem Dienst *cancel* wird zugelassen. Der mit *addCancelButtonLeft* (*<Ganzzahl>*, *top: <Ganzzahl>* [,*default: <Bool-Ausdruck>*]) eingesetzte Button wird ebenso enabled. Diese Option ist voreingestellt.

enableEmergencyExit(*<Bool-Ausdruck>*)

Mit dem Dienst wird dem Bediener der Notausstieg eines Dialoges ermöglicht oder untersagt. Wenn der Notausstieg zugelassen wird (*True*), kann die Dialog alternativ zum **Cancel**-Button auch mit der Tastenkombination **Strg+Umschalt+Alt+Fensterschließknopf** geschlossen werden, auch wenn der **Cancel**-Button mit dem Dienst *disableCancel* deaktiviert ist. Die Standardeinstellung ist *True*.

enableMenu(*<Menütitel-Variable>*, *item: <Menüeintrag-Variable >*)

Mit dem Dienst wird ein Menüeintrag anwählbar gemacht. Für *<Menütitel-Variable>* schreiben Sie den sichtbaren Namen des Menütitels. Für *<Menüeintrag-Variable>* schreiben Sie den sichtbaren Namen des Menüeintrages.

disableAccept

Das Schließen des Dialoges mit dem **Accept**-Button und mit dem Dienst *accept* ist nicht möglich. Der mit *addAcceptButtonLeft* (*<Ganzzahl>*, *top: <Ganzzahl>* [,*default: <Bool-Ausdruck>*]) eingesetzte Button wird ebenso disabled und grau dargestellt.

Achtung: Wenn Sie beide Schließvarianten (*accept* und *cancel*) deaktivieren kann der Dialog nur mit dem Notausstieg geschlossen werden.

disableCancel

Das Schließen des Dialoges mit dem Fensterschließknopf, mit dem **Cancel**-Button und mit dem Dienst *cancel* ist nicht möglich. Der mit *addCancelButtonLeft* (*<Ganzzahl>*, *top: <Ganzzahl>* [,*default: <Bool-Ausdruck>*]) eingesetzte Button wird ebenso disabled und grau dargestellt.

Achtung: Wenn Sie beide Schließvarianten (*accept* und *cancel*) deaktivieren kann der Dialog nur mit dem Notausstieg geschlossen werden.

Hinweis: Für fehlgeleitete Dialoge ist der Notausstieg mit **Umschalt- + Strg- + Alt-Taste + Fensterschließknopf/Cancel**-Button möglich. Nach einer Sicherheitsabfrage wird die Dialogbearbeitung unabhängig von diesem Dienst abgebrochen, und der Dialog geschlossen. Dieser Notausstieg sollte nur im äußersten Notfall verwendet werden, da dadurch vom vorgeplanten Ablauf abgewichen wird, und dadurch die Ergebnisse der Dialogbearbeitung wahrscheinlich unvollständig sind.

disableMenu(*<Menütitel-Variable>*, *item: <Menüeintrag-Variable >*)

Mit dem Dienst wird ein Menüeintrag gegraut, und ist somit nicht mehr anwählbar. Für *<Menütitel-Variable>* schreiben Sie den sichtbaren Namen des Menütitels. Für *<Menüeintrag-Variable>* schreiben Sie den sichtbaren Namen des Menüeintrages.

height(<Ganzzahl>)

Bestimmt die Höhe des Dialoges in Bildpunkten. Wenn der Dialog geöffnet ist, wird die Dialoghöhe aktualisiert. Dadurch können Elemente des Dialoges während der Bearbeitung dynamisch versteckt oder sichtbar gemacht werden.

horizontalScrollBar

Der Dienst liefert `True`, wenn für den Dialog eine horizontale Bildlaufleiste konfiguriert ist.

horizontalScrollBar(<Variable>)

Mit dem Booleanwert `True` für `<Variable>` erhält der Dialog eine horizontale Bildlaufleiste. Wenn die eingestellte Arbeitsfläche des Dialoges schmaler als die erforderliche Breite ist (siehe Kapitel *width(<Ganzzahl>)*), um alle Dialogelemente darzustellen, können Sie mit der horizontalen Bildlaufleiste die Arbeitsfläche zu den noch nicht sichtbaren Dialogelementen verschieben.

isAltKey

Der Dienst liefert `True`, wenn die **Alt**-Taste gedrückt ist.

isShiftKey

Der Dienst liefert `True`, wenn die **Umschalt**-Taste gedrückt ist.

isCtrlKey

Der Dienst liefert `True`, wenn die **Strg**-Taste gedrückt ist.

width(<Ganzzahl>)

Bestimmt die Breite des Dialoges in Bildpunkten. Wenn der Dialog geöffnet ist, wird die Dialogbreite aktualisiert. Dadurch können Elemente des Dialoges während der Bearbeitung dynamisch versteckt oder sichtbar gemacht werden.

width(<Ganzzahl>, height: <Ganzzahl>)

Bestimmt die Breite und Höhe des Dialoges in Bildpunkten. Wenn der Dialog geöffnet ist, werden die Ausmaße des Dialoges aktualisiert. Dadurch können Elemente des Dialoges während der Bearbeitung dynamisch versteckt oder sichtbar gemacht werden.

registerPage(<Variable>)

Wenn für das Argument `<Variable>` der Wert `True` verwendet wird, wird der Dialog zur Verwendung als Seite in einem Register vorbereitet. Die Dienste `open`, `accept` und `cancel` werden bei der Ausführung des Makros ignoriert. Dafür ist der übergeordnete Dialog, der das Register mit der Seite enthält, verantwortlich. **OK**- und **Abbruch**-Schaltflächen, die mit den Diensten `addAcceptButtonLeft(<Ganzzahl>, top: <Ganzzahl> [,default: <Bool-Ausdruck>])`, `addCancelButtonLeft(<Ganzzahl>, top: <Ganzzahl> [,default: <Bool-Ausdruck>])` oder einer Variante eingefügt wurden, werden bei der Ausführung des Makros, dem Öffnen der Seite, ignoriert. Siehe auch Dienst `addRegister(<Registerseiten>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)`.

notebookPage(<Variable>)

Der Dienst ist ein Alias für den Dienst `registerPage(<Variable>)` und ist ab der nächsten Hauptversion nicht mehr verfügbar.

verticalScrollBar

Der Dienst liefert `True`, wenn für den Dialog eine vertikale Bildlaufleiste konfiguriert ist.

verticalScrollBar(<Variable>)

Mit dem Booleanwert `True` für `<Variable>` erhält der Dialog eine vertikale Bildlaufleiste. Wenn die eingestellte Arbeitsfläche des Dialoges niedriger als die erforderliche Höhe ist (siehe Kapitel *height(<Ganzzahl>)*), um alle Dialogelemente darzustellen, können Sie mit der vertikalen Bildlaufleiste die Arbeitsfläche zu den noch nicht sichtbaren Dialogelementen verschieben.

uncheckMenu(<Menütitel-Variable>, item: <Menüeintrag-Variable>)

Mit dem Dienst wird das Häkchen vor einem Menüeintrag entfernt. Für `<Menütitel-Variable>` schreiben Sie den sichtbaren Namen des Menütitels. Für `<Menüeintrag-Variable>` schreiben Sie den sichtbaren Namen des Menüeintrages.

Dienste zum Einfügen von Elementen

Dienstnamen, die mit `add` beginnen, fügen Elemente im Dialog ein. Dabei muss der Name des Elementes eindeutig sein. Es können also nicht zwei Elemente mit dem selben Namen eingefügt werden. Das Einfügen eines Elementes, dessen Name bereits existiert, wird kommentarlos ignoriert.

ActiveX-Komponente

Der Dienst fügt eine visuelle Komponente im Dialog ein. Die Komponente muss eine sog. ActiveX-Komponente des Betriebssystems Microsoft Windows © sein. D.h. Dieser Dienst ist nur auf der Microsoft Plattform verfügbar.


addComponent(<Komponente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)

`<Ganzzahl>` gibt die linke obere Ecke im Dialog sowie die Breite und Höhe an. `<Name>` gibt der Komponente einen Namen mit sie später angesprochen werden kann. Auf der Windows Plattform ist dies ein **ActiveX** Control. Dazu muss die entsprechende Bibliothek mit der **Library** Anweisung geladen sein. Die Komponente muss zuerst mit der Anweisung **New** erzeugt werden. Anschließend wird sie dem Dienst `addComponent(...)` als Argument übergeben. Zu den **ActiveX** Control Namen lesen Sie bitte die entsprechende Komponentenbeschreibung nach. Ein kleines *Beispiel mit Internet Explorer* ist nachfolgend aufgeführt.

activeX(<Name>)

Der Dienst liefert die ActiveX Komponente die zuvor mit dem Dienst `addComponent(<Komponente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)` unter dem Namen `<Name>` eingefügt wurde. Die Komponente unterstützt die Dienste entsprechend seiner Komponentenbeschreibung.

Aktionsschaltfläche

Beispiel	Beschreibung
	Die Dienste fügen eine Schaltfläche mit der angegebenen Benennung ein. Ähnlich wie bei den beschriebenen Schaltflächen Abbruch - und OK - wird durch Klick mit der Maus darauf eine Aktion ausgeführt. Die Aktion wird hier jedoch im ersten Argument mit Makroanweisungen beschrieben. Das Begrenzungszeichen " für Literale muss dabei doppelt geschrieben werden. Die Dienstvarianten mit dem Argument <code>default: True</code> fügen einen Defaultbutton ein. Ein Defaultbutton ist dick umrandet und kann auch mit der Return-Taste aktiviert werden.

`addActionButton(<Anweisungen>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl> [,default: <Bool-Ausdruck>] [,label: <Bezeichner>])`

Der Dienst fügt eine Aktionsschaltfläche im Dialog ein. <Name> ist der interne Name mit dem die Schaltfläche z.B. mit den Diensten `disable()` und `enable()` angesprochen werden kann. <Ganzzahl> gibt die linke obere Ecke des Buttons im Dialog an. Die Breite wird auf 80, die Höhe auf 30 Bildpunkte gesetzt. Für <Anweisungen> schreiben Sie die Anweisungen, die beim Drücken des Buttons im Kontext des Makros ausgeführt werden. Ein Anwendungsbeispiel sehen Sie im *Beispiel mit Internet Explorer*. <Bezeichner> ist die sichtbare Benennung der Schaltfläche. Wenn das Argument `label: nicht` angegeben ist, wird <Name> als der sichtbare Bezeichner verwendet. Mit dem Zeichen & in <Name> kann die Aktionsschaltfläche auch mit der auf & folgenden Taste aktiviert werden.

`addActionButton(<Anweisungen>, named: <Name>, left: <Ganzzahl>,top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [,default: <Bool-Ausdruck>] [,label: <Bezeichner>])`

Der Dienst fügt eine Aktionsschaltfläche im Dialog ein. Die Argumente `width:` und `height:` bestimmen die Breite und Höhe der Schaltfläche in Bildpunkten. Für die Beschreibung der restlichen Argumente siehe `addActionButton(<Anweisungen>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl> [,default: <Bool-Ausdruck>] [,label: <Bezeichner>])`.

`addActionButton(<Anweisungen>, named: <Name>[, width: <Ganzzahl>] [,default: <Bool-Ausdruck>] [,label: <Bezeichner>])`

Der Dienst notiert zur Einfügen eine Aktionsschaltfläche. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. Für die Beschreibung der restlichen Argumente siehe oben. Der Layoutmanager setzt Aktionsschaltflächen immer am unteren Rand des Dialoges von links nach rechts ein. Für die Beschreibung der restlichen Argumente siehe `addActionButton(<Anweisungen>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl> [,default: <Bool-Ausdruck>] [,label: <Bezeichner>])`.

Aktion bei Doppelklick

Der Dienst benennt Makroanweisungen, die durch den Doppelklick auf ein Dialogelement ausgeführt werden. Der Dienst ist nur für Listen- und Tabellenelemente (siehe Kapitel *Listfeld* und Kapitel *Tabelle*) sinnvoll.

`addDoubleAction(<Anweisungen>, named: <Name>)`

Für <Anweisungen> schreiben Sie die Makroanweisungen, die bei einem Doppelklick auf das Dialogelement <Name> ausgeführt werden sollen. Die Anweisungen werden im Kontext des Makros ausgeführt werden. Das Begrenzungszeichen " für Literale müssen Sie in den Anweisungen doppelt schreiben.

Ein Beispiel:

```

...
Dim namen As Array          'Array mit Kundennamen belegen
...
dialog.addList( namen, named: "liste", left: 10, top: 10, width: 300,
               height: 200 )
dialog.addDoubleAction("If dialog.selection("&""liste"&"" ) <> Null
                      Then dialog.accept End If", named: "liste")

```

Aktion bei Inhaltsänderung

Der Dienst benennt Makroanweisungen, die bei der Inhaltsänderung eines Dialogelementes ausgeführt werden. Per Voreinstellung werden die Anweisungen nur durch eine Benutzerinteraktion (z.B. Eingabe über die Tastatur) ausgeführt. Dadurch können Sie dynamisch auf Änderungen in einem Dialogelement reagieren.

Hinweis: Einzeilige Eingabefelder (siehe Kapitel *Eingabefeld* und *Eingabefeld mit Pfeiltasten*) dürfen nicht gleichzeitig eine Aktion und eine Formatregel enthalten ! Das Ergebnis der Eingabe ist in diesem Fall nicht vorhersehbar !

Bei Verwendung des Dienstes *inlineActions* (<Variable>) werden die Anweisungen auch bei programmatischen Änderungen durch entsprechende Dienste ausgeführt.

addAction(<Anweisungen>, named: <Name>[, continuous: <Bool-Ausdruck>])

Für <Anweisungen> schreiben Sie die Makroanweisungen, die bei Veränderung des Inhalts im Kontext des Makros ausgeführt werden. Das Begrenzungszeichen " für Literale müssen Sie in den Anweisungen doppelt schreiben. Mit <Name> benennen Sie das Dialogelement, bei dessen Änderung die Anweisungen ausgeführt werden. Mit <Bool-Ausdruck> bestimmen Sie, ob in einem einzeiligen Eingabefeld (siehe Kapitel *Eingabefeld*, *Eingabefeld mit Pfeiltasten* und *Auswahlliste*) die Aktion bei jeder Inhaltsänderung (True), oder nur beim Verlassen des Feldes (False) ausgeführt wird. Bei allen anderen Dialogelementarten hat das Argument keine Auswirkung. Es ist mit False vorbelegt.

Eine mögliche Endlosschleife durch Aktionen wird bei der Makroausführung erkannt ! Die Ausführung der Aktionskette wird an der erkannten Rekursionsstelle beendet.

Ein Beispiel für eine durch `dialog.inlineActions(True)` hervorgerufene und erkannte Endlosschleife:

```

...
dialog.addInput("&","named: "text1", ...)
dialog.addInput("&","named: "text2", ...)
dialog.inlineActions( True )
dialog.addAction("dialog.value("&""text2"&""
                    put: dialog.value("&""text1"&""))",
                named: "text1", continuous: True)
dialog.addAction("dialog.value("&""text1"&""
                    put: dialog.value("&""text2"&""))",
                named: "text2", continuous: True)
...

```

Die Aktivierung der Aktion durch programmatische Änderungen kann temporär ein- und ausgeschaltet werden. Ein Beispiel soll dies verdeutlichen. Gegeben seien drei Textfelder `text1`, `text2` und `text3`. `text2` soll bei allen Änderungen in `text1` (Benutzerinteraktionen und programmatische Änderungen) mit `text1` synchronisiert werden. `text3` soll nur bei Benutzerinteraktionen und nicht durch programmatische Änderungen in `text2` mit `text2` synchronisiert werden.

```

...
dialog.addInput("&","named: "text1", ...)
dialog.addInput("&","named: "text2", ...)
dialog.addInput("&","named: "text3", ...)
dialog.inlineActions( True )

```

```

dialog.addAction("dialog.inlineActions( False )
                 dialog.value("text2",put: dialog.value("text1"))
                 dialog.inlineActions( True )",
                 named: "text1",continuous: True)
dialog.addAction(" ",named: "text2", continuous: True)
dialog.addAction("dialog.value("text3",put:dialog.value("text2"))"
                 ,named: "text2", continuous: True)
...

```

Aktion beim Dialogstart

Der Dienst benennt Makroanweisungen, die unmittelbar nach dem Öffnen des Dialoges ausgeführt werden.


addPostOpenAction(<Anweisungen>)

Für <Anweisungen> schreiben Sie die Makroanweisungen, die unmittelbar nach dem Öffnen des Dialoges ausgeführt werden sollen. Die Anweisungen werden im Kontext des Makros ausgeführt. Das Begrenzungszeichen " für Literale müssen Sie in den Anweisungen doppelt schreiben.

Hinweis: Im Dialog ist nur eine Aktion für den Dialogstart erlaubt.

Das Kapitel *Beispiel eines Dialoges mit Aktionen zum Start* zeigt ein Beispiel der Verwendung des Dienstes.

Auswahlliste

Beispiel	Beschreibung
	Die Dienste fügen eine Auswahlliste, auch Combobox genannt, ein. Zur Positionierung der Auswahlliste können Sie aus verschiedenen Dienstvarianten wählen. Das Argument <code>readOnly: <Bool-Ausdruck></code> macht das Auswahlfeld editierbar oder nur wählbar. Die Dienstvarianten ohne das Argument <code>readOnly:</code> entsprechen den Dienstvarianten mit dem Argument <code>readOnly: True</code> .

addComboBox (<Auswahl>, choices: <Liste>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [,readOnly: <Bool-Ausdruck>])

Der Dienst fügt eine Combobox im Dialog ein. <Auswahl> bezeichnet die Variable deren Inhalt im Auswahlfeld der Combobox eingesetzt wird. Die Variable kann auch Schriftauszeichnungen (Formatierungsanweisungen) enthalten. Einzelheiten dazu finden Sie im Kapitel *format (<NameOderArrayMitNamen>, with: <Formatregel>)*. <Liste> gibt die Einträge der Auswahlliste an und muss ein Array sein. Die einzelnen Einträge können auch Schriftauszeichnungen (Formatierungsanweisungen) enthalten. Einzelheiten dazu finden Sie im Kapitel *format (<NameOderArrayMitNamen>, with: <Formatregel>)*. <Name> gibt der Combobox einen Namen mit dem sie später angesprochen werden kann. <Ganzzahl> für left: und top: gibt die linke obere Ecke der Combobox im Dialog an. <Ganzzahl> für width: gibt die Breite der Combobox an. Die Höhe wird auf 30 Bildpunkte gesetzt. Mit <Bool-Ausdruck> können Sie das Eingabefeld editierbar (False) oder nur wählbar (True) machen.

addComboBox (<Auswahl>, choices: <Liste>, named: <Name>, leftFraction: <Zahl>, topFraction: <Zahl>, rightFraction: <Zahl> [,readOnly: <Bool-Ausdruck>])

Der Dienst fügt eine Combobox im Dialog ein. <Auswahl> bezeichnet die Variable deren Inhalt im Eingabefeld der Combobox eingesetzt wird. <Liste> gibt die Einträge der Auswahlliste an und muss ein Array sein. <Name> gibt der Combobox einen Namen mit dem sie später angesprochen werden kann. <Zahl> gibt jeweils die relative Koordinate für links

(leftFraction:), oben (topFraction:) und rechts (rightFraction:) im Dialog an. Die Höhe wird auf 30 Bildpunkte gesetzt. Mit <Bool-Ausdruck> können Sie das Eingabefeld editierbar (False) oder nur wählbar (True) machen.

addComboBox(<Auswahl>, choices: <Liste>, named: <Name>, width: <Ganzzahl>, label: <LabelText> [,readOnly: <Bool-Ausdruck>])

Der Dienst notiert zur Einfügen eine Combobox. Der Layoutmanager übernimmt mit dem Dienst build oder buildWithColumns() das eigentliche Einfügen im Dialog. <LabelText> wird als Bezeichner vor die Combobox gesetzt. Die übrigen Argumente siehe Dienst addComboBox (<Auswahl>, choices: <Liste>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [,readOnly: <Bool-Ausdruck>]).

addComboBox (<Auswahl>, choices: <Liste>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, type: "Object")

Der Dienst fügt eine Combobox im Dialog ein. <Auswahl> bezeichnet die Variable deren Inhalt im Auswahlfeld der Combobox eingesetzt wird. <Liste> gibt die Einträge der Auswahlliste an und muss ein Array sein. <Name> gibt der Combobox einen Namen mit dem sie später angesprochen werden kann. <Ganzzahl> für left: und top: gibt die linke obere Ecke der Combobox im Dialog an. <Ganzzahl> für width: gibt die Breite der Combobox an. Die Einträge in <Auswahl> und <Liste> werden von ihrem jeweiligen Piktogramm angeführt. Die Piktogramme können mit dem Dienst withoutIcons (<Name>) entfernt werden. Das Argument type: "Object" bewirkt die Behandlung der Einträge in <Liste> und in <Auswahl> als Objekte und nicht wie bei den übrigen addComboBox (<Auswahl>, choices: <Liste>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [,readOnly: <Bool-Ausdruck>])-Varianten als Zeichenketten. Deshalb ist das Ergebnis des Dienstes value (<Name>) auch das Objekt und nicht die textuelle Darstellung des Eintrages.

Beispiel

Durch den Dienst

```
addCombobox("", choices: worker.children, named: "combo", left: 10,
top: 10, width: 200, readOnly: True)
```

.....

ist das Ergebnis des Dienstes value("combo") der Name des gewählten Bearbeiters als Zeichenkette (String).

Im Gegensatz zu

```
addCombobox("", choices: worker.children, named: "combo", left: 10,
top: 10, width: 200, type: "Object")
```

.....

ist das Ergebnis des Dienstes value("combo") der gewählte Bearbeiter mit seinem Datentyp (Department, Office, Team, Desk oder Machine).

Wenn die Auswahl der Combobox in einem nachfolgenden Dienst als Objekt verwendet werden soll (z.B. stepscheduler.goTo(<Bearbeiter>)), ist die Variante des Dienstes addCombobox(..., type: "Object") ökonomischer, weil die Auswahl direkt an den Dienst stepscheduler.goTo(<Bearbeiter>) weitergereicht werden kann.

Bezeichner

Beispiel	Beschreibung
Bezeichner:	Die Dienste fügen einen Bezeichner, das sind Bezeichnungen für andere Elemente, ein. Der Name des Bezeichners setzt sich aus der Bezeichnung und dem Suffix <code>Label</code> zusammen.

`addLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>)`

Der Dienst fügt einen Bezeichner im Dialog ein. `<Bezeichnung>` ist vom Typ `String` und wird als Text angezeigt. Mit `<Ganzzahl>` geben Sie die X- (`left:`) und Y- (`top:`) Koordinaten im Dialog an.

`addLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>, key: <Label>)`

Der Dienst fügt einen Bezeichner im Dialog ein. Mit dem Schlüssel `<Label>` und der Bezeichnergruppe des aktuellen Bearbeiters wird in der Bezeichnerdatei der zu verwendende Bezeichnertext gesucht. Wenn kein Bezeichnertext gefunden wird, wird `<Bezeichnung>` verwendet. Weitere Einzelheiten siehe `addLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>)`.

Hinweis: Die Datei der Bezeichnergruppen muss im Zeichenformat UTF-8 erstellt sein !


`addFixedLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>)`

Der Dienst fügt einen Bezeichner im Dialog ein. Die verwendete Schrift des Textes ist dabei `Courier`. Weitere Einzelheiten siehe `addLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>)`.

`addFixedLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>, key: <Label>)`

Der Dienst fügt einen Bezeichner im Dialog ein. Die verwendete Schrift des Textes ist dabei `Courier`. Weitere Einzelheiten siehe `addLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>, key: <Label>)`.

Eingabefeld

Beispiel	Beschreibung
	Die Dienste fügen ein Element zur einzeiligen Texteingabe ein. Zur Positionierung des Eingabefeldes können Sie aus verschiedenen Dienstvarianten wählen.

`addInput (<Inhalt>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [,height: <Ganzzahl>] [, border: : <Bool-Ausdruck>])`

Der Dienst fügt ein Eingabefeld im Dialog ein. `<Inhalt>` bezeichnet eine Variable deren Inhalt eingesetzt wird. `<Name>` gibt dem Eingabefeld einen Namen mit dem es später angesprochen werden kann. `<Ganzzahl>` für `left:` und `top:` gibt die Position des Eingabefeldes im Dialog an. `<Ganzzahl>` für `width:` gibt die Breite des Textfeldes an. Wenn das Argument `height:` nicht angegeben ist, wird die Höhe auf 30 Bildpunkte gesetzt. Mit `False` für `<Bool-Ausdruck>` hat das Eingabefeld keinen Textrahmen. Wenn das Argument fehlt, hat das Eingabefeld immer einen Textrahmen. Für den Anwender ist im Dialog ein leeres Eingabefeld ohne Textrahmen visuell nicht zu erkennen. Deshalb eignet ein Eingabefeld ohne Textrahmen in der Regel auch nur als Bezeichner in Verbindung mit einer **Hintergrundfarbe**.

addInput (<Inhalt>, named: <Name>, width: <Ganzzahl>, label: <Labeltext>)


Der Dienst notiert zur Einfügen ein Eingabefeld. Der Layoutmanager übernimmt mit dem Dienst *build* oder *buildWithColumns*(<Spalten>) das eigentliche Einfügen im Dialog.

<Labeltext> wird als Bezeichner vor das Eingabefeld gesetzt. Die übrigen Argumente siehe Dienst *addInput* (<Inhalt>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [,height: <Ganzzahl>] [, border: : <Bool-Ausdruck>]).

addInput (<Inhalt>, named: <Name>, leftFraction: <Zahl>, topFraction: <Zahl>, rightFraction: <Zahl> [, border: : <Bool-Ausdruck>])

Der Dienst fügt ein Eingabefeld im Dialog ein. <Inhalt> bezeichnet eine Variable deren Inhalt eingesetzt wird. <Name> gibt dem Eingabefeld einen Namen mit dem es später angesprochen werden kann. <Zahl> gibt jeweils die relative Koordinate für links (*leftFraction*:), oben (*topFraction*:) und rechts (*rightFraction*:) im Dialog an. Mit *False* für <Bool-Ausdruck> hat das Eingabefeld keinen Textrahmen. Wenn das Argument fehlt, hat das Eingabefeld immer einen Textrahmen. Für den Anwender ist im Dialog ein leeres Eingabefeld ohne Textrahmen visuell nicht zu erkennen. Deshalb eignet ein Eingabefeld ohne Textrahmen in der Regel auch nur als Bezeichner in Verbindung mit einer **Hintergrundfarbe**.

Menü

Beispiel	Beschreibung
	Die Dienste fügt im Dialog einen neuen Menütitel mit seinem ersten Menüeintrag oder zu einem Menütitel einen weiteren Menüeintrag ein.

addMenu (<Anweisungen>, menu: <Menütitel>, item: <Menüeintrag>, help: <Hilfe>, image: <Piktogramm>)

Für <Anweisungen> schreiben Sie die Makroanweisungen, die bei der Auswahl des Menüeintrages ausgeführt werden. Das Begrenzungszeichen " für Literale müssen Sie in den Anweisungen doppelt schreiben. Für <Menütitel> schreiben Sie den sichtbaren Namen des Menütitels. Einen Buchstaben des Namen im Menütitel und im Menüeintrag können Sie mit vorangestelltem Zeichen & als Tastaturkürzel bestimmen. Für <Hilfe> schreiben Sie den Hilfetext für den Menüeintrag. Für <Piktogramm> schreiben Sie den Namen des Piktogramms, der dem Menüeintragstext vorangestellt wird.

addMenu (<Anweisungen>, menu: <Menütitel>, item: <Menüeintrag>, help: <Hilfe>)

Siehe ausführliches Anweisungsformat.


addMenu (<Anweisungen>, menu: <Menütitel>, item: <Menüeintrag>, image: <Piktogramm>)

Siehe ausführliches Anweisungsformat.

addMenu (<Anweisungen>, menu: <Menütitel>, item: <Menüeintrag>)

Siehe ausführliches Anweisungsformat.

Eingabefeld mit Pfeiltasten

Beispiel	Beschreibung
	Die Dienste fügen ein einzeliges Eingabefeld mit sogenannten Spinbuttons ein. Spinbuttons sind kleine Pfeiltasten rechts vom Eingabefeld, mit denen der Inhalt durch Mausklick auf die Pfeiltaste schrittweise vergrößert oder verkleinert werden kann. Zur Positionierung des Eingabefeldes können Sie aus verschiedenen Dienstvarianten wählen. Das Argument <code>readOnly: <Bool-Ausdruck></code> macht das Eingabefeld direkt editierbar oder nur mittels Pfeiltasten änderbar. Die Varianten ohne das Argument <code>readOnly</code> : entsprechen den Varianten mit dem Argument <code>readOnly: True</code> .

`addSpinInput (<Inhalt>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [, readOnly: <Bool-Ausdruck>])`

Der Dienst fügt ein Eingabefeld mit angrenzenden Spinbuttons ein. Zu den Argumenten siehe Dienst `addInput (<Inhalt>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [, height: <Ganzzahl>] [, border: : <Bool-Ausdruck>])`.

`addSpinInput (<Inhalt>, named: <Name>, leftFraction: <Zahl>, topFraction: <Zahl>, rightFraction: <Zahl> [, readOnly: <Bool-Ausdruck>])`

Der Dienst fügt ein Eingabefeld mit angrenzenden Spinbuttons ein. Zu den Argumenten siehe Dienst `addInput (<Inhalt>, named: <Name>, leftFraction: <Zahl>, topFraction: <Zahl>, rightFraction: <Zahl> [, border: : <Bool-Ausdruck>])`.


`addSpinInput (<Inhalt>, named: <Name>, rightFraction: <Zahl>, bottomOffset: <Ganzzahl> [, readOnly: <Bool-Ausdruck>])`

Der Dienst fügt ein Eingabefeld mit angrenzenden Spinbuttons ein. Zu den Argumenten siehe Dienst `addInput (<Inhalt>, named: <Name>, leftFraction: <Zahl>, topFraction: <Zahl>, rightFraction: <Zahl> [, border: : <Bool-Ausdruck>])`.

`addSpinInput (<Inhalt>, named: <Name>, width: <Ganzzahl>, label: <Labeltext> [, readOnly: <Bool-Ausdruck>])`

Der Dienst notiert zur Einfügen ein Eingabefeld. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. `<Labeltext>` wird als Bezeichner vor das Eingabefeld gesetzt. Zu den übrigen Argumenten siehe Dienst `addSpinInput (<Inhalt>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> [, readOnly: <Bool-Ausdruck>])`

Fortschrittsanzeige

Beispiel	Beschreibung
	Der Dienst fügt eine Fortschrittsanzeige ein. Damit kann dem Anwender der Ablauf länger dauernder Aufgaben besser visualisiert werden. Mit dem Dienst <code>progressBar(<Name>, increment: <Ganzzahl>)</code> an geeigneter Stelle wird der farbige Fortschrittsbalken sukzessive bis 100% vergrößert.

`addProgressBar (<Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>, total: <Gesamt>, horizontal: <Bool-Ausdruck>, reverse: <Bool-Ausdruck>)`

Der Dienst fügt eine Fortschrittsanzeige ein. `<Name>` gibt der Fortschrittsanzeige einen Namen mit dem sie später angesprochen werden kann. `<Ganzzahl>` für `left`: und `top`: gibt die linke obere Position der Fortschrittsanzeige im Dialog an. `<Ganzzahl>` für `width`:

und `height`: gibt die Breite und Höhe der Fortschrittsanzeige an. Mit `True` für das Argument `horizontal`: wird der Fortschrittsbalken in horizontaler Richtung gefüllt. Mit `False` für das Argument `horizontal`: wird der Fortschrittsbalken in vertikaler Richtung gefüllt. Mit `False` für das Argument `reverse`: wird der Fortschrittsbalken durch den Dienst `progressBar(<Name>, increment: <Ganzzahl>)` von links nach rechts gefüllt. Mit `True` für das Argument `reverse`: wird der Fortschrittsbalken zum Start vollständig gefüllt und durch den Dienst `progressBar(<Name>, increment: <Ganzzahl>)` von rechts nach links geleert.

Hinweis: Mit den Dienst `value(<Name>)` erhalten Sie den aktuellen Wert der Fortschrittsanzeige (`<Name>` ersetzen Sie mit dem Namen der Fortschrittsanzeige).

Mit den Dienst `value(<Name>)` erhalten Sie den 100%-Wert der Fortschrittsanzeige (`<Name>` ersetzen Sie mit dem Namen der Fortschrittsanzeige).

Grafikelement

Die Dienste fügen eine Grafik im Dialog ein. Die Grafik kann in einem der gängigen Formate (siehe Dokumentation [Business-Process-Management](#), Kapitel *Galerie*) vorliegen. Zur Positionierung der Grafik können Sie aus verschiedenen Dienstvarianten wählen.

`addImage(<Dateiname>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)`

Der Dienst fügt im Dialog die mit `<Dateiname>` benannte Grafik ein. Mit `<Name>` benennen Sie den Dateinamen der Grafik. Mit dem Argument `<Ganzzahl>` geben Sie den Abstand von der linken (`left:`) und oberen (`top:`) Ecke im Dialog in Bildpunkten an. Wenn die Grafikdatei nicht gefunden wird, oder ein unbekanntes Format enthält, erscheint eine Warnmeldung.



```
dialog.addImage("Mann.jpg", named: "bild", left: 0, top: 0)
```

Hinweis: Als Zeichenmodus wird `opaque` verwendet. Dabei wird, im Gegensatz zu `transparent`, an Stelle der weißen Bildpunkte die Hintergrundfarbe des Dialoges verwendet.

`addImage(<Dateiname>, named: <Name>, right: <Ganzzahl>, top: <Ganzzahl>)`

Der Dienst fügt im Dialog die mit `<Dateiname>` benannte Grafik `opaque` ein (siehe Hinweis im Kapitel `addImage(<Dateiname>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)`). Mit dem Argument `<Ganzzahl>` geben Sie den Abstand der Grafik von der rechten (`right:`) oberen (`top:`) Ecke im Dialog an.



```
dialog.addImage("Mann.jpg", named: "bild", right: 0, top: 0)
```

addImage(<Dateiname>, named: <Name>, left: <Ganzzahl>, bottom: <Ganzzahl>)

Der Dienst fügt im Dialog die mit <Dateiname> benannte Grafik **opaque** ein (siehe Hinweis im Kapitel *addImage(<Dateiname>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)*). Mit dem Argument <Ganzzahl> geben Sie den Abstand der Grafik von der linken (left:) unteren (bottom:) Ecke im Dialog an.



```
dialog.addImage("Mann.jpg", named: "bild", left: 0, bottom: 0)
```

addImage(<Dateiname>, named: <Name>, right: <Ganzzahl>, bottom: <Ganzzahl>)

Der Dienst fügt im Dialog die mit <Dateiname> benannte Grafik **opaque** ein (siehe Hinweis im Kapitel *addImage(<Dateiname>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)*). Mit dem Argument <Ganzzahl> geben Sie den Abstand der Grafik von der rechten (right:) unteren (bottom:) Ecke im Dialog an.



```
dialog.addImage("Mann.jpg", named: "bild", right: 0, bottom: 50)
```


addImage(<Dateiname>, named: <Name>, centered: <Ganzzahl>)

Der Dienst fügt in der Dialogmitte die mit <Dateiname> benannte Grafik **opaque** ein (siehe Hinweis im Kapitel *addImage(<Dateiname>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)*). Mit dem Argument <Ganzzahl> geben Sie den Versatz der Grafik von der Dialogmitte nach links oben an.



```
dialog.addImage("Mann.jpg", named: "bild", centered: 0)
```

Größenänderer

Beispiel	Beschreibung
	Mit den Diensten fügen Sie horizontale oder vertikale Balken zur dynamischen Größenänderung, auch Resizer genannt, im Dialog ein. Der Benutzer kann den Resizer mit der Maus verschieben und ändert damit gleichzeitig die Größe angrenzender Elemente. Das ist z.B. für Listfelder, wenn sie sehr unterschiedlich große Inhalte darstellen müssen, sinnvoll.

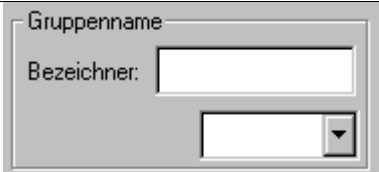
addResizer(<Name>, left: <Ganzzahl>, top: <Ganzzahl>, height: <Ganzzahl> , [thickness: <Ganzzahl>], lefts: <linke Elementnamen>, rights: <rechte Elementnamen>)

Der Dienst fügt einen vertikalen Resizer ein. Mit dem Argument *thickness*: wird die Breite des Resizers bestimmt. Wenn das Argument fehlt ist der Resizer 4 Bildpunkte breit. Für <linke Elementnamen> geben Sie die Namen der Elemente links vom Resizer an. Für <rechte Elementnamen> geben Sie die Namen der Elemente rechts vom Resizer an. Nur die hier angegebenen Dialogelemente werden durch das Verschieben des Resizer vergrößert oder verkleinert. Die Argumente *left*: und *top*: bestimmen die linke obere Ecke des Resizers im Dialog. Das Argument *height*: bestimmt die Höhe des Resizers in Bildpunkten.

addResizer(<Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl> , [thickness: <Ganzzahl>], aboves: <obere Elementnamen>, belows: <untere Elementnamen>)

Der Dienst fügt einen horizontalen Resizer ein. Mit dem Argument *thickness*: kann die Höhe des Resizers bestimmt werden. Wenn das Argument fehlt ist der Resizer 4 Bildpunkte hoch. Für <obere Elementnamen> geben Sie die Namen der Elemente über dem Resizer an. Für <untere Elementnamen> geben Sie die Namen der Elemente unter dem Resizer an. Nur die hier angegebenen Dialogelemente werden durch das Verschieben des Resizers vergrößert oder verkleinert. Die Argumente *left*: und *top*: bestimmen die linke obere Ecke des Resizers im Dialog. Das Argument *height*: bestimmt die Breite des Resizers in Bildpunkten.

Gruppenrahmen

Beispiel	Beschreibung
	Die Dienste fügen einen Rahmen, auch GroupBox genannt, um ein oder mehrere Elemente im Dialog ein. Das ist ein Rechteck mit dem Sie thematisch zusammengehörige Elemente visuell gruppieren können. Am oberen Rand wird die Benennung gezeigt.

addGroupBox(<Array der Gruppenelemente>, named: <Name>)


Der Dienst fügt einen Rahmen im Dialog ein. <Array der Gruppenelemente> sind dabei die Namen der Elemente, die vom Rahmen umschlossen werden.

addGroupBox(<Array der Gruppenelemente>, named: <Name>, width: <Ganzzahl>)

Der Dienst fügt einen Rahmen im Dialog ein. Falls die Breite <Ganzzahl> größer ist, als das breiteste zu umschließende Element, wird <Ganzzahl>, ansonsten die Breite des größ-

ten Elementes für den Rahmen verwendet. Es gelten die übrigen Anwendungsregeln (siehe Dienst `addGroupBox(<Array der Gruppenelemente>, named: <Name>)`).

Kalender

Beispiel	Beschreibung
	Die Dienste fügen eine Schaltfläche zum Öffnen eines Kalenders ein. Das im Kalender ausgewählte Datum wird in ein Eingabeelement eingetragen. Mit Ausnahme von Text -Eingabeelementen (siehe Kapitel <i>Textfeld</i>) muss das Eingabeelement dafür die Formatierung Datum (siehe Kapitel <i>format(<NameOderNamenArray>, as: <Formattyp>, with: <Formatregel>)</i>) besitzen.

`addCalendarButton(<Eingabeelement>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)`

Der Dienst fügt eine Kalenderschaltfläche im Dialog ein.

Das Argument `<Eingabeelement>` enthält den Namen des Elementes, in dem das im Kalender ausgewählte Datum eingetragen wird. `<Name>` gibt der Schaltfläche einen Namen mit dem sie später angesprochen werden kann. `<Ganzzahl>` für `left:` und `top:` gibt die Position der Schaltfläche im Dialog an. `<Ganzzahl>` für `width:` und `height:` gibt die Breite und Höhe der Schaltfläche an.

`addCalendarButton(<Eingabeelement>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)`

Der Dienst fügt eine Kalenderschaltfläche im Dialog ein. Die Breite und Höhe der Schaltfläche ist vorbelegt. Für die Beschreibung der übrigen Argumente siehe Dienst

`addCalendarButton(<Eingabeelement>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)`

`addCalendarButton(<Eingabeelement>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl>)`

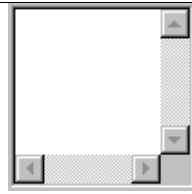
Der Dienst notiert eine Kalenderschaltfläche zum Einfügen im Dialog. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. `<Ganzzahl>` für `width:` und `height:` gibt die Breite und Höhe der Schaltfläche an. Die übrigen Argumente siehe Dienst `addCalendarButton(<Eingabeelement>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)`.

`addCalendarButton(<Eingabeelement>, named: <Name>)`

Der Dienst notiert eine Kalenderschaltfläche zum Einfügen im Dialog. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. Die Breite und Höhe der Schaltfläche ist vorbelegt. Die übrigen Argumente siehe Dienst `addCalendarButton(<Eingabeelement>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl>)`.

Listfeld

Beispiel



Beschreibung

Die Dienste fügen ein Listfeld zur Darstellung von Listen und der Auswahl einer oder mehrere Zeilen daraus ein. Die Listeneinträge werden mit proportionaler Schriftart (`addList`), oder mit der Schriftart Courier (`addFixedList`) dargestellt. Zur Positionierung des Listfeldes können Sie verschiedene Dienstvarianten wählen. Die Dienstvarianten mit dem Argument `multiSelection: True` betreffen Listfelder mit Mehrfachauswahl. Die Zeilen werden mit dem Dienst `withIcons` (`<Name>`) von ihrem jeweiligen Piktogramm angeführt.

`addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein.

Das Argument `<Listenelemente>` enthält die Zeilen der Liste und muss vom Typ Array sein. `<Listenelemente>: Array(<Zeile>[, <Zeile>, ...])`. Zeilen können neben dem Inhalt auch Schriftauszeichnungen (Formatierungsanweisungen) enthalten (siehe Kapitel *Zeileninhalt mit Textauszeichnung*).

`<Name>` gibt dem Listfeld einen Namen mit dem es später angesprochen werden kann.

`<Ganzzahl>` für `left:` und `top:` gibt die Position des Listfeldes im Dialog an. `<Ganzzahl>` für `width:` und `height:` gibt die Breite und Höhe des Listfeldes an. Mit `True` für das Argument `multiSelection:` können im Listfeld mehrere Zeilen gleichzeitig gewählt werden. Bei gedrückter **Strg**-Taste wird die geklickte Zeile der Auswahl hinzugefügt. Bei gedrückter **Umschalt**-Taste werden von der letzten ausgewählten Zeile bis zur geklickten Zeile alle dazwischen liegenden Zeilen zur Auswahl hinzugefügt.

`addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>, horizontalScroll: <Bool-Ausdruck> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein. Die Liste erhält entsprechend dem Argument `horizontalScroll:` einen horizontalen Rollbalken. Weitere Beschreibung siehe Dienst `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`. Die Beschreibung des Argumentes `multiSelection:` finden Sie unter `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`.

`addList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein. `<Zahl>` gibt jeweils die relative Koordinate für rechts (`rightFraction`) und unten (`bottomFraction`) im Dialog an. `<Ganzzahl>` für `topOffset:` gibt den oberen Versatz von an. Weitere Beschreibung siehe Dienst `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`. Die Beschreibung des Argumentes `multiSelection:` finden Sie unter `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`.

`addList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl>, horizontalScroll: <Bool-Ausdruck> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein. Die Liste erhält, wenn das Argument `horizontalScroll: True` ist, einen horizontalen Rollbalken. Weitere Beschreibung siehe Dienst `addList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl> [, multiSelection: <Bool-Ausdruck>])`. Die Beschreibung des Argumen-

tes multiSelection: finden Sie unter `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`.

`addList (<Listenelemente>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl> [, horizontalScroll: <Bool-Ausdruck>] [, multiSelection: <Bool-Ausdruck>], label: <LabelText>)`

Der Dienst notiert zur Einfügen ein Listfeld. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. `<LabelText>` wird als Bezeichner vor das Listfeld gesetzt. Die übrigen Argumente siehe Dienst `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`.

`addFixedList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein. Die Schrift des Listeninhaltes ist Courier. Weiteres siehe Dienst `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`.

`addFixedList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>, horizontalScroll: <Bool-Ausdruck> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein. Die Liste erhält entsprechend dem Argument `horizontalScroll`: einen horizontalen Rollbalken. Weiteres siehe Dienst `addList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`.

`addFixedList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein. Die Schrift des Listeninhaltes ist Courier. Weiteres siehe Dienst `addList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl> [, multiSelection: <Bool-Ausdruck>])`.

`addFixedList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl>, horizontalScroll: <Bool-Ausdruck> [, multiSelection: <Bool-Ausdruck>])`

Der Dienst fügt ein Listfeld im Dialog ein. Die Liste erhält, wenn das Argument `horizontalScroll: True` ist, einen horizontalen Rollbalken. Weiteres siehe Dienst `addList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl> [, multiSelection: <Bool-Ausdruck>])`.

`addFixedList (<Listenelemente>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl> [, horizontalScroll: <Bool-Ausdruck>] [, multiSelection: <Bool-Ausdruck>], label: <LabelText>)`

Der Dienst notiert zur Einfügen ein Listfeld. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. `<LabelText>` wird als Bezeichner, abweichend von den anderen Elementarten, oberhalb des Listfeldes gesetzt. Die übrigen Argumente siehe Dienst `addList(<Listenelemente>, named: <Name>, rightFraction: <Zahl>, topOffset: <Ganzzahl>, bottomFraction: <Zahl> [, multiSelection: <Bool-Ausdruck>])`.

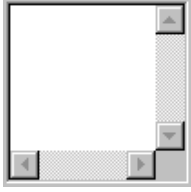
Zeileninhalt mit Textauszeichnung

Sie können die einzelnen Zeileninhalte mit verschiedenen Schriftauszeichnungen (Formatierungen) versehen. Das sind z.B. Schriftarten wie **Fett** oder **Unterstrichen**, und Farben. Diese Schriftauszeichnungen werden im Argument `<Listenelemente>` angegeben. Damit sind Schriftauszeichnungen auf einzelne Zeilen bezogen. Die Zeilen können so unterschiedliche Schriftauszeichnungen erhalten.

Die Syntax für Spalten mit Schriftauszeichnungen lautet:

```
<Zeile>:      Array( <Auszeichnung>, <Zeileninhalt> )
               <Auszeichnung>: Array( <Schriftart>[ , <Farbe|Schriftart>, ... ]
               <Schriftart>: {bold|italic|underline|strikeout}
               <Farbe>: Farbnamen siehe Kapitel Die Farbnamen
               <Zeileninhalt>: Der Inhalt der Zeile.
```

Baumlistfeld

Beispiel	Beschreibung
	<p>Die Dienste fügen ein Listfeld zur Darstellung von hierarchisch angeordneten Listen, auch Baumlisten genannt, und der Auswahl einer Zeile daraus ein.</p> <hr/> <p>Hinweis: Wenn die Elemente der Baumliste, z.B. die Bearbeiter (Systemvariable worker) eigene Symbole besitzen, werden diese vorangestellt.</p>

addTreeList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>, withRoot: <Bool-Ausdruck>, useFolder: <Bool-Ausdruck>)

Der Dienst fügt ein Baumlistfeld im Dialog ein. Die Einträge werden ähnlich einem Dateibrowser als Hierarchie dargestellt.

Das Argument <Listenelemente> enthält die Zeilen der Liste und muss vom Typ Array sein. Das erste Element des Arrays ist die Wurzel der hierarchischen Liste. Anschließend folgen die untergeordneten Einträge. Die Syntax lautet:

```
Array( <wurzel>, Array( <Child> [ , <Child>, ... ] ) )
      Child: Array( <element>[ , Array( <Child> [ , ... ] ) ] ) | <element>
```

<Name> gibt dem Baumlistfeld einen Namen mit dem es später angesprochen werden kann.

<Ganzzahl> für left: und top: gibt die Position des Baumlistfeldes im Dialog an.

<Ganzzahl> für width: und height: gibt die Breite und Höhe des Baumlistfeldes an. Mit False für das Argument withRoot: wird in der Liste das Wurzelelement unterdrückt.

Mit True für das Argument useFolder: werden die Elemente von einem Ordnersymbol angeführt.

addTreeList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)

Die Argumente withRoot: und useFolder: werden mit True belegt. Beschreibung siehe Dienst addTreeList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>, withRoot: <Bool-Ausdruck>, useFolder: <Bool-Ausdruck>).

addTreeList (<Listenelemente>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl>, label: <LabelText>, withRoot: <Bool-Ausdruck>, useFolder: <Bool-Ausdruck>)

Der Dienst notiert zur Einfügen ein Baumlistfeld. Der Layoutmanager übernimmt mit dem Dienst build oder buildWithColumns(<Spalten>) das eigentliche Einfügen im Dialog.

<LabelText> wird als Bezeichner vor das Baumlistfeld gesetzt. Die übrigen Argumente siehe Dienst addTreeList (<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>, withRoot: <Bool-Ausdruck>, useFolder: <Bool-Ausdruck>).

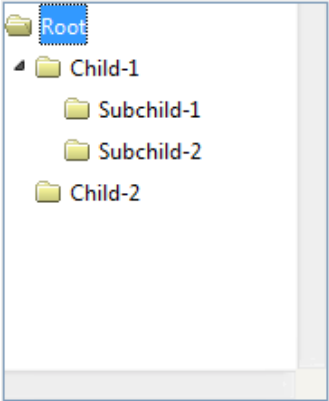
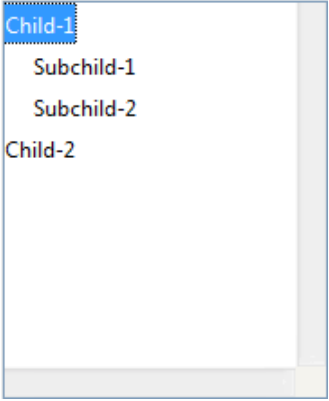
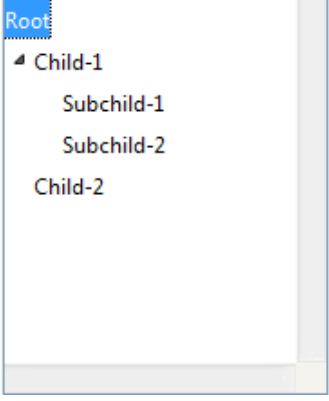
addTreeList (<Listenelemente>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl> , label: <Labeltext>)

Beschreibung siehe Dienst **addTreeList (<Listenelemente>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl> , label: <Labeltext>, withRoot: <Bool-Ausdruck>, useFolder: <Bool-Ausdruck>)**

Beispiele mit benutzerdefinierten Daten als Baumliste

```
Array( "Root", Array( Array( "Child-1", Array( "Subchild-1",  
"Subchild-2" ) ), Array( "Child-2" ) ) ) )
```

als Argument <Listenelemente> ergibt folgende Baumlistenergebnisse:

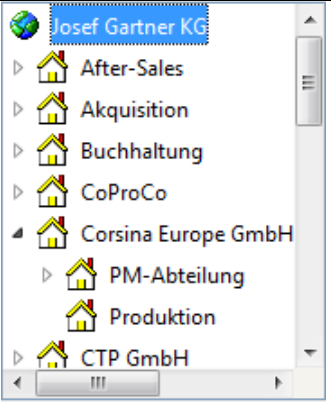
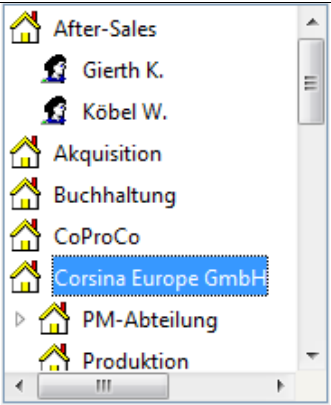
Argumente	Baumliste
withRoot: True, useFolder: True	
withRoot: False, useFolder: False	
withRoot: True, useFolder: False	

Beispielsbilder mit Benutzerelementen

Beispiele mit Organigramm als Baumliste

worker

als Argument `<Listenelemente>` mit der Anmeldung als Unternehmen ergibt folgende Baumlistenenergebnisse. Das Argument `useFolder`: wird ignoriert.

Argumente	Baumliste
<code>withRoot: True, useFolder: False</code>	
<code>withRoot: False, useFolder: False</code>	

Beispielsbilder mit Organigramm

Markierung

Beispiel	Beschreibung
<input type="checkbox"/> Markierung	<p>Die Dienste fügen ein kleines Rechteck, auch Checkbox genannt, mit rechts stehender Benennung ein. Das Rechteck kann mit der Maus angekreuzt werden. Die entsprechende Abfrage nach der Markierung mit <code>value (<Name>)</code> liefert <code>True</code> für angekreuzt oder <code>False</code> für nicht angekreuzt. Zur Positionierung des Eingabefeldes können Sie aus verschiedenen Dienstvarianten wählen.</p>

`addCheckBox (<Label>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)`

Der Dienst fügt eine Checkbox im Dialog ein. `<Label>` ist die Benennung der Checkbox, und `<Name>` gibt der Checkbox einen internen Namen, mit dem sie später angesprochen werden kann. `<Ganzzahl>` für `left`: und `top`: gibt die Position der linken oberen Ecke der Checkbox im Dialog an. Mit dem Zeichen `&` in `<Label>` kann die Markierung der Checkbox auch mit der auf `&` folgenden Taste geändert werden.

`addCheckBox (<Label>, named: <Name>, leftFraction: <Zahl>, leftOffset: <Ganzzahl>, topFraction: <Zahl>, topOffset: <Ganzzahl>)`


Der Dienst fügt eine Checkbox im Dialog ein. `<Label>` ist die Benennung der Checkbox, und `<Name>` gibt der Checkbox einen internen Namen, mit dem sie später angesprochen werden kann. `<Zahl>` gibt jeweils die relative Koordinate für links (`leftFraction`:) und oben (`topFraction`:) im Dialog an. `<Ganzzahl>` gibt den jeweiligen Versatz an. Zu den

Argumenten siehe Dienst `addCheckBox (<Label>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)`.

`addCheckBox (<Label>, named: <Name>)`

Der Dienst notiert zur Einfügen eine Checkbox. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. Zu den Argumenten siehe Dienst `addCheckBox (<Label>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>)`.

Option

Beispiel	Beschreibung
 Option	Die Dienste fügen einen kleinen Knopf, auch Radiobutton genannt, mit rechts stehender Benennung ein. Der Knopf kann mit der Maus ausgewählt werden. Die Dialogsteuerung stellt sicher, dass immer nur ein Radiobutton innerhalb seiner Gruppe ausgewählt ist. Zur Abfrage des Radiobuttons wird der <code><Gruppenname></code> benutzt. Als Ergebnis des Dienstes <code>value (<Name>)</code> erhalten Sie den Namen des gewählten Radiobuttons. Für die Positionierung des Radiobuttons können Sie aus verschiedenen Dienstvarianten wählen.

`addRadioButton (<Label>, named: <Name>, group: <Gruppenname>, left: <Ganzzahl>, top: <Ganzzahl>)`

Der Dienst fügt einen Radiobutton im Dialog ein. `<Label>` ist die Benennung des Knopfs, und `<Name>` gibt dem Radiobutton einen internen Namen, mit dem er später angesprochen werden kann. Mit dem Zeichen `&` in `<Label>` kann die Wahl des Radiobuttons auch mit der auf `&` folgenden Taste geändert werden. `<Gruppenname>` bezeichnet die Gruppe, zu welcher der Radiobutton gehört. `<Ganzzahl>` für `left:` und `top:` gibt die Position der linken oberen Ecke des Radiobuttons im Dialog an. Siehe dazu auch das nachfolgende *Beispiel mit Elementen mit und ohne Positionsangaben*.

`addRadioButton (<Label>, named: <Name>, group: <Gruppenname>, leftFraction: <Zahl>, leftOffset: <Ganzzahl>, topFraction: <Zahl>, topOffset: <Ganzzahl>)`

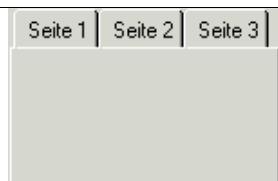
Der Dienst fügt einen Radiobutton im Dialog ein. Beschreibung der Argumente siehe oben. `<Zahl>` gibt jeweils die relative Koordinate für links (`leftFraction:`) und oben (`topFraction:`) im Dialog an. `<Ganzzahl>` gibt den jeweiligen Versatz in Bildpunkten an. Zu den Argumenten siehe Dienst `addRadioButton (<Label>, named: <Name>, group: <Gruppenname>, left: <Ganzzahl>, top: <Ganzzahl>)`.

`addRadioButton (<Label>, named: <Name>, group: <Gruppenname>)`

Der Dienst notiert zur Einfügen einen Radiobutton. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns(<Spalten>)` das eigentliche Einfügen im Dialog. Zu den Argumenten siehe Dienst `addRadioButton (<Label>, named: <Name>, group: <Gruppenname>, left: <Ganzzahl>, top: <Ganzzahl>)`.

Register

Beispiel



Beschreibung

Der Dienst fügt ein Register ein. Damit können verschiedene Inhalte thematisch zusammengefasst werden. Die einzelnen Seiten des Registers werden mit üblichen Makros für Dialoge realisiert. Als einzige Ausnahme müssen diese Makros die Anweisung `registerPage(<Variable>)` enthalten und müssen den erstellten Dialog retournieren: (z.B.. `Return <Dialogvariable>`).

`addRegister(<Registerseiten>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)`

Der Dienst fügt ein Register im Dialog ein. `<Registerseiten>` sind dabei die Seiten des Registers als 2-dimensionales Array (z.B. `Array(Array(<Seitenname>, <dialog>), ...)`).

`<Name>`: ist der interne Name des Dialogelementes.

`<Ganzzahl>`: gibt die linke obere Ecke im Dialog sowie die Breite und Höhe an.

`<Seitenname>`: Die sichtbare Registerbezeichnung der Lasche als String.

`<dialog>`: {ScriptDialog | `<Skiptname>.<Makroname> [(Argumente...)]`}

`<ScriptDialog>`: Dialog der Registerseite den ein Makro als Ergebnis (`Return <dialog>`) liefert oder

`<Skiptname>.<Makroname>` der Name des Makros.

Im ersten Fall (Beispiel siehe Kapitel *Beispiel mit Register*) müssen vor dem Dienst die Makros zum Erstellen der Registerseiten mittels `Call` gerufen werden. Im zweiten Fall (Beispiel siehe Kapitel *Beispiel mit Register (on demand)*) werden die Registerseiten erst bei der Wahl des Registerlasche durch den Benutzer erstellt. Mit der ersten Strategie können Sie nach dem Schließen des Dialoges auf die einzelnen Felder der Registerseiten zugreifen. Das Öffnen des Dialoges dauert jedoch etwas länger, da zuerst auch alle Registerseiten erstellt werden müssen. Mit der zweiten Strategie müssen alle Zugriffe auf die Dialogelemente der Registerseiten in den Makros der Registerseiten selbst enthalten sein. Das Öffnen des Dialoges erfordert jedoch weniger Zeit, da die einzelnen Registerseiten erst bei Anwahl durch den Benutzer erstellt werden müssen. Beispiele für die Verwendung eines Registers finden Sie in den Kapiteln *Beispiel mit Register* und *Beispiel mit Register (on demand)*.

Hinweis: Wenn Sie die zweite Strategie wählen, werden falsche Makronamen und falsche Argumente erst zur Laufzeit gemeldet. Der Skriptcompiler kann Fehler an dieser Stelle nicht erkennen !

Bemerkung

Um ein Registerelement zu erstellen und zu verwenden, gehen Sie nach folgendermaßen vor:

1. Registerseiten mit Makros, die die Anweisung `registerPage(<Variable>)` enthalten und den erstellten Dialog als Variable retournieren, erstellen.
2. Die Registerseiten mit dem Dienst `addRegister(<Registerseiten>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)` einsetzen.
3. Den Dialog mit dem Dienst `open` öffnen.
4. Die Dialoginhalte der Registerseiten unter Verwendung der Registerseitendialogvariablen entnehmen. z.B. `märz = registerseiteUmsatz.value("März")`. Diese Übernahmeanweisungen können

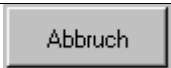
auch in ein eigenes Makro ausgelagert werden. z.B.
`Call Accept.Umsatz(registerseiteUmsatz).`

Hinweis: Statt der Dialogvariable der Registerseite (`registerseiteUmsatz`) kann für die meisten Dienste zur Steuerung, Inhaltsanforderung und Inhaltsübergabe der Dialogelemente auch die Dialogvariable des Hauptdialoges verwendet werden.

`addNotebook(<Registerseiten>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)`

Der Dienst ist ein Alias für `addRegister(<Registerseiten>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)` und sollte nicht mehr verwendet werden, da er zur nächsten Hauptversion entfernt wird.

Schaltfläche für den Dialogabbruch

Beispiel	Beschreibung
	<p>Die Dienste fügen eine Schaltfläche mit der Benennung Abbruch ein. Die Auswahl dieser Schaltfläche schließt den Dialog. Der Dienst <i>open</i> und liefert dafür als Ergebnis <code>False</code>.</p> <p>Die Dienstvarianten mit dem Argument <code>default: True</code> fügen einen Defaultbutton ein. Ein Defaultbutton ist dick umrandet und kann auch mit der Return-Taste aktiviert werden. Die Varianten ohne das Argument <code>default: True</code> entsprechen der Angabe <code>False</code>. Zur Positionierung der Schaltfläche können Sie aus verschiedenen Dienstvarianten wählen.</p>

`addCancelButtonLeft (<Ganzzahl>, top: <Ganzzahl> [, default: <Bool-Ausdruck>])`

Der Dienst fügt an der angegebenen Position einen Button mit dem Text **Abbruch** ein. `<Ganzzahl>` gibt die linke obere Ecke des Buttons im Dialog an. Die Breite wird auf 80, die Höhe auf 30 Bildpunkte gesetzt. Der interne Name des Buttons ist **Cancel**. Für den Dienst *disable* (`<NameOderArrayMitNamen>`) und *enable* (`<NameOderArrayMitNamen>`) verwenden Sie diesen Namen.


`addCancelButtonLeft (<Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, default: <Bool-Ausdruck>])`

Der Dienst fügt an der angegebenen Position einen Button mit dem Text **Abbruch** ein. `<Ganzzahl>` gibt die linke obere Ecke des Buttons im Dialog sowie die Breite und Höhe an.

`addCancelButton und addCancelButtonDefault [(<Bool-Ausdruck>)]`

Die beiden Dienste notieren zur Einfügen einen Button mit dem Text **Abbruch**. Der Layoutmanager übernimmt mit dem Dienst *build* oder *buildWithColumns* (`<Spalten>`) das eigentliche Einfügen im Dialog. Der Dienst `addCancelButton` fügt den Abbruch-Button ohne Defaulteigenschaft ein. Mit dem Dienst `addCancelButtonDefault` bestimmen Sie im Argument `<Bool-Ausdruck>` die Defaulteigenschaft. Der Layoutmanager setzt den Abbruch-Button am unteren Rand links neben dem OK-Button, falls vorhanden.

Schaltfläche für die normale Dialogbeendigung

Beispiel	Beschreibung
	<p>Die Dienste fügen eine Schaltfläche mit der Benennung OK ein. Die Auswahl dieser Schaltfläche schließt den Dialog. Der Dienst <i>open</i> und liefert dafür als Ergebnis <code>True</code>. Zur Positionierung der Schaltfläche können Sie aus verschiedenen Dienstvarianten wählen.</p> <p>Die Dienstvarianten mit dem Argument <code>default: True</code> fügen einen Defaultbutton ein. Ein Defaultbutton ist dick umrandet und kann auch mit der <code>Return</code>-Taste aktiviert werden. Die Dienstvarianten ohne das Argument <code>default:</code> fügen immer einen Defaultbutton ein.</p>

addAcceptButtonLeft (<Ganzzahl>, top: <Ganzzahl> [,default: <Bool-Ausdruck>])

Der Dienst fügt an der angegebenen Position einen Button mit dem Text **OK** ein. <Ganzzahl> gibt die linke obere Ecke des Buttons im Dialog an. Die Breite wird auf 80, die Höhe auf 30 Bildpunkte gesetzt. Der interne Name des Buttons ist **Accept**. Für den Dienst `disable()` und `enable()` benötigen Sie diesen Namen.

addAcceptButtonLeft (<Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [,default: <Bool-Ausdruck>])

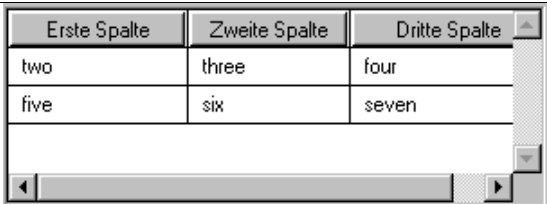
Der Dienst fügt an der angegebenen Position einen Button mit dem Text **OK** ein. <Ganzzahl> gibt die linke obere Ecke des Buttons im Dialog sowie die Breite und Höhe an.

addAccept und addAcceptButtonDefault [(<Bool-Ausdruck>)]

Die beiden Dienst notieren zum Einfügen einen Button mit dem Text **OK**. Der Layoutmanager übernimmt mit dem Dienst *build* oder *buildWithColumns*(<Spalten>) das eigentliche Einfügen im Dialog. Der Dienst `addAcceptButton` fügt einen Defaultbutton ein. Mit dem Dienst `addAcceptButtonDefault` bestimmen Sie im Argument <Bool-Ausdruck> die Defaulteigenschaft. Der Layoutmanager setzt den **OK**-Button immer am unteren rechten Dialogrand ein.

Hinweis: Die normale Dialogbeendigung wird bei einem Fehlen vorgeschiebener Eingaben (siehe Kapitel *mandatory* (<NameOderArrayMitNamen>)) und bei syntaktisch falschen Eingaben in Datums-, Zeit- und Zahlenfeldern (z.b. Buchstabe in einem Datumsfeld) mit einem Hinweisdialog abgelehnt. Dabei wird das betroffene Eingabefeld mit einem Fragezeichen gekennzeichnet.

Tabelle

Beispiel	Beschreibung
	<p>Die Dienste fügen eine Tabelle mit Spalten ein. Ein Beispiel dazu finden Sie im Kapitel <i>Beispiel mit Tabelle</i>.</p>

addTable (<Zeilen>, columns: <Überschrift>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])

Der Dienst fügt eine Tabelle im Dialog ein.

<Zeilen> ist ein `Array` und enthält den Tabelleninhalt. Eine einzelne Zeile wird wiederum als `Array` mit je einem Element pro Spalte dargestellt:

```
<Zeilen>: Array( <Zeile> [,<Zeile>,...])
```

```
<Zeile>: Array(<Spalte> [,<Spalte>,...])
```

<Spalte> kann neben dem Spalteninhalt weitere Schriftauszeichnungen (Formatierungsanweisungen) enthalten (siehe Kapitel *Spalteninhalt mit Schriftauszeichnung*).

<Überschrift> ist ein Array und beschreibt die Spalten der Tabelle.

```
<Überschrift>: Array(<Spalte>[,<Spalte>,...])
```

```
<Spalte>: Array(<Zeichenkette> [,<Ausrichtung>], <Breite> [,<Typ>]
[,<Hintergrundfarbe>])
```

<Zeichenkette>: Die Überschrift der Spalte.

<Ausrichtung>: {L[eft]|R[ight]|C[enter]}: Die Ausrichtung des Spalteninhaltes. **Left** ist voreingestellt.

<Breite>: Die Spaltenbreite in Bildpunkten.

<Typ> {S[tring]|D[ate]|T[ime]|N[umber]|O[bject]}: Der Datentyp der Spalte muss verwendet werden, wenn nach der Spalte dynamisch sortiert werden soll, aber der Datentyp des Spalteninhaltes nicht den originären Datentyp enthält. Wenn die Angabe fehlt, wird der Datentyp `Text` angenommen. Weitere Einzelheiten siehe Kapitel *Eigenschaftsdialog Tabelle in Business-Process-Management*.

<Hintergrundfarbe>: Die Hintergrundfarbe der Spalte (siehe Kapitel *Die Farbennamen*)

Hinweis: Die Reihenfolge der Argumente ist frei wählbar. Die einzelnen Elemente einer <Spalte> können vom Typ {Variable | Literal | Ausdruck | Funktion} sein.

<Name> gibt der Tabelle einen Namen mit dem sie später angesprochen werden kann.

<Ganzzahl> für left: und top: gibt die Position der linken oberen Ecke im Dialog an.

<Ganzzahl> für width: und height: gibt die Breite und Höhe der Tabelle an. Die Beschreibung des Argumentes multiSelection: finden Sie in `addList` (

<Listenelemente>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>]). Das Beispiel eines Dialoges mit Tabelle finden Sie im Kapitel *Beispiel*.

Spalteninhalt mit Schriftauszeichnung

Sie können die einzelnen Spalteninhalte mit verschiedenen Schriftauszeichnungen (Formatierungen) versehen. Das sind z.B. Schriftarten wie **Fett** oder **Unterstrichen**, und Farben. Diese Schriftauszeichnungen werden im Argument <Zeile> angegeben. Damit sind Schriftauszeichnungen auf einzelne Zeilen und nicht auf eine Spalte aller Zeilen bezogen. Die Spalten verschiedener Zeilen können so unterschiedliche Schriftauszeichnungen erhalten.

Die Syntax für Spalten mit Schriftauszeichnungen lautet:

```
<Spalte>: Array( <Auszeichnung>, <Spalteninhalt> )
```

```
<Auszeichnung>: Array(<Schriftart>[,<Farbe|Schriftart>,...])
```

```
<Schriftart>: {bold|italic|underline|strikeout}
```

<Farbe>: Farbennamen siehe Kapitel *Die Farbennamen*

<Spalteninhalt>: Der Inhalt der Spalte

addTable (<Zeilen>, columns: <Überschrift>, named: <Name>, width: <Ganzzahl>; height: <Ganzzahl>, [multiSelection: <Bool-Ausdruck>], label: <LabelText>)

Der Dienst notiert zur Einfügen eine Tabelle. Der Layoutmanager übernimmt mit dem Dienst `build` oder `buildWithColumns()` das eigentliche Einfügen im Dialog. <LabelText> wird als Bezeichner vor die Tabelle gesetzt. Die übrigen Argumente siehe Dienst `addTable (<Zeilen>, columns: <Überschrift>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>])`.

Beispiel

DialogWithTable

'Der Dialog enthält eine Tabelle mit einfachen Textzeilen.
'Mit der Schaltfläche "Sprachwechsel" kann zwischen deutsch und
'englisch umgeschaltet werden.
'Das Beispiel zeigt die Verwendung einer Tabelle, in der die Spalten
'verschieden ausgerichtet sind (linksbündig, rechtsbündig, zentriert)
'und der Spalteninhalt mit Schriftattributen (Font und <Farbe>
'ausgezeichnet sind.

```
Dim dialog As Joops.Scripting.ScriptDialog
Dim records As Array
Dim records2 As Array
Dim columns As Array
Dim theLine As Array
Dim language As Boolean

language = True
records = Array(Array( Array(Array("red","bold"),"redbold"),
                        Array(Array("italic"),"onlyitalic"),
                        Array(Array("blue"), "onlyblue") ),
                Array( Array(Array("underline","bold"),"underlinebold"),
                        "six", "seven"),
                Array( "eight", "nine", "then"),
                Array( "eleven", "twelve", "thirteen"),
                Array( "fourteen", "fifteen", "sixteen"),
                Array( "seventeen", "eighteen", "nineteen"),
                Array( "twenty", "twentyone", "twentytwo")))

records2 = Array(Array( "eins", "zwei", "drei" ),
                Array( "vier", "fünf", "sechs"),
                Array( "sieben", "acht", "neun"),
                Array( "zehn", "elf", "zwölf"),
                Array( "dreizehn", "vierzehn", "fünfzehn"),
                Array( "sechzehn", "siebzehn", "achzehn"),
                Array( "neunzehn", "zwanzig", "einundzwanzig"))

columns = Array(Array("Left", 120), Array("Center", "Center", 100),
                Array("Right", "Right", 110))
dialog = New Joops.Scripting.ScriptDialog
dialog.title( "Die Überschrift" )
dialog.addLabel( "Die Tabelle", left: 5, top: 10 )
dialog.addTable(records, columns: columns, named: "table",
                left: 5, top: 15, width: 300, height: 110)
dialog.addAction("If (dialog.selection("&quot;table&quot;") <> Null)
                Then theLine = dialog.selection("&quot;table&quot;")
                MsgBox( "&quot;Row: '" & Join(theLine, "' ") &
                        "' selected")

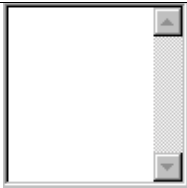
                End If",
                named: "table")
dialog.addAcceptButtonLeft( 225, top: 130 )
```

```

dialog.addActionButton("If language = True
                        Then dialog.put("table", value: records2)
                        language = False
                        Else dialog.put("table", value: record)
                        language = True
                        End If",
                        named: "Sprachwechsel", left: 130, top: 130,
                        default: False)
dialog.addCancelButtonLeft( 40, top: 130 )
dialog.help("table", with: "Die Tabelle mit den zweisprachigen Texten")
dialog.help("Sprachwechsel",
            with: "Damit kann die Sprache umgestellt werden")
dialog.help("Accept",
            with: "Damit wird der Dialog ordnungsgemäß geschlossen")
dialog.help("Cancel", with: ": "Damit wird der Dialog abgebrochen ")
dialog.select("table", value: records(6))
dialog.width( 310 )
dialog.height( 160 )
dialog.open
MsgBox(dialog.selectionIndex( "table" ) & "-st item selected")
Return dialog.selection ( "table" )

```

Textfeld

Beispiel	Beschreibung
	Die Dienste fügen ein Feld für mehrzeilige Texteingaben ein. Zur Positionierung des Textfeldes können Sie aus verschiedenen Dienstvarianten wählen. Die Tab-Taste fügt im Gegensatz zu den anderen Dialogelementen im Text einen Tabulator ein. Zur Tab-Taste siehe auch Kapitel <i>disable</i> (<i><NameOderArrayMitNamen></i>).

addText (<Text>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)

Der Dienst fügt ein Texteingabefeld im Dialog ein. <Text> bezeichnet eine Variable deren Inhalt im Textfeld eingesetzt wird. <Name> gibt dem Textfeld einen Namen mit dem es später angesprochen werden kann. Mit <Ganzzahl> geben Sie die Position (left: und top:) des Textfeldes im Dialog an. <Ganzzahl> für width: und height: gibt die Breite und Höhe des Textfeldes in Punkten an.


addText (<Text>, named: <Name>, rightFraction: <Zahl>, topFraction: <Zahl>, bottomFraction: <Zahl>)

Der Dienst fügt ein Texteingabefeld im Dialog ein. <Text> bezeichnet eine Variable deren Inhalt im Textfeld eingesetzt wird. <Name> gibt dem Textfeld einen Namen mit dem es später angesprochen werden kann. <Zahl> gibt jeweils die relative Koordinate für rechts (rightFraction:), oben (topFraction:) und unten (bottomFraction:) im Dialoge an.

addText (<Text>, named: <Name>, width: <Ganzzahl>, height: <Ganzzahl>, label: <LabelText>)

Der Dienst notiert zur Einfügen ein Textfeld. Der Layoutmanager übernimmt mit dem Dienst *build* oder *buildWithColumns*(*<Spalten>*) das eigentliche Einfügen im Dialog. <LabelText> wird als Bezeichner vor das Textfeld gesetzt. Zu den übrigen Argumenten siehe Dienst *addText* (<Text>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>).

Trennlinie

Beispiel	Beschreibung
	Die Dienste fügen eine Trennlinie im Dialog ein.

addDivider(<Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>)

Der Dienst fügt eine horizontale Trennlinie im Dialog ein. <Name> gibt der Trennlinie einen Namen mit dem sie später angesprochen werden kann. <Ganzzahl> gibt dabei den linken oberen Startpunkt im Dialog und die Breite der Trennlinie an.

addDividerLeft(<Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>)

Der Dienst fügt eine horizontale Trennlinie im Dialog ein. Der Name der Trennlinie ist Divider1. Beschreibung siehe Dienst *addDivider(<Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>)*.

addDivider(<Name>, left: <Ganzzahl>, top: <Ganzzahl>, height: <Ganzzahl>)

Der Dienst fügt eine vertikale Trennlinie im Dialog ein. <Name> gibt der Trennlinie einen Namen mit dem sie später angesprochen werden kann. <Ganzzahl> gibt dabei den linken oberen Startpunkt im Dialog und die Höhe der Trennlinie an.

addDividerLeft(<Ganzzahl>, top: <Ganzzahl>, height: <Ganzzahl>)

Der Dienst fügt eine vertikale Trennlinie im Dialog ein. Der Name der Trennlinie ist Divider1. Beschreibung siehe Dienst *addDivider(<Name>, left: <Ganzzahl>, top: <Ganzzahl>, height: <Ganzzahl>)*.

Dialog konstruieren

Die Dienste konstruieren den Dialog mit seinen Elementen. Die Elemente müssen zuvor mit den `add`-Diensten ohne Positionierungsangaben eingefügt worden sein. Zusätzlich finden Sie in diesem Abschnitt einige Hilfsdienste dafür. Für weitere Details zum Konstruieren eines Dialoges lesen Sie bitte auch das Kapitel *Strategien für das Konstruieren eines Dialoges*.

build

Der Dienst fügt die Elemente mit dem Layoutmanager im Dialog ein. Der Dialog wird quadratisch erstellt. D.h. Die Anzahl die Spalten wird so gewählt, dass der Dialog annähernd quadratisch ist. Es werden dabei nur die Elemente ohne Positionsangabe eingefügt. Zusätzlich können Sie weitere Elemente mit Positionsangabe im Dialog einfügen. Der Dienst darf für einen Dialog nur **einmal** verwendet werden. Sie dürfen `build` und `buildWithColumns()` nicht gleichzeitig verwenden.

buildWithColumns(<Spalten>)

Der Dienst fügt die Elemente mit dem Layoutmanager mehrspaltig im Dialog ein. `<Spalten>` ist dabei die Anzahl der zu verwendenden Spalten. Es werden dabei nur die Elemente ohne Positionsangabe eingefügt. Zusätzlich können Sie weitere Elemente mit Positionsangabe im Dialog einfügen. Der Dienst darf für einen Dialog nur **einmal** verwendet werden. Sie dürfen `build` und `buildWithColumns()` nicht gleichzeitig verwenden.

buildWithColumns(<Spalten>, startAt: <Offset>)

Siehe Dienst *buildWithColumns(<Spalten>)*. Das erste Element der Spalte wird mit dem vertikalen Abstand `<Offset>` Bildpunkte im Dialog eingesetzt.

buildWithColumns(<Spalten>, buttonsOffset: <Buttonoffset>)

Siehe Dienst *buildWithColumns(<Spalten>)*. Die Schaltflächen werden mit dem Abstand `<Buttonoffset>` zu den übrigen Eingabefeldern nach unten versetzt im Dialog eingefügt.

buildWithColumns(<Spalten>, startAt: <Offset>, buttonsOffset: <Buttonoffset>)

Der Dienst ist eine Kombination der beiden Dienste *buildWithColumns(<Spalten>, startAt: <Offset>)* und *buildWithColumns(<Spalten>, buttonsOffset: <Buttonoffset>)*.

buildStartAt(<Offset>)

Siehe Dienst *build*. Das erste Element wird mit dem vertikalen Abstand `<Offset>` Bildpunkte im Dialog eingesetzt.

buildButtonsOffset(<Buttonoffset>)

Siehe Dienst *build*. Die Schaltflächen werden mit dem Abstand `<Buttonoffset>` zu den übrigen Eingabefeldern noch unten versetzt im Dialog eingefügt.

buildStartAt(<Offset>, buttonsOffset: <Buttonoffset>)

Der Dienst ist eine Kombination der beiden Dienste *buildStartAt(<Offset>)* und *buildButtonsOffset(<Buttonoffset>)*.

buttonsOrigin

Der Dienst liefert die X- und Y-Koordinate des oberen Randes Schaltflächen.

buttonsCorner

Der Dienst liefert die X- und Y-Koordinate des unteren Randes der Schaltflächen

buttonsWidth

Der Dienst liefert die Breite aller Schaltflächen.

buttonsHeight

Der Dienst liefert die Höhe aller Schaltflächen

inputsCorner

Der Dienst liefert die X- und Y-Koordinate des unteren Endes der Eingabefelder

inputsHeight

Der Dienst liefert die Höhe aller Eingabefelder.

inputsOrigin

Der Dienst liefert die X- und Y-Koordinate des oberen Randes Eingabefelder

inputsWidth

Der Dienst liefert die Breite aller Eingabefelder.

Feldeigenschaften vergeben und Feldinhalte erfragen

Mit den Diensten in diesem Abschnitt vergeben Sie für ein Feld weitere Eigenschaften und erhalten den Inhalt der Texteingabe- oder des Auswahlfeldes. Prinzipiell müssen Sie immer den Namen des Feldes angeben. Das ist der Name, der im korrespondierenden add-Dienst vergeben wurde.

backgroundColor (<NameOderArrayMitNamen>, with: <Farbename>)

Der Dienst bestimmt die Hintergrundfarbe in den mit <NameOderArrayMitNamen> benannten Elemente. Der Dienst ist für *Trennlinien* und *Größenänderer* nicht erlaubt. <Farbename> ist der Name der Hintergrundfarbe. Die möglichen Farbennamen können Sie dem Kapitel *Die Farbennamen* entnehmen.

bold (<NameOderArrayMitNamen>)

Der Dienst setzt die Schriftart bei den mit <NameOderArrayMitNamen> benannten Elementen auf **Fett**. Der Dienst hat nur Auswirkung auf Elemente mit einem Bezeichnertext (*Bezeichner*, *Option*, *Markierung*, usw.).

bold (<NameOderArrayMitNamen>, register: <Elementname>)

Der Dienst setzt die Schriftart der mit <NameOderArrayMitNamen> benannten Registerlaschen des Registerelementes <Elementname> auf **Fett**.

color (<NameOderArrayMitNamen>, with: <Farbename>)

Der Dienst setzt die Vordergrundfarbe in den mit <NameOderArrayMitNamen> benannten Elementen. In Elementen mit einem Bezeichner (*Bezeichner*, *Option*, *Markierung*, usw.) betrifft das die Schriftfarbe des Bezeichners. In Elementen mit einem Eingabefeld oder Listfeld betrifft das die Farbe des Textes. <Farbename> ist der Name der Vordergrundfarbe. Die möglichen Farbennamen können Sie dem Kapitel *Die Farbennamen* entnehmen.

color (<NameOderArrayMitNamen>, register: <Elementname>, with: <Farbename>)

Der Dienst setzt die Schriftfarbe der mit <NameOderArrayMitNamen> benannten Registerlaschen des Registerelementes <Elementname>. <Farbename> ist der Name der Hintergrundfarbe. Die möglichen Farbenamen können Sie dem Kapitel *Die Farbenamen* entnehmen.

columns(<Name>)

Der Dienst liefert in einem Array die Spalten des Tabellenelementes mit dem Namen <Name>. Das Ergebnis:

Array(<Spalte>, ...)

<Spalte>: Array(<Spaltenüberschrift>, <Breite>)

< Spaltenüberschrift>: Die Spaltenüberschrift als String

<Breite>: Die Spaltenbreite als Integer

columns(<Name>, width: <Spalten>)

Der Dienst bestimmt die Breite der Spalten im Tabellenelement mit dem Namen <Name>. <Spalten> ist ein Array mit fortlaufenden Spaltenbreiten oder ist ein Array jeweils Spaltenüberschrift und Breite als Array.

<Spalten>: Array(<Spalte>, ...)

<Spalte>: Array(<Breite>) oder

Array(<Spaltenüberschrift>, <Breite>)

Hinweis: Der Dienst wird mit einer Fehlermeldung quittiert, wenn das Tabellenelement die Spalte mit der Bezeichnung <Spaltenüberschrift> nicht enthält.

disable (<NameOderArrayMitNamen>)

Der Dienst deaktiviert das mit <NameOderArrayMitNamen> benannte Dialogelement. Maus- und Tastatureingaben sind somit für dieses Element nicht mehr möglich. <NameOderArrayMitNamen> kann ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Das Element wird i.d.R. grau dargestellt. In Eingabefeldern sind die Feldfunktionen **Kopieren** und **Suchen** mit der Tastatur und der Maus, sowie die Tab-Taste zum Verlassen des Dialogelementes weiterhin möglich.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns*(<Spalten>) benutzt werden.

In deaktivierten Textfeldern (siehe Kapitel *Textfeld*) wirkt die Tab-Taste, im Gegensatz zu aktivierten Textfeldern wie die Tastenkombination **Strg+Tab**.

doubleClick(<Name>)

Mit dem Dienst wird ein Doppelklick auf eine Zeile in einem Dialogelement mit dem Namen <Name> simuliert. <Name> muss ein listenartiges Dialogelement (siehe Kapitel *Listfeld*), oder eine Tabelle (siehe Kapitel *Tabelle*) benennen. I.d.R. wird der Dienst innerhalb eines Anweisungsblocks des Dienstes *addAction*(...) verwendet.

emphasize (<Name>, with: <FarbenameOderSchriftart>, from: <von>, to: <bis>)

Der Dienst koloriert den Textinhalt oder bestimmt die Schriftart des Textinhaltes im Dialogelement <Name>. <FarbenameOderSchriftart> benennt die Farbe oder die Schriftart (siehe Kapitel *Die Farbenamen* und *Zeileninhalt mit Textauszeichnung*). Mit <von> und <bis> bestimmen Sie den betroffenen Textbereich.

Hinweis: Der Dienst ist nur für Dialogelemente mit einzeiligem Text anwendbar (z.B. *Bezeichner*, *Option*, *Markierung*, *Eingabefeld*, usw.). Das Dialogelement sollte nicht editierbar sein (siehe Kapitel *disable* (*<NameOderArrayMitNamen>*)).

enable (*<NameOderArrayMitNamen>*)

Der Dienst aktiviert das mit *<NameOderArrayMitNamen>* benannte Dialogelement. *<NameOderArrayMitNamen>* kann ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Maus- und Tastatureingaben sind somit für dieses Element möglich.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns*(*<Spalten>*) benutzt werden.

expandRoot (*<Name>*)

Der Dienst expandiert den obersten Eintrag, die Wurzel des mit *<Name>* benannten Baumdialogelementes.

focus (*<Name>*)

Der Dienst setzt den Eingabefokus auf das angegebene Dialogelement. Der Eingabefokus wird durch die Schreibmarke in einem *Eingabefeld* und der Markierung seines Inhaltes oder durch einen gestrichelten Rahmen um das Dialogelement (*Listfeld* und *Auswahlliste*) angezeigt. Bei einer *Tabelle* wird die erste Zeile in der Tabelle markiert. Bei einer *Option* wird der Fokus auf den ersten Radiobutton innerhalb der Optionsgruppe gesetzt, wenn *<Name>* den Gruppennamen der Option bezeichnet, ansonsten auf den Radiobutton mit dem Namen *<Name>*.

Hinweis: Wenn der Dienst vor dem Öffnen des Dialoges mit dem Dienst *open* verwendet wird, wird dadurch das Dialogelement bestimmt, das beim Öffnen des Dialoges den Fokus erhält. Ansonsten findet der Dienst in einem Aktionsblock (siehe Kapitel *Aktionsschaltfläche*, *Aktion bei Doppelklick* und *Aktion bei Inhaltsänderung*) seine Anwendung. Nur das letzte Vorkommen des Dienstes *focus* (*<Name>*), *focus* (*<ArrayMitNamen>*, *empty: <Bool-Ausdruck>*) und *focus* (*<Name>*, *proceed: <Bool-Ausdruck>*) vor dem Dienst *open* wird verwendet.

focus (*<ArrayMitNamen>*, *empty: <Bool-Ausdruck>*)

Der Dienst setzt abhängig vom Argument *<Bool-Ausdruck>* den Fokus auf das erste leere Dialogelement (*True*) oder auf das erste nicht leere Dialogelement (*False*) aus der Namensliste *<ArrayMitNamen>*.

Hinweis: Wenn der Dienst vor dem Öffnen des Dialoges mit dem Dienst *open* verwendet wird, wird dadurch das Dialogelement bestimmt, das beim Öffnen des Dialoges den Fokus erhält. Ansonsten findet der Dienst in einem Aktionsblock (siehe Kapitel *Aktionsschaltfläche*, *Aktion bei Doppelklick* und *Aktion bei Inhaltsänderung*) seine Anwendung. Nur das letzte Vorkommen des Dienstes *focus* (*<Name>*), *focus* (*<ArrayMitNamen>*, *empty: <Bool-Ausdruck>*) und *focus* (*<Name>*, *proceed: <Bool-Ausdruck>*) vor dem Dienst *open* wird verwendet.

focus (*<Name>*, *proceed: <Bool-Ausdruck>*)

Der Dienst setzt den Fokus auf das mit *<Name>* benannte Dialogelement. Wenn *<Bool-Ausdruck>* *True* ergibt, und *<Name>* ein *Eingabefeld* oder ein *Textfeld* bezeichnet, wird die Schreibmarke an das Ende des Inhaltes gesetzt. Wenn *<Name>* ein anderes Dialogelement bezeichnet, siehe *focus* (*<Name>*).

Hinweis: Wenn der Dienst vor dem Öffnen des Dialoges mit dem Dienst *open* verwendet wird, wird dadurch das Dialogelement bestimmt, das beim Öffnen des Dialoges den Fokus erhält. Ansonsten findet der Dienst in einem Aktionsblock (siehe Kapitel *Aktionsschaltfläche*, *Aktion bei Doppelklick* und *Aktion bei Inhaltsänderung*) seine Anwendung. Nur das letzte Vorkommen des Dienstes *focus* (*<Name>*), *focus* (*<ArrayMitNamen>*, *empty: <Bool-Ausdruck>*) und *focus* (*<Name>*, *proceed: <Bool-Ausdruck>*) vor dem Dienst *open* wird verwendet.

font (*<NameOderArrayMitNamen>*, *use: <Fontname>*)

Der Dienst verwendet für den Bezeichner oder für die Texteingabe des Dialogelementes mit dem Namen *<NameOderArrayMitNamen>* die Schrift *<Fontname>*. *<Fontname>* kann die Wildcards ? und * enthalten.

font (*<NameOderArrayMitNamen>*, *use: <Fontname>*, *scale: <Faktor>*)

Der Dienst ist eine Zusammenfassung der beiden Dienste *font* (*<NameOderArrayMitNamen>*, *use: <Fontname>*) und *scale* (*<NameOderArrayMitNamen>*, *by: <Faktor>*). Er verwendet für den Bezeichner oder für die Texteingabe des Dialogelementes mit dem Namen *<NameOderArrayMitNamen>* die Schrift *<Fontname>* und die Skalierung *<Faktor>*.

format (*<NameOderArrayMitNamen>*, *with: <Formatregel>*)

Mit dem Dienst wird für das Eingabefeld des Dialogelementes mit dem Namen *<NameOderArrayMitNamen>* die Formatierungsregel *<Formatregel>* festgelegt. Dabei kann *<NameOderArrayMitNamen>* ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns* (*<Spalten>*) benutzt werden.

Die Anweisung muss zum Datentyp der Variable passen. D.h. Die Regel kann nur für eine Variable mit konkretem Inhalt bestimmt werden.

<Formatregel> wird für die Variablenarten in der Tabelle beschrieben, wobei bei der Farbenregel die umschließenden [] geschrieben werden müssen:

Variablenart	Formatregel	Beschreibung
String	[Farbenregel] @	Das darzustellende Zeichen wird mit @ maskiert. Alle anderen Zeichen der Formatregel werden direkt übernommen. Überschüssige Zeichen im Textfeld werden unterdrückt. Überschüssige Formatregelzeichen werden ignoriert. Der erste Regelteil, falls vorhanden, stellt das Textfeld in der entsprechenden Farbe dar (siehe Tabelle <i>Regelteil [Farbenregel]</i>). Der Text 102030 wird mit der Regel [BLUE]@@ - @@ / @@ als 10 - 20 / 30 dargestellt.
Integer Float Double	[Farbenregel] #0?., \$-+/():space	Die Regel enthält vier Teile, getrennt durch das Zeichen ;. <ol style="list-style-type: none"> 1. Regel für positive Zahl 2. Regel für negative Zahl 3. Regel für 0

Variablenart	Formatregel	Beschreibung
Date	[Farbenregel] d.m.y	<p>4. Regel für Null Variable</p> <p># Platzhalter für Ziffern. Enthält die Zahl mehr Nachkommastellen als # in der Regel, wird die Ausgabe gerundet. Überschüssige # nach dem Komma und vor dem Dezimalpunkt werden unterdrückt. Die Zahl 123.456 mit der Regel #.## ergibt 123,46.</p> <p>0 Platzhalter für Ziffern. Die Zahl wird um die Platzhalter an der angegebenen Stelle erweitert. Die Zahl 123.45 mit der Regel 0#.## ergibt 0123,45.</p> <p>? Platzhalter für Ziffern entspricht dem Platzhalter 0. Zusätzlich wird für überschüssige ? ein Leerzeichen ausgegeben. Die Zahl 123.45 mit der Regel \$???###.## ergibt die Ausgabe \$ 123,45.</p> <p>, Tausendertrenner, wenn in der Regel enthalten, wird in die Ausgabe, falls erforderlich, der Tausendertrenner übernommen. Die Zahl 12345 mit der Regel 00000000,000 ergibt 00000012.345. Die Zahl 12345 mit der Regel #,# ergibt 12.345.</p> <p>. Dezimaltrenner gibt anhand seiner Position an, wieviel Vor- und Nachkommastellen verwendet werden. Die Zahl 12345.67 mit der Regel 0000000,000.00 ergibt 0012.345,67. Die Zahl 12345.67 mit der Regel #,#.00 ergibt 12.345,67.</p> <p>\$-+/() :space Trennzeichen werden positionsgerecht übernommen.</p> <p>Zusätzlich gelten die Formatregeln für Farbdarstellung (siehe Tabelle <i>Regelteil [Farbenregel]</i>). Die Zahl -123.45 mit der Regel ##,#.##;[RED](\$#,###.##);0.0 ergibt (\$123,45).</p> <p>d steht für die Tage. Enthält die Regel dafür d, werden die Tage 1- oder 2-stellig ausgegeben (1-31). Enthält die Regel dafür dd, werden die Tage 2-stellig ausgegeben (01-31). Enthält die Regel dafür ddd, werden die Abkürzungen der Tagesnamen ausgegeben (Mon., Die., Don., usw.). Enthält die Regel dafür dddd, werden</p>

Variablenart	Formatregel	Beschreibung
Time	[Farbenregel] h:m:s AM/PM	<p>vollständigen Tagesnamen ausgegeben.</p> <p>m steht für das Monat. Enthält die Regel mmmm, werden die Monatsnamen ausgegeben (Januar, Februar, usw.). Enthält die Regel mmm, werden die Abkürzungen der Monatsnamen ausgegeben (Jan., Feb., usw.). Enthält die Regel mm, werden Monatszahlen 2-stellig (01-12) ausgegeben. Enthält die Regel m, werden Monatszahlen 1- oder 2-stellig (1-12) ausgegeben.</p> <p>y steht für das Jahr. Enthält die Regel yy, wird das Jahr 1- oder 2-stellig ausgegeben (00-99). Enthält die Regel yyyy, wird das Jahr mit Jahrhundert ausgegeben (2004).</p> <p>Das Datum 1.1.2004 mit der Regel d.mmmm.yyyy ergibt 1. Januar 2004.</p> <p>Zusätzlich gelten die Formatregeln für die Farbdarstellung (siehe Tabelle <i>Regelteil [Farbenregel]</i>).</p> <p>h steht für die Stunden. h gibt die Stunden 1- bis 2-stellig (1-24) aus. hh gibt die Stunden immer 2-stellig aus (01-24). Wenn die Regel AM/PM enthält basiert die Tagesberechnung auf 12, ansonsten auf 24 Stunden</p> <p>m steht für die Minuten. m gibt die Minuten 1- bis 2-stellig (1-60) aus. mm gibt die Minuten immer 2-stellig aus (01-60).</p> <p>s steht für die Sekunden. s gibt die Sekunden 1- bis 2-stellig (0-60) aus. ss gibt die Sekunden immer 2-stellig aus (00-60).</p> <p>Die Uhrzeit 14:06:58 mit der Regel hh.mm.ss AM/PM ergibt 02:06:58 AM/PM.</p> <p>Zusätzlich gelten die Formatregeln für Farbdarstellung (siehe Tabelle <i>Regelteil [Farbenregel]</i>).</p>
Timestamp	d.m.y h:m:s	<p>Der Name ist eine Kombination aus den Arten <i>Date</i> und <i>Time</i>.</p>

Regelteil [Farbenregel]
[Farbenregel]

	Beschreibung
<p>[Farbename]</p> <p>[Float Float Float]</p>	<p>Der erste Regelteil, falls vorhanden, stellt das Textfeld in der entsprechenden Farbe dar. Dabei kann</p> <ul style="list-style-type: none"> der Name der Farbe (siehe Tabelle <i>Die Farbnamen</i>) oder der RGB Farbwerte <p>angegeben werden. Jeder RGB-Farbwert</p>

muss dabei mit 0.0 bis 1.0 angegeben werden.

Die Farbnamen

Farbname	Beschreibung
BLACK	Schwarz
BLUE	Blau
CYAN	Zyan
GREEN	Grün
MAGENTA	Lila
RED	Rot
WHITE	Weiß
YELLOW	Gelb
ORANGE	Orange
BROWN	Braun
GRAY	Gau
NAVY	Marinegrün
OLIVE	Olivgrün
ORCHID	Ochideenrosa
PINK	Pink
PURPLE	Purpur
ROYALBLUE	Königsblau
SPRINGGREEN	Frühlingsgrün
SALMON	Lachsfarben
CHARTREUSE	Hellgrün
DARKGRAY	Dunkelgrau
DARKCYAN	Grünblau
DARKGREEN	Dunkelgrün
DARKMAGENTA	Dunkellila
DARKRED	Dunkelrot
LIGHTCYAN	Hellzyan
LIGHTGRAY	Hellgrau
LIGHTYELLOW	Hellgelb
PALEGREEN	Blaßgrün
VERYLIGHTGRAY	Sehr helles Grau
VERYDARKGRAY	Sehr dunkles Grau

formatPassword (<NameOderArrayMitNamen>)

Mit dem Dienst wird anstelle eines eingetippten Zeichens ein * angezeigt. Damit können Sie die Eingabe, wie für das Passwort erforderlich, anonymisiert zeigen. <NameOderArrayMitNamen> kann ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns*(<Spalten>) benutzt werden.

format(<NameOderNamenArray>,as: <Formattyp>,with: <Formatregel>)

Mit dem Dienst wird für das Eingabefeld des Dialogelementes mit dem Namen <NameOderArrayMitNamen> die Formatierungsregel <Formattyp> festgelegt. Dabei kann <NameOderArrayMitNamen> ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Das Eingabefeld muss dabei im Gegensatz zum Dienst *format*(<NameOderArrayMitNamen>, with: <Formatregel>) nicht zuerst mit einem passendes Datum gefüllt werden.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns*(<Spalten>) benutzt werden. n.

Für <Formattyp> können Sie schreiben:

Password	Die Eingabe ist ein Passwort, das erste Zeichen aus <Formatregel> wird, wenn angegeben, als Ersatzzeichen verwendet. Ist Null als Formatregel angegeben, wird das Zeichen * verwendet.
String	Die Eingabe ist eine Zeichenkette. <Formatregel> enthält die passende Formatierungsregel. Ist Null als Formatregel angegeben, wird keine Regel verwendet.
Date	Die Eingabe ist ein Datum. <Formatregel> enthält die passende Formatierungsregel. Ist Null als Formatregel angegeben, wird dd.mm.yy verwendet.
Time	Die Eingabe ist eine Uhrzeit. <Formatregel> enthält die passende Formatierungsregel. Ist Null als Formatregel angegeben, wird hh.mm.ss verwendet.
Timestamp	Die Eingabe ist ein Datum mit Uhrzeit. <Formatregel> enthält die passende Formatierungsregel.
Number	Die Eingabe ist eine Ganzzahl. <Formatregel> enthält die passende Formatierungsregel. Ist Null als Formatregel angegeben, wird #,##0;-#,## verwendet.
Decimal	Die Eingabe ist eine Dezimalzahl. <Formatregel> enthält die passende Formatierungsregel. Ist Null als Formatregel angegeben, wird #,##0.00;-#,##.00 verwendet.

Die möglichen Formatierungsregeln sind beim Dienst *format* (<NameOderArrayMitNamen>, with: <Formatregel>) beschrieben.

format (<Name>, with: <Formatregel>, value: <Variable>)

Der Dienst überträgt in das Eingabefeld des Dialogelementes mit dem Namen <Name> den Inhalt der Variablen <Variable>. Details dazu finden Sie im Kapitel *value* (<Name>, put: <Variable>). Für die Darstellung werden die Formatierungsregeln in <Formatregel> verwendet. Details dazu finden Sie im Kapitel *format* (<NameOderArrayMitNamen>, with:

`<Formatregel>`). Diesen Dienst müssen Sie verwenden, wenn für ein Eingabefeld erst bei der Übertragung des Wertes eine Formatierungsregel festgelegt werden kann. Für alle anderen Eingabefelder können Sie die Formatierungsregeln, so erforderlich, mit dem Dienst `format(..., with: ...)` festlegen.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst `build` oder `buildWithColumns(<Spalten>)` benutzt werden. n.

`format (<NameOderNamenArray>, as: <Formattyp>)`

Der Dienst legt für das Eingabefeld des Dialogelementes mit dem Namen `<NameOderNamenArray>` die Systemformatierungsregel für `<Formattyp>` fest. Dabei kann `<NameOderNamenArray>` ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Details dazu finden Sie im Kapitel `format (<NameOderArrayMitNamen>, with: <Formatregel>)`. Mit diesem Dienst können Sie die Formatierung in einem Dialogelement ohne die Angabe eines Elementinhaltes festlegen (siehe Dienst `format (<Name>, with: <Formatregel>, value: <Variable>)`).

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst `build` oder `buildWithColumns(<Spalten>)` benutzt werden.

Für den Formattyp "String" muss der Dienst `format(<NameOderNamenArray>, as: <Formattyp>, with: <Formatregel>)` verwendet werden, da sonst die korrekte Darstellung des Inhaltes nicht sichergestellt ist. Für alle anderen Formattypen wird die Systemformatierung verwendet.

`help (<Name>, with: <Hilfetext>)`

Der Dienst bestimmt für das mit `<Name>` benannte Dialogelement den Text `<Hilfetext>` als Onlinehilfe. Dieser Text wird kurz eingeblendet, sobald der Benutzer das Element mit der Maus berührt. `<Hilfetext>` kann spezielle Formatanweisungen enthalten:

`<n>` Zeilenwechsel

`<t>` Tabulator

% Quotet das nächste Zeichen (z.B.: %< im Hilfetext wird in der Anzeige zu <)

Die Zeichen < und > müssen dabei geschrieben werden !

Damit Hilfetexte eingeblendet werden, muss die Benutzungsart von OfficeTalk auf **Beginner** oder **Normal** stehen (siehe Dokumentation Business-Process-Management, Kapitel *Einstellungen, Allgemeines*).

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst `build` oder `buildWithColumns(<Spalten>)` benutzt werden. n.

`hyperlink (<Name>)`

Der Dienst aktiviert für das mit `<Name>` benannte Dialogelement die Darstellung von Hyperlinks. Hyperlinks sind in { { und } } eingeschlossene Strings und werden dadurch Blau und unterstrichen dargestellt.

`inlineActions (<Variable>)`

Wenn für das Argument `<Variable>` der Wert `True` verwendet wird, werden die mit dem Dienst `addAction(<Anweisungen>, named: <Name>[, continuous: <Bool-Ausdruck>])` installierten Aktionen auch bei programmatischen Änderungen des Dialogelementinhaltes durch die Dienste `value (<Name>, put: <Variable>)` und `value (<Name>, select: <Variable>)` angestoßen. Mit dem Argument `False` wird das programmatische Starten dieser inline Aktionen wieder abgeschaltet.

inputItems

Der Dienst liefert die Namen aller Elemente im Dialog, die Benutzereingabe ermöglichen. Das sind Eingabefelder, Optionsfelder und Ankreuzfelder.

invalidate (<Variable>)

Der Dienst aktualisiert unverzüglich den Inhalt der mit <Variable> angegebenen Dialogelemente. Wenn z.B. die Tabelle einer Datenbank mit vielen Datensätzen aktualisiert wird und das Vortschrittergebnis mit `dialog.value (<Name>, put: <Variable>)` angezeigt werden soll, verhindern die permanenten Datenbankaktionen die sofortige Aktualisierung des Vortschrittergebnisses. Deshalb wird mit diesem Dienst nach jeder Datenbankaktion der Inhalt des Vortschrittergebnisses unverzüglich aktualisiert.

isContaining (<Variable>)

Der Dienst liefert `True`, wenn der Dialog das Dialogelement <Variable> enthält. Oder mit anderen Worten: Der Dienst liefert `True`, wenn das Dialogelement mit einem `add`-Dienst im Dialog eingefügt wurde.

isEnabled (<Variable>)

Der Dienst liefert `True`, wenn das Dialogelement <Variable> nicht gesperrt ist, ansonsten `False`. Der Dienst liefert `Null`, wenn das Dialogelement <Variable> nicht existiert.

isMandatoryMissingSilent(<Bool-Ausdruck>)

Der Dienst liefert `True`, wenn ein, mit dem Dienst *mandatory (<NameOderArrayMitNamen>)*, vorgeschriebenes Dialogelement nicht ausgefüllt ist. Wenn für <Bool-Ausdruck> `False` verwendet wird, markiert der Dienst das betroffene Dialogelement temporär mit einem Fragezeichen und informiert Sie zusätzlich mit einem Hinweisdialog.

Hinweis: Der Dienst ist nur innerhalb den Anweisungen einer *Aktionsschaltfläche* sinnvoll. Bevor vorgeschriebene Eingabewerten weiterverarbeitet, kann damit sichergestellt werden, dass die diese Eingaben dafür gemacht wurden. Ein Beispiel:

```
dialog.addActionButton("If dialog.isMandatoryMissing = False"  
    Then  
        If dialog.value("lieferdatum") > Date  
        Then  
            weitere Anweisungen...  
        End If  
    End If", named: "lieferbutton",...)
```

isMandatoryMissing

Der Dienst ist ein synonym für den Dienst *isMandatoryMissingSilent(<Bool-Ausdruck>)*.

isValidSilent(<Bool-Ausdruck>)

Der Dienst liefert `True`, wenn alle Eingaben syntaktisch richtig sind. Das betrifft Eingaben für Datums-, Zeit- und Zahlenfelder. Diese Eingaben müssen den Formatregeln entsprechen (siehe Kapitel *format (<NameOderArrayMitNamen>, with: <Formatregel>)* und nachfolgende Kapitel zur Formatierung). Wenn für <Bool-Ausdruck> `False` verwendet wird, markiert der Dienst das betroffene Dialogelement temporär mit einem Fragezeichen und informiert Sie zusätzlich mit einem Hinweisdialog.

Hinweis: Der Dienst ist nur innerhalb den Anweisungen einer *Aktionsschaltfläche* sinnvoll. Bevor betroffene Eingabewerte weiterverarbeitet werden, kann damit sichergestellt werden, dass diese Eingaben syntaktisch richtig vorliegen. Syntaktisch falsche Eingaben wer-

den durch ein kurzes Blinken des Eingabefeldes angezeigt. Der Dienst *value* (*<Name>*) liefert in diesem Fall den ursprünglichen Wert. Ein Beispiel:

```
dialog.addActionButton(
    "If dialog.isValid = True
    Then
        If dialog.value("lieferdatum") > Date
        Then
            weitere Anweisungen...
        End If
    Else
        dialog.value("lieferdatum") '= ursprüngliches Datum
    End If", named: "lieferbutton",...)
```

isValid

Der Dienst ist ein synonym für den Dienst *isValidSilent*(*<Bool-Ausdruck>*).

label (*<Name>*)

Der Dienst liefert den Bezeichner mit dem Schlüssel *<Name>* als String unter Berücksichtigung der aktuellen Bezeichnergruppe. *<Name>* ist ein String. Weitere Einzelheiten dazu siehe Dokumentation [Business-Process-Management](#), Kapitel *Adresse*.

label (*<Name>*, *ifNone*: *<Variable>*)

Der Dienst liefert den Bezeichner mit dem Schlüssel *<Name>* als String unter Berücksichtigung der aktuellen Bezeichnergruppe. *<Name>* und *<Variable>* sind ein String. Falls der Schlüssel *<Name>* nicht vorhanden ist, wird der Bezeichner *<Variable>* geliefert. Weitere Einzelheiten dazu siehe Dokumentation [Business-Process-Management](#), Kapitel *Adresse*.

label (*<Name>*, *with*: *<Variable>*)

Durch dem Dienst bestimmen Sie für die Schaltfläche mit dem Namen *<Name>* das Bild mit dem Namen *<Variable>*. Die Benennung der Schaltfläche wird durch den Dienst mit dem angegebenen Bild ersetzt. *<Variable>* kann der Dateiname eines Bildes sein, oder den vordefinierten Namen eines Bildes aus der OfficeTalk-Galerie bezeichnen. Namen und zulässige Bildformate siehe Dokumentation [Business-Process-Management](#), Kapitel *Galerie*.

label(*<Seitenname>*, *register*: *<Registernamen>*, *with*: *<Variable>*)

Durch dem Dienst erhält der Bezeichner der Registerseite *<Seitenname>* im Register *<Registernamen>* das Bild mit dem Namen *<Variable>*. *<Seitenname>* bezeichnet die sichtbare Laschenbezeichnung der Registerseite und *<Registernamen>* den internen Namen des Dialogelementes (siehe Kapitel *addRegister*(*<Registerseiten>*, *named*: *<Name>*, *left*: *<Ganzzahl>*, *top*: *<Ganzzahl>*, *width*: *<Ganzzahl>*, *height*: *<Ganzzahl>*)). *<Variable>* kann der Dateiname eines Bildes sein, oder den vordefinierten Namen eines Bildes aus der OfficeTalk-Galerie bezeichnen. Namen und zulässige Bildformate siehe Dokumentation [Business-Process-Management](#), Kapitel *Galerie*.

labelGroup (*<Name>*)

Mit dem Dienst bestimmen Sie die Bezeichnergruppe, in der mit dem Schlüssel (Argument *key*:) der Bezeichner gesucht wird. Die Bezeichnergruppe des angemeldeten Bearbeiters wird ignoriert. Weitere Einzelheiten dazu siehe Kapitel *addLabel* (*<Bezeichnung>*, *left*: *<Ganzzahl>*, *top*: *<Ganzzahl>*, *key*: *<Label>*) und Dokumentation [Business-Process-Management](#), Kapitel *Adresse*.

labelGroup

Der Dienst liefert die aktuelle Bezeichnergruppe. Das ist entweder die mit `labelGroup(...)` eingestellte Bezeichnergruppe, oder die Bezeichnergruppe des ausführenden Bearbeiters. Weitere Einzelheiten dazu siehe Kapitel *addLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>, key: <Label>)* und Dokumentation Business-Process-Management, Kapitel *Adresse*.

limit (<NameOderArrayMitNamen>, maxChar: <Variable>)

Mit dem Dienst legen Sie für das Eingabefeld des Dialogelementes mit dem Namen `<NameOderArrayMitNamen>` die maximale Anzahl der Eingabezeichen fest. Dabei kann `<NameOderArrayMitNamen>` ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Der Dienst ist nur bei Dialogelementen, die mit `addInput` oder `addText` im Dialog eingefügt wurden, sinnvoll. Bei allen anderen Eingabefeldarten hat der Dienst keine Wirkung.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst `build` oder `buildWithColumns(<Spalten>)` benutzt werden. n.

mandatory (<NameOderArrayMitNamen>)

Der Dienst benennt mit `<NameOderArrayMitNamen>` Dialogelemente, die in jedem Fall ausgefüllt werden müssen, damit der Dialog normal beendet (Dienst `accept` oder **OK**-Schaltfläche (siehe Kapitel *Schaltfläche für die normale Dialogbeendigung*)) werden kann. Fehlt die Angabe für ein hier angegebenes Dialogelement, wird das Beenden mit einer entsprechenden Meldung abgelehnt. `<NameOderArrayMitNamen>` kann ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Eine *Markierung* kann hier nicht verwendet werden, da sie keinen leeren Zustand kennt.

Hinweis: Der Dialog kann in jedem Fall mit dem Dienst `cancel` oder mit dem Fensterschließknopf abgebrochen werden.

mandatoryMessage (<Meldungstext>)

Mit dem Dienst wird der Text angegeben, mit dem eine fehlende Dialogelementangabe (siehe Dienst `mandatory (<NameOderArrayMitNamen>)`) gemeldet wird. `<Meldungstext>` kann spezielle Formatanweisungen enthalten (siehe Dienst `help (<Name>, with: <Hilfetext>)`).

notTabable (<NameOderArrayMitNamen>)

Mit dem Dienst ignorieren die, mit `<NameOderArrayMitNamen>` benannten Elemente, die TAB-Taste. D.h. Die TAB-Taste überspringt das Element. Der Dienst ist besonders für Schaltflächen sinnvoll, da diese Elemente i.d.R. mit der Maus bedient sollen werden.

optional(<NameOderArrayMitNamen>)

Der Dienst legt fest, dass das Ausfüllen der mit `<NameOderArrayMitNamen>` angegebenen Dialogelemente optional ist. Mit dem Dienst kann der Dienst `mandatory (<NameOderArrayMitNamen>)` für ein oder mehrere Dialogelemente dynamisch zur Laufzeit widerrufen werden.

page(<Seitenname>, register: <Registername>)

Der Dienst liefert den Dialog der Registerseite `<Seitenname>` im Register `<Registername>` als Systemvariable `ScriptDialog.<Seitenname>` bezeichnet die sichtbare Laschenbezeichnung der Registerseite und `<Registername>` den internen Namen des Dialogelementes (siehe Kapitel *addRegister(<Registerseiten>, named: <Name>*,

left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)). Ein Beispiel für die Anwendung des Dienstes finden Sie im Kapitel *Beispiel mit Register (on demand)*.

page(<Seitenname>, register: <Registername> , indication: <Variable>)

Mit dem Dienst kann im Bezeichner der Registerseite <Seitenname> des Registers <Registername>, abhängig von der Booleanvariable <Variable>, ein Hinweispiktogramm vorangestellt oder entfernt werden. <Seitenname> bezeichnet die sichtbare Laschenbezeichnung der Registerseite und <Registername> den internen Namen des Dialogelementes (siehe Kapitel *addRegister(<Registerseiten>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl>)*)

press(<Name>)

Mit dem Dienst wird ein Mausklick auf das Dialogelement mit dem Namen <Name> simuliert. <Name> muss eine **Accept**-, **Cancel**- oder **Aktionsschaltfläche** benennen. I.d.R. wird der Dienst innerhalb eines Anweisungsblocks des Dienstes *addAction(<Anweisungen>, named: <Name>[, continuous: <Bool-Ausdruck>])* oder *addDoubleAction(<Anweisungen>, named: <Name>)* verwendet.

put (<Name>, value: <Variable>)

Beschreibung siehe Dienst *value (<Name>, put: <Variable>)*.

progressBar(<Name>, increment: <Ganzzahl>)

Der Dienst inkrementiert den farbigen Fortschrittbalken in der mit <Name> benannten Fortschrittsanzeige. Wenn der Fortschrittbalken mit dem Argument *reverse: False* angelegt wurde, wird dadurch der farbige Balken vergrößert. Wenn der Fortschrittbalken mit dem Argument *reverse: True* angelegt wurde, wird dadurch der farbige Balken verkleinert.

properties

Der Dienst liefert die wichtigsten Eigenschaften des Dialoges als Array.

Array-element	Typ	Beschreibung
0	Integer	Die Breite des Dialoges.
1	Integer	Die Höhe des Dialoges.
2	Boolean	Die Einstellung für den Dienst <i>accept</i> (auch OK -Schaltfläche) True <i>accept</i> ist erlaubt. False <i>accept</i> ist nicht erlaubt.
3	Boolean	Die Einstellung für den Dienst <i>cancel</i> (auch Abbruch -Schaltfläche) True <i>cancel</i> ist erlaubt. False <i>cancel</i> ist nicht erlaubt.
4	String	Die Bezeichnergruppe
5	String	Der Text für die Warnung bei fehlenden vorgeschriebenen Angaben.
6	Array	Die Namen aller visuellen Dialogelemente.
7	Array	Die Namen der vorgeschriebenen Dialogelemente.
8	Array	Die Eigenschaften der visuellen Dialogelemente (siehe Kapitel

		<code>properties(<Name>)</code>
--	--	---

properties(<Name>)

Der Dienst liefert die wichtigsten Eigenschaften des mit <Name> angegebenen Dialogelementes in einem Array. Hauptsächlich sind die Eigenschaften von Dialogelementen, bei denen Tastatureingaben oder Mausaktivitäten vom Benutzer erwartet werden, erfragbar. Ein Arrayelement ist `Null`, wenn die Eigenschaft nicht konfiguriert ist, oder wenn das Dialogelement diese Eigenschaft nicht unterstützt.

Array-element	Typ	Beschreibung
0	String	Der Name des Dialogelementes. Der Name eines Bezeichner lautet " <code><Name>:Label</code> ". Der Name der OK -Schaltfläche lautet " <code>Accept</code> ". Der Name der Abbruch -Schaltfläche lautet " <code>Cancel</code> ".
1	String	Der Type des Dialogelementes: <div> <div>Tree</div> <div>Baumlistfeld</div> </div> <div> <div>List</div> <div>Listfeld</div> </div> <div> <div>Table</div> <div>Tabelle</div> </div> <div> <div>Combo</div> <div>Auswahllistfeld</div> </div> <div> <div>SpinInput</div> <div>Eingabefeld mit Pfeiltasten</div> </div> <div> <div>Input</div> <div>Eingabefeld</div> </div> <div> <div>Text</div> <div>Textfeld</div> </div> <div> <div>Check</div> <div>Ankreuzfeld</div> </div> <div> <div>Radio</div> <div>Optionsfeld</div> </div> <div> <div>Divider</div> <div>Trennlinie</div> </div> <div> <div>Resizer</div> <div>Größenänderer</div> </div> <div> <div>Button</div> <div>OK-, Abbruch- oder Aktionsschaltfläche</div> </div> <div> <div>Register</div> <div>Register</div> </div> <div> <div>Label</div> <div>Bezeichner</div> </div> <div> <div>Group</div> <div>Gruppenrahmen</div> </div> <div> <div>Any</div> <div>Irgend ein anderes Dialogelement</div> </div>
2	Boolean	<div> <div>True</div> <div>Die Angabe im Dialogelement ist vorgeschriebene.</div> </div> <div> <div>False</div> <div>Die Angabe im Dialogelement ist optionale.</div> </div> Ein Dialogelement vom Typ <code>Option</code> ist vorgeschrieben, wenn seine Gruppe vorgeschrieben ist.
3	Boolean	<div> <div>True</div> <div>Das Dialogelement ist gesperrt.</div> </div> <div> <div>False</div> <div>Das Dialogelement ist nicht gesperrt.</div> </div>
4	Integer	Die Limitierung der Anzahl für die Zeicheneingabe (z.B. Input , SpinInput).
5	String	Die Art der Formatierung im Dialogelement: <div> <div>password</div> <div>Formatierung für Passwort</div> </div> <div> <div>number</div> <div>Formatierung für Gleitpunktzahl</div> </div>

		number	Formatierung für Gleitpunktzahl
		decimal	Formatierung für Zahl/Betrag
		date	Formatierung für Datum
		time	Formatierung für Zeit
		timestamp	Formatierung für Zeitstemple
		string	Formatierung für Text
6	String	Das Muster für die Formatierung im Dialogelement.	

rowAndColumn (<Name>)

Der Dienst liefert die gewählte oder die aktivierte (Doppelklick mit der Maus) Zeile und Spalte des Tabellenelementes mit dem Namen <Name>. Die Ergebnis wird als Array (<Nummer der Zeile>, <Nummer der Spalte>) geliefert. Die Zählbasis von Nummer ist 1. Nummer bezeichnet das Element, auf das mit der Maus geklickt wurde.

Hinweis: Der Dienst sollte nur in Aktionsblöcken verwendet werden (siehe Kapitel *Aktionsschaltfläche*, *Aktion bei Doppelklick* und *Aktion bei Inhaltsänderung*).

scale (<NameOderArrayMitNamen>, by: <Faktor>)

Der Dienst skaliert die Schriftgröße des Bezeichners oder die Schriftgröße der Texteingabe der mit <NameOderArrayMitNamen> benannten Dialogelemente. <Faktor> muss ein positiver numerischer Wert sein. Mit dem Faktor 0.99 oder kleiner wird die Schrift verkleinert. Mit dem Faktor 1.01 oder größer wird die Schrift vergrößert.

scrollBars(<Variable>)

Mit dem Booleanwert True für <Variable> erhält der Dialog horizontale und vertikale Bildlaufleisten. Wenn die eingestellte Arbeitsfläche des Dialoges kleiner als die erforderliche ist (siehe Kapitel *width(<Ganzzahl>, height: <Ganzzahl>)*), um alle Dialogelemente darzustellen, können Sie mit der horizontalen, und/oder vertikalen Bildlaufleiste die Arbeitsfläche zu den noch nicht sichtbaren Dialogelementen verschieben.

select (<Name>, value: <Variable>)

Beschreibung siehe Dienst *value (<Name>, select: <Variable>)*.

selected (<Name>)

Der Dienst liefert bei listenartigen Dialogelementen (*Listfeld*, *Auswahlliste*, *Tabelle*, usw.) mit dem Namen <Name> den zuletzt gewählten Eintrag. Damit kann vor allem bei Listfeldern und Tabellen mit Mehrfachauswahl die zuletzt gewählte Zeile ermittelt werden (siehe auch Kapitel *value (<Name>)*). Bei Eingabefeldern liefert der Dienst den markierten Textteil des Eingabefeldes.

selection (<Name>)

Der Dienst liefert bei listenartigen Dialogelementen (*Listfeld*, *Auswahlliste*, *Tabelle*, usw.) mit dem Namen <Name> den gewählten Eintrag (siehe auch Kapitel *value (<Name>)*). Bei Listfeldern und Tabellen mit Mehrfachauswahl liefert der Dienst alle gewählten Zeilen in einem Array. Bei Eingabefeldern liefert der Dienst den markierten Textteil des Eingabefeldes.

selectionIndex (<Name>)

Der Dienst liefert bei listenartigen Dialogelementen (*Listfeld*, *Auswahlliste*, *Tabelle*, usw.) mit dem Namen <Name> den Index des ausgewählten Elementes. Wenn das Listfeld oder die Tabelle Mehrfachauswahl unterstützen, liefert der Dienst ein `Array` mit den Zeilennummern der gewählten Zeilen. In einem Eingabefeld liefert der Dienst die Position der Schreibmarke (Cursor) im Eingabefeld. Die Zählbasis des Index ist 1.

selectItems

Der Dienst liefert die Namen aller Elemente im Dialog, die eine Benutzerauswahl ermöglichen. Das sind Auswahllistfelder, Listfelder, usw.

shortcut (<Bezeichnername>, for: <Zielname>)

Der Dienst bestimmt für das Tastaturkürzel (Buchstabe mit vorangestelltem **&**) in dem Bezeichner mit dem Namen <Bezeichnername> das Dialogelement mit dem Namen <Zielname> als Ziel. Die Tastenkombination **Alt+<Tastaturkürzel>** aktiviert das als Ziel angegebene Dialogelement. Eingabeartige Dialogelemente (*Eingabefeld*, *Listfeld*, usw.) erhalten die Schreibmarke, wählbare Dialogelemente (*Markierung*, *Option*, usw.) werden ausgewählt.

Hinweis: Der Dienst sollte nur in Aktionsblöcken verwendet werden (siehe Kapitel *Aktionsschaltfläche*, *Aktion bei Doppelklick* und *Aktion bei Inhaltsänderung*).

show (<NameOderArrayMitNamen>)

Der Dienst macht das mit <NameOderArrayMitNamen> benannte Dialogelement, das mit dem Dienst `hide()` ausgeblendet wurde, sichtbar. <NameOderArrayMitNamen> kann ein einzelner Name eines Dialogelementes oder ein `Array` mit mehreren Elementnamen sein.

Hinweis: Der Dienst sollte nur in Aktionsblöcken verwendet werden (siehe Kapitel *Aktionsschaltfläche*, *Aktion bei Doppelklick* und *Aktion bei Inhaltsänderung*).

value (<Name>)

Der Dienst liefert den Inhalt des Dialogelementes mit dem Namen <Name>. Benennt <Name> den Namen einer Optionsgruppe, liefert der Dienst die gewählte Option. Bei listenartigen Dialogelementen (*Listfeld*, *Auswahlliste*, *Tabelle*, usw.) liefert der Dienst *selection (<Name>)* den gewählten Eintrag.

Hinweis: Wenn das Dialogelement ein *Listfeld* oder eine *Auswahlliste* bezeichnet, und das Dialogelement *asAssociations* enthält, liefert der Dienst den Schlüssel des gewählten Eintrages. Voraussetzung dafür ist, dass das Dialogelement beim Erstellen (*Auswahlliste* und *Listfeld*), oder bei *value (<Name>, put: <Variable>)* mit einem `Array` mit *asAssociations* bestückt wurde.

values (<Name>)

Der Dienst liefert bei listenartigen Dialogelementen (*Listfeld*, *Auswahlliste*, *Tabelle*, usw.) mit dem Namen <Name> alle Einträge.

value (<Name>, expand: <Variable>)

Der Dienst expandiert in dem Baumlistdialogelement mit dem Namen <Name> die Zeile, die mit dem Inhalt von <Variable> übereinstimmt. Die Zeile wird abschließend markiert. Alle übergeordneten Zeilen der Zeile <Variable> werden dazu expandiert.

value (<Name>, put: <Variable>)

Der Dienst überträgt in das Dialogelement mit dem Namen <Name> den Inhalt der Variablen <Variable>. Sollen mehrere Zeilen in ein Listfeld eingefügt werden, muss <Variable> ein Array mit den Zeilen sein. Zum Übertragen des Inhalts einer Tabelle siehe Kapitel *addTable* (<Zeilen>, columns: <Überschrift>, named: <Name>, left: <Ganzzahl>, top: <Ganzzahl>, width: <Ganzzahl>, height: <Ganzzahl> [, multiSelection: <Bool-Ausdruck>]).

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns*(<Spalten>) benutzt werden.

Wenn das Dialogelement eine Formatregel enthält (siehe *format* (<NameOderArrayMitNamen>, with: <Formatregel>)), darf für das Argument *put* : nur ein kompatibler Wert verwendet werden, ansonsten wird die Ausführung des Dienstes mit der Meldung **Unzulässiges Argument...** geblockt. Null wird immer als kompatibler Wert akzeptiert.

Der Dienst führt für eine mit dem Dienst *disable* (<NameOderArrayMitNamen>) gesperrte *Auswahlliste* zu der Fehlermeldung **Im Kontext unzulässiger Dienst "value:put: in der" Anweisung....**

Die Dialogelemente *Listfeld* und *Auswahlliste* können auch mit *asAssociations* belegt werden. Zu den Besonderheiten beim Ermitteln des ausgewählten Eintrages siehe Kapitel *value* (<Name>).

value (<Name>, replace: <Zeilennummer>, with: <Zeile>)

Der Dienst ersetzt in der Tabelle oder in dem Listfeld mit dem Namen <Name> die Zeile mit der Nummer <Zeilennummer> mit der neuen Zeile <Zeile>. Die Zählbasis für <Zeilennummer> ist dabei 1. Für Tabellen muss <Zeile> ein Array mit den Spalten der neuen Zeile sein. Bei Listfeldern ist <Zeile> i.d.R. eine Zeichenkette. Um die Änderung sofort sichtbar zu machen, muss anschließend der Dienst *invalidate* (<Variable>) gerufen werden. Bei Tabellen und Listfeldern mit sehr vielen Zeilen bietet dieses Vorgehen für das Aktualisieren einzelner Zeilen einen grossen Zeitvorteil. Bei allen anderen Dialogelementen hat der Dienst keine Auswirkung.

Hinweis: Wenn das Dialogelement eine Formatregel enthält (siehe *format* (<NameOderArrayMitNamen>, with: <Formatregel>)), darf für das Argument *with* : nur ein kompatibler Wert verwendet werden, ansonsten wird die Ausführung des Dienstes mit der Meldung **Unzulässiges Argument...** geblockt. Null wird immer als kompatibler Wert akzeptiert.

value (<Name>, select: <Variable>)

Der Dienst selektiert in listenartigen Dialogelementen (*Listfeld*, *Auswahlliste*, *Tabelle*, usw.) mit dem Namen <Name> die Zeile, die mit dem Inhalt von <Variable> übereinstimmt. Ist in dem Dialogelement **Mehrfachauswahl** (*Listfeld* oder *Tabelle*) eingestellt, müssen Sie für <Variable> `Array(<element>[, ...])` verwenden, um ein oder mehrere Zeilen gleichzeitig zu selektieren. <element> ist ein Element aus der Liste (erstes Argument in *Listfeld* oder *Tabelle*).

Hinweis: Wenn das Dialogelement *asAssociations* enthält, muss <Variable> den Schlüssel der Association enthalten.

value(<Name>, selectedIndex: <Zeilennummer>)

Der Dienst selektiert in listenartigen Dialogelementen (*Listfeld*, *Auswahlliste*, *Tabelle*, usw.) mit dem Namen <Name> die n-te mit <Zeilennummer> angegebene Zeile. Ist in dem Dialogelement **Mehrfachauswahl** (*Listfeld* oder *Tabelle*) eingestellt, müssen Sie für

`<Zeilennummer> Array(<Zeilennummer>[,...])` verwenden, um ein oder mehrere Zeilen gleichzeitig zu selektieren.

withIcons (<Name>)

Mit dem Dienst werden die Einträge in dem mit `<Name>` benannten *Listfeld* oder *Auswahlliste* von ihrem Piktogramm angeführt. Für OfficeTalk spezifische Listenelemente (z.B. Bearbeiter), ist es das jeweilige Piktogramm des OfficeTalk-Elementes.

Hinweis: Wenn mit einem neuen Listeninhalt von der Anzeige ohne Piktogramm auf Anzeige mit Piktogramm gewechselt werden soll, muss zuerst das Listfeld mit dem neuen Inhalt versehen werden.

```
dialog.value(<Name>, put: <Zeilenarray>)
dialog.withIcons(<Name>)
```

Wenn diese Anweisungsreihenfolge vertauscht wird, fehlt der horizontale Abstand der Piktogramme am linken Rand.

withIcons (<Name>, stringWith: <Stringoperator>, iconWith: <Iconoperator>)

Siehe *withIcons (<Name>)*. Der Dienst ist nur für Sonderfälle sinnvoll und erfordert genaueste Systemkenntnisse von OfficeTalk. `<Stringoperator>` benennt den Operator, der den Text des Elementes liefert. `<Iconoperator>` benennt den Operator, der das Piktogramm des Elementes liefert.

withoutIcons (<Name>)

Mit dem Dienst werden die führenden Piktogramme in dem mit `<Name>` benannten benannten *Listfeld* oder *Auswahlliste* entfernt. Der Dienst widerruft den Dienst *withIcons (<Name>)*.

withSeconds (<NameOderArrayMitNamen>)

Mit dem Dienst legen Sie für das Eingabefeld des Dialogelementes mit dem Namen `<NameOderArrayMitNamen>` fest, dass die Uhrzeit mit Sekunden angezeigt wird. Dabei kann `<NameOderArrayMitNamen>` ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Das Textfeld muss bereits eine Zeit enthalten, damit das Format mit diesem Dienst eingestellt werden kann. Das heißt, in der Anweisungsfolge

```
dialog.addInput( "", named: "Datum", left: ... )
dialog.withSeconds( "Datum" )
```

wird durch die zweite Anweisung **nicht** die Sekundenbearbeitung aktiviert, weil in der vorangegangenen Anweisung kein Zeitwert übergeben wurde. Ohne die Anwendung dieses Dienstes werden Uhrzeiten als HH:MM angezeigt.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns(<Spalten>)* benutzt werden.

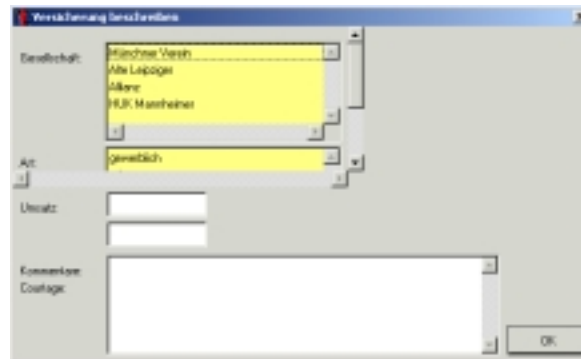
withoutSeconds (<NameOderArrayMitNamen>)

Mit dem Dienst legen Sie für das Eingabefeld des Dialogelementes mit dem Namen `<NameOderArrayMitNamen>` fest, dass die Uhrzeit ohne Sekunden angezeigt wird. Dabei kann `<NameOderArrayMitNamen>` ein einzelner Name eines Dialogelementes oder ein Array mit mehreren Elementnamen sein. Das Textfeld muss bereits eine Zeit enthalten, damit das Format mit diesem Dienst eingestellt werden kann.

Hinweis: Der Dienst darf, wenn der Dialog mit dem Layoutmanager konstruiert wird, erst nach dem Dienst *build* oder *buildWithColumns(<Spalten>)* benutzt werden.

Bildlaufbereiche für einzelne Dialogelemente definieren

Ähnlich wie für den gesamten Dialog, können auch einzelne Elemente eines Dialoges mit Bildlaufleisten versehen werden. Das bietet sich an, wenn die Dialogelemente größer, als der Dialog selbst sind. Diese Funktionalität wird Komponierten (**composite**) genannt.



Beim Erstellen der Dialogelemente (siehe Kapitel *Dienste zum Einfügen von Elementen*) wird vor dem ersten Dialogelement der Komposition der Dienst `startComposite`, und nach dem letzten Dialogelement der Komposition der Dienst `endCompositeLeft(<Variable>, top: <Variable>, width: <Variable>, height: <Variable>, vertical: <Boolean>, horizontal: <Boolean>)` oder der Dienst `endCompositeWidth(<Variable>, height: <Variable>, vertical: <Boolean>, horizontal: <Boolean>)` verwendet. Die Ausmaße des Bildlaufbereiches werden durch die letzteren Dienste bestimmt. Die Dialogelemente innerhalb der Komposition können mit den Bildlaufleisten gescrollt werden.

Hinweis: Kompositionen werden in der **Workbench** zwar angezeigt, aber in die Komposition können keine zusätzlichen Dialogelemente eingefügt werden. Außerdem kann die Größe der Komposition in der **Workbench** nicht geändert werden. Ein Ändern der Komposition kann nur im Quelltext des Makros erfolgen.

startComposite

Der Dienst startet eine Komposition. Alle nachfolgend eingefügten Dialogelemente sind Bestandteil der Komposition.

`endCompositeLeft(<Variable>, top: <Variable>, width: <Variable>, height: <Variable>, vertical: <Boolean>, horizontal: <Boolean>)`

Die Komposition wird beendet. Mit den Argumenten für `<Variable>` werden die Position und die Größe des Bereiches bestimmt. Mit den Argumenten für `<Boolean>` werden die zu verwendenden Bildlaufleisten bestimmt. Mit den Positions- und Größenangaben werden die Ausmaße der Bildlaufleisten bestimmt.

`endCompositeWidth(<Variable>, height: <Variable>, vertical: <Boolean>, horizontal: <Boolean>)`

Die Komposition wird beendet. Mit den Argumenten für `<Variable>` wird die Größe des Bereiches bestimmt. Die Position der Komposition wird durch das erste Dialogelement der Komposition bestimmt. Mit den Argumenten für `<Boolean>` werden die zu verwendenden Bildlaufleisten bestimmt. Mit den Größenangaben werden die Ausmaße der Bildlaufleisten bestimmt.

Hinweis: Der Dienst kann nur für Kompositionen, die am linken Dialogrand liegen, verwendet werden. Für Kompositionen, die in der Dialogmitte, oder am rechten Rand des Dialoges liegen, muss der Dienst `endCompositeLeft(<Variable>, top: <Variable>, width: <Variable>, height: <Variable>, vertical: <Boolean>, horizontal: <Boolean>)` verwendet werden.

Strategien für das Konstruieren eines Dialoges

Im Dialog können Sie die Elemente (*Auswahlliste, Eingabefeld, Listfeld, Markierung, Option, usw.*) auf verschiedene Arten einfügen. Sie können die Position der Elemente selbst, oder vom integrierten Layoutmanager bestimmen lassen. Nur wenn Sie besondere Anforderungen an das Dialoglayout haben, müssen Sie die Position der Elemente selbst errechnen. Ansonsten kann Ihnen bei diesen Berechnungen der Layoutmanager behilflich sein.

Einfügen mit direkter Positionsangabe.

Für jede Elementart stehen entsprechende Dienste mit direkter Positionsangabe zur Verfügung. Damit können Sie die Zielposition der Elemente exakt kontrollieren. Durch die manuelle Berechnung ist diese Vorgehensweise aber auch mit mehr Aufwand verbunden. Sie müssen die Dienste in folgender Reihenfolge verwenden:

<code>Dim dialog As Joops.Scripting.ScriptDialog</code>	'Variable deklarieren
<code>dialog = New Joops.Scripting.ScriptDialog</code>	'Dialog erzeugen
<code>dialog.add.....</code>	'Dialogelemente mit 'Positionsangabe einfügen
<code>dialog.width(...)</code>	'Dialogbreite festlegen
<code>dialog.height(...)</code>	'Dialoghöhe festlegen
<code>dialog.title(...)</code>	'Dialogtitel bestimmen
<code>dialog.open</code>	'Dialog öffnen

Einfügen der Elemente mit dem Layoutmanager

Der integrierte Layoutmanager übernimmt für Sie die Berechnung der Elementposition. Dabei wird die Benennung links vom zugehörigen Element gesetzt. Die Elemente werden in Spalten organisiert. Radiobuttons und Checkboxes werden innerhalb der Spalte mit den Eingabefeldern auf die gleiche Spalte gestellt. Die *Schaltfläche für die normale Dialogbeendigung* und die *Schaltfläche für den Dialogabbruch* werden am unteren Rand des Dialoges horizontal angeordnet. Sie müssen die Dienste in folgender Reihenfolge verwenden:

<code>Dim dialog As Joops.Scripting.ScriptDialog</code>	'Variable deklarieren
<code>dialog = New Joops.Scripting.ScriptDialog</code>	'Dialog erzeugen
<code>dialog.add.....</code>	'Dialogelemente ohne 'Positionsangabe einfügen
<code>dialog.title(...)</code>	'Dialogtitel bestimmen
<code>dialog.build</code>	'einspaltigen Dialog konstruieren
oder	
<code>dialog.buildWithColumns(...)</code>	'mehrspaltigen Dialog konstruieren
<code>dialog.open</code>	'Dialog öffnen

Sehen Sie dazu auch das *Beispiel mit mehrspaltiger Dialogerstellung durch dem Layoutmanager* für die mehrspaltige Dialogerstellung mit dem Layoutmanager (Kapitel *Elemente in Spalten einfügen*).

Elemente in Spalten einfügen

Der Dienst `buildWithColumns(<Spalten>)` und seine Varianten fügt die Elemente im Dialog in Spalten ein. Die Aufteilung und Position der einzelnen Elemente übernimmt dabei der integrierte Layoutmanager. Voraussetzung dafür ist, dass die Elemente mit den `add-`Diensten *ohne* Positionsangaben im Dialog eingefügt wurden. Nachfolgend sehen Sie den Quelltext und als Ergebnis den Dialog mit der Elementanordnung durch den Layoutmanager in zwei Spalten.

```
Dim dialog As Joops.Scripting.ScriptDialog

dialog = New Joops.Scripting.ScriptDialog
dialog.title( "Die Überschrift" )
dialog.addInput("1. Zeile", named: "1", width: 100, label: "1.
Zeile:")
dialog.addInput("200. Zeile", named: "2", width: 100, label: "200.
Zeile:")
dialog.addGroupBox(Array("radio1", "radio2", "radio3"), named: "Ra-
dio", width: 150)
dialog.addRadioButton("Radio-1", named: "radio1", group: "group")
dialog.addRadioButton("Radio-2", named: "radio2", group: "group")
dialog.addRadioButton("Radio-3", named: "radio3", group: "group")
dialog.addInput("3. Zeile", named: "3", width: 100, label: "3.Zeile:")
dialog.addInput("400000. Zeile", named: "4", width: 100, label:
"400000. Zeile:")
dialog.addGroupBox(Array("check1", "check2", "check3"),named: "Check")
dialog.addCheckBox("Check me-1", named: "check1")
dialog.addCheckBox("Check me-2", named: "check2")
dialog.addCheckBox("Check me-3", named: "check3")
dialog.addInput("5 .Zeile", named: "5", width: 100,label: "5. Zeile:")
dialog.addSpinInput( Date, named: "date", width: 100, label:"Datum:" )
dialog.addText( "der Text", named: "text", width: 200, height: 50, la-
bel: "Text:")
dialog.addList( Array(1,2), named: "array", width: 100, height: 50,
horizontalScroll: True, label: "Array:" )
dialog.addComboBox( "Choice", choices: Array("Choice", "Choice-2",
"Choice-3"), named: "combo", width: 100, label: "Combobox:" )
dialog.addAcceptButton
dialog.addCancelButton
dialog.addActionButton("MsgBox("Hallo-1")",named:"test-1",width:100)
dialog.addActionButton("MsgBox("Hallo-2")",named:"test-2",width:100)
dialog.buildWithColumns(2, startAt: 50, buttonsOffset: 50)
dialog.open
Return "Autobuild"
```



Mehrspaltigen Dialog durch Layoutmanager erstellt

Quadratische Erstellung des Dialoges

Der Dienst *build* und seine Varianten verteilen die Elemente auf Spalten und versucht dabei den Dialog quadratisch auszurichten. Voraussetzung dafür ist, dass die Elemente mit den add-Diensten *ohne* Positionsangaben im Dialog eingefügt wurden.

Erstellung des Dialoges mit abweichendem Rand

Wenn Sie die Positionen der Elemente durch den Layoutmanager festlegen lassen wollen, und das erste Element eine Gruppenbox ist, wird der durch den Layoutmanager festgelegte obere Rand zu klein sein. Die Benennung der Gruppenbox würde visuell am oberen Dialogrand „kleben“. Um das zu verhindern, können Sie dem Layoutmanager den oberen Rand mitteilen.

In dem Beispiel ist das erste Dialogelement eine Gruppenbox. Sie muss etwas nach unten versetzt werden (`dialog.buildWithColumns(1, startAt: 15)`). Des weiteren werden für ein gutes Design die letzten drei Radiobuttons mit direkter Positionsangabe (`dialog.addSpinInput(..., named: ..., left: ..., top: ..., width: ..., readOnly: ...)`) eingefügt. Ohne würden alle sechs Radiobuttons untereinander stehen, was nicht gerade beeindruckend aussieht. Auch die Uhrzeit wird mit direkter Positionsangabe neben das Datumsfeld eingefügt (Beschreibung zum Einfügen gemischter Elementen siehe nächstes Kapitel *Erstellung des Dialoges mit gemischten Elementen*).

NächsteAktivität

```
Dim dialog As ScriptDialog
```

```
Dim grp As String
```

```
grp = "nächste Aktivität"
```

```
dialog = New Joops.Scripting.ScriptDialog
```

```
dialog.title( "Nachfassen der letzten Aktivität" )
```

```
dialog.addRadioButton( "Auftrag abgegeben", named: "auf", group: grp )
```

```
dialog.addRadioButton( "Wiedervorlage", named: "wieder", group: grp )
```

```
dialog.addRadioButton( "Präsentation", named: "präsent", group: grp )
```

```
dialog.addSpinInput(Date,named:"nextDate",width:100,label:"am:",  
readOnly: False )
```

```
dialog.addText("", named: "comment", width: 400, height: 200,  
label: "Bemerkung:" )
```

```
dialog.addActionButton( "Call Erfassen.Kontakt ( False )",  
named: "Kontakt", width: 100 )
```

```

dialog.addActionButton( "stepscheduler.openHistory",
                        named: "Historie", width: 100 )

dialog.addActionButton( "If IsNull( dialog.value( grp ) ) = True
                        Then MsgBox( ""Bitte nächste Aktivität wählen"" )
                        Else dialog.accept
                        End If ", named: "OK", width: 100, default: True )

dialog.buildWithColumns( 1, startAt: 15)

dialog.addRadioButton( "Evaluation senden", named: "evaluation",
group: grp, left: 200, top: 25 )

dialog.addRadioButton( "Infomaterial senden", named: "information",
group: grp, left: 200, top: 60 )

dialog.addRadioButton( "nicht Interessiert", named: "nichtinteres-
siert", group: grp, left: 200, top: 95 )

dialog.addGroupBox( Array( "auftrag", "wiedervorlage", "präsentation",
"evaluation", "information", "nichtinteressiert" ), named: "nA" )

dialog.addLabel( "um:", left: 185, top: 131 )

dialog.addSpinInput( Time, named: "nextTime", left: 210, top: 126,
width: 80, readOnly: False )

dialog.limit( "comment", maxChar: step.maxCharComment - 20 )

dialog.disableCancel

dialog.open

```

Erstellung des Dialoges mit gemischten Elementen

Elemente mit und ohne Positionsangabe können in einem Dialog gemischt verwendet werden. In Sonderfällen kann dies auch erforderlich sein. Damit können Sie einen Teil der Elemente automatisch und einen Teil selbst positionieren. Dabei haben die durch den Layoutmanager errechneten Positionen Vorrang. Sie müssen sich also mit den Positionsangaben nach den Vorgaben des Layoutmanagers orientieren.

Beispiel mit Elementen mit und ohne Positionsangaben

```

Dim dialog As ScriptDialog
Dim answer As Boolean

dialog = New Joops.Scripting.ScriptDialog

dialog.title( "The test" )

dialog.addInput( "*", named: "Pattern", width: 200, label: "Pattern:" )

dialog.addList( Array( "a", "b", "c" ), named: "List", rightFraction:
1, topOffset: 24, bottomFraction: 0.5 )

dialog.addAcceptButton

dialog.addCancelButton

dialog.addCheckBox( "Check me", named: "Check1", left: 100, top: 200 )

dialog.addCheckBox( "Check me", named: "Check2", left: 100, top: 230 )

dialog.addRadioButton( "Option-1", named: "option1", group: "group",
left: 100, top: 280 )

dialog.addRadioButton( "Option-2", named: "option2", group: "group",
left: 100, top: 310 )

```

```
dialog.buildWithColumns(1, buttonsOffset: 300)
dialog.value( "List", select: "a" )
dialog.value( "group", select: "option2" )
dialog.width( 300 )
dialog.height( 380 )
answer = dialog.open
MsgBox( answer )
MsgBox("Group: " & dialog.value("group" ))
```

Beispiel mit Internet Explorer als Dialogelement

```
Library "$(VISUALWORKS)\Library\Microsoft InternetExplorer.pcl"

Dim dialog As ScriptDialog
Dim browser As SmallCOM.SHDocVw.WebBrowserWidgetSpec

browser = New SmallCOM.SHDocVw.WebBrowserWidgetSpec
dialog = New Joops.Scripting.ScriptDialog
dialog.title("Der Browser")
dialog.addComponent( browser, named: "browser", left: 0, top: 0,
                    width: 300, height: 390 )
dialog.addActionButton( "dialog.activeXNamed( ""browser""
).navigate2(""www.joops.com"")",named: "Browse", left: 50, top: 400 )
dialog.width( 300 )
dialog.height( 450 )
dialog.open
```

Beispiel mit mehrspaltiger Dialogerstellung durch dem Layoutmanager

```
Dim dialog As Joops.Scripting.ScriptDialog
dialog = New Joops.Scripting.ScriptDialog
dialog.title( "Die Überschrift" )
dialog.addInput("", named: "1", width: 100, label: "1. Zeile:")
dialog.addInput("", named: "2", width: 100, label: "2. Zeile:")
dialog.addRadioButton("Radio-1", named: "radio1", group: "group")
dialog.addRadioButton("Radio-2", named: "radio2", group: "group")
dialog.addRadioButton("Radio-3", named: "radio3", group: "group")
dialog.addInput("", named: "3", width: 100, label: "3. Zeile:")
dialog.addInput("", named: "4", width: 100, label: "4. Zeile:")
dialog.addCheckBox("Check me-1", named: "Check me-1")
dialog.addCheckBox("Check me-2", named: "Check me-2")
dialog.addCheckBox("Check me-3", named: "Check me-3")
dialog.addInput("", named: "5", width: 100, label: "5. Zeile:")
dialog.addSpinInput( Date, named: "date", width: 100, label:"Datum:")
```

```
dialog.addText("",named: "text", width: 200, height: 50, label:
"Text:")

dialog.addList( Array(1,2), named: "array", width: 100, height: 50,
horizontalScroll: True, label: "Array:" )

dialog.addComboBox( "Choice", choices: Array("Choice-1", "Choice-2",
"Choice-3"), named: "combo", width: 100, label: "Combobox:" )

dialog.addActionButton("MsgBox("Hallo")",named:"test-1", width: 50)
dialog.addActionButton("MsgBox("Hallo")",named:"test-2", width: 50)
dialog.addAcceptButton
dialog.addCancelButton
dialog.buildWithColumns(2)
dialog.open
```

Beispiel mit Tabelle

```
Table
Dim dialog As Joops.Scripting.ScriptDialog
Dim records As Array
Dim columns As Array

records = Array(Array( "two", "three", "four" ), Array( "five",
"six", "seven"))

columns = Array(Array("Erste Spalte", 100), Array("Zweite Spalte",
90), Array("Dritte Spalte", 110))

dialog = New Joops.Scripting.ScriptDialog
dialog.title( "The table with some rows" )
dialog.addLabel( "Die Tabelle", left: 5, top: 10 )
dialog.addTable(records, columns: columns, named: "table", left: 5,
top: 15, width: 300, height: 110)
dialog.addAction( "If dialog.selection("table") <> Null Then MsgBox(
"Row: " & dialog.selection("table") & " selected") End If",
named: "table" )
dialog.value("table", select: records(1))
dialog.addAcceptButtonLeft( 225, top: 130 )
dialog.addCancelButtonLeft( 130, top: 130 )
dialog.width( 310 )
dialog.height( 160 )
dialog.open
Return dialog.selection( "table" )
```

Beispiel mit Register

```
Kundendialog(Kundennummer As Integer)
'Der Dialog zeigt die Stammdaten eines Kunden und seine Umsätze
'Stammdaten und Umsätze sind in zwei Registerseiten aufgeteilt
```

```
'Die Registerseiten werden zuerst mittels Call... erstellt

Dim dialog As Joops.Scripting.ScriptDialog
Dim stammdatenDialog As Joops.Scripting.ScriptDialog
Dim umsatzDialog As Joops.Scripting.ScriptDialog

stammdatenDialog = Call Kunde.Stammdaten(Kundennummer)
umsatzDialog = Call Kunde.Umsatz(Kundennummer)
dialog = New Joops.Scripting.ScriptDialog
dialog.title( "Umsatz des Kunden" )
dialog.addRegister (Array(Array( "Stammdaten", stammdatenDialog),
                             Array( "Verkäufe", umsatzDialog)),
                    named: "register", left: 5, top: 15, width: 300, height: 110)
dialog.addAcceptButtonLeft( 225, top: 130 )
dialog.addCancelButtonLeft( 130, top: 130 )
dialog.width( 310 )
dialog.height( 160 )
accepted = dialog.open
If accepted = True
Then
    Call Accept.Stammdaten(stammdatenDialog) ' Übernahme der Stammdaten
    Call Accept.Umsatz(umsatzDialog)         ' Übernahme der Umsatzwerte
End If
Return accepted                          ' Rückgabe der gewählten Schaltfläche
' (True ist OK oder False ist Abbruch)
```

Beispiel mit Register (on demand)

```
Kundendialog(Kundennummer As Integer)

'Der Dialog zeigt die Stammdaten eines Kunden und seine Umsätze
'Stammdaten und Umsätze sind auf zwei Registerseiten aufgeteilt
'Die Registerseiten werden on demand (Mausklick auf die Registerlasche
' durch den Anwender) erstellt

Dim dialog As Joops.Scripting.ScriptDialog
...
dialog = New Joops.Scripting.ScriptDialog
dialog.title( "Umsatz des Kunden" )
dialog.addRegister(Array(Array( "Stammdaten",
                                "Kunde.Stammdaten(Kundennummer) " ),
                          Array( "Verkäufe", "Kunde.Umsatz(Kundennummer) " )),
                    named: "register", left: 5, top: 15, width: 300, height: 110)
dialog.addAcceptButtonLeft( 225, top: 130 )
dialog.addCancelButtonLeft( 130, top: 130 )
dialog.width( 310 )
```



```
dialog.height( 160 )
accepted = dialog.open
If accepted = True
Then
    Call Accept.Stammdaten(dialog.page("Stammdaten",
                                     register: "register")) 'Übernahme der Stammdaten
    Call Accept.Umsatz(dialog.page("Verkäufe", register: "register"))
                                     'Übernahme der Umsatzwerte
End If
Return accepted '(True ist OK oder False ist Abbruch)
```

Makro Kunde.Stammdaten

```
Stammdaten(Kundennummer As Integer)
'Der Dialog zeigt die Stammdaten eines Kunden
Dim dialog As Joops.Scripting.ScriptDialog
dialog = New Joops.Scripting.ScriptDialog
dialog.registerPage(True) ' Dialog ist eine Registerseite
dialog.width( 250 )
dialog.height( 140 )
Return dialog
```

Makro Kunde.Umsatz

```
Umsatz(Kundennummer As Integer)
'Der Dialog zeigt den Umsatz eines Kunden

Dim dialog As Joops.Scripting.ScriptDialog
dialog = New Joops.Scripting.ScriptDialog
dialog.registerPage(True) ' Dialog ist eine Registerseite
...Umsätze anhand Kundennummer ermitteln und in den Dialogelementen
einsetzen...
dialog.width( 250 )
dialog.height( 140 )
Return dialog
```

Makro Accept.Stammdaten

```
Stammdaten(dialog As ScriptDialog)
'Das Makro übernimmt die Werte aus der Registerseite Stammdaten
processdata.item("Name", with: dialog.value("Vorname"), in: "Kunde")
processdata.item("Ort", with: dialog.value("Ort"), in: "Kunde")
processdata.item("Strasse", with: dialog.value("Strasse"), in:
"Kunde")
... usw. ...
```

Makro Accept.Umsatz

```

Umsatz(dialog As ScriptDialog)

'Das Makro übernimmt die Werte aus der Registerseite Umsatz

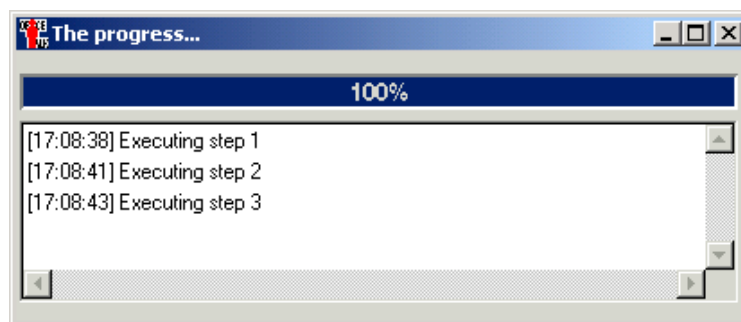
processdata.item("Januar", with: dialog.value("Januar"), in: "Umsatz")
processdata.item("Ort", with: dialog.value("Ort"), in: "Kunde")
processdata.item("Februar", with: dialog.value("Februar"), in:
"Umsatz")

... usw. ...

```

Beispiel eines nicht modalen Dialoges

Das Beispiel zeigt exemplarisch die Verwendung eines modalen Dialoges. In einem Hauptmakro werden mehrere Arbeiten ausgeführt und der Bearbeiter soll über den Fortschritt informiert werden. Das Beispiel besteht aus dem Makro *Test*. Es führt die Arbeiten aus, öffnet mit dem Makro *Start* den nicht modalen Dialog mit dem Fortschrittsbalken, und aktualisiert mit den Makros *BeginStep* (dialog As ScriptDialog, total As Integer, message As String) und *IncrementStep* (dialog As ScriptDialog, increment As Integer) den Dialog des Fortschrittsbalkens mit dem aktuellen Fortschritt der Arbeiten. Der Fortschrittsdialog besteht aus dem Fortschrittsbalken und einem Listfeld, in dem die einzelnen Arbeitsschritte protokolliert werden.

**Test**

```

'Tests the progressbar with a dummy progress

'Arguments:
'Returns:

Dim dialog As ScriptDialog
Dim i As Integer

dialog = Call ProgressBar.Start
Call ProgressBar.BeginStep ( dialog, 10, Executing step 1" )
For i = 1 To 10 Step 1
    Delay( 100 )
    Call ProgressBar.IncrementStep ( dialog, 1 )
Next
Call ProgressBar.BeginStep ( dialog, 10, Executing step 2" )
For i = 1 To 10 Step 1
    Delay( 100 )
    Call ProgressBar.IncrementStep ( dialog, 1 )
Next
Call ProgressBar.BeginStep ( dialog, 10, Executing step 3" )
For i = 1 To 10 Step 1

```

```

    Delay( 100
    Call ProgressBar.IncrementStep ( dialog, 1 )
Next
dialog.accept

```

Start

```

'Start the progressbar

'Arguments:
'Returns:      ScriptDialog      The dialog with progressbar

Dim dialog As ScriptDialog

dialog = New ScriptDialog
dialog.title("The progress...")
dialog.beModeless
dialog.addProgressBar( "progressbar", left: 2, top: 12, width: 400,
height: 20, total: 0, horizontal: True, reverse: False )
dialog.addList( Array(), named: "list", left: 2, top: 38, width: 400,
height: 100, horizontalScroll: True )
dialog.width( 407 )
dialog.height( 148 )
dialog.disableCancel
dialog.open
Return dialog

```

BeginStep (dialog As ScriptDialog, total As Integer, message As String)

```

'Start the progressbar new, beginning at 0 to total
'The steps of the overall process are printed with messages in the
list below
,
'Arguments:  dialog      The dialog with progressbar
'            total       The total amount of this step
'            message     The name of this step
,
'Returns:

Dim theSteps As Array
Dim theMessage As String

theMessage = "[" & Time & "]" & message
theSteps = dialog.values( "list" )
If theSteps = Null
    Then theSteps = Array( theMessage )
    Else theSteps( UBound( theSteps ) ) = theMessage
End If
dialog.value( "list", put: theSteps )
dialog.value( "progressbarTotal", put: total )
dialog.value( "progressbarCurrent", put: 0 )
dialog.invalidate(array("progressbar", "list"))

```

IncrementStep (dialog As ScriptDialog, increment As Integer)

```

'Increments the progressbar with increment
,
'Arguments:  dialog      The dialog containing the progressbar
'            increment   The increment value
,

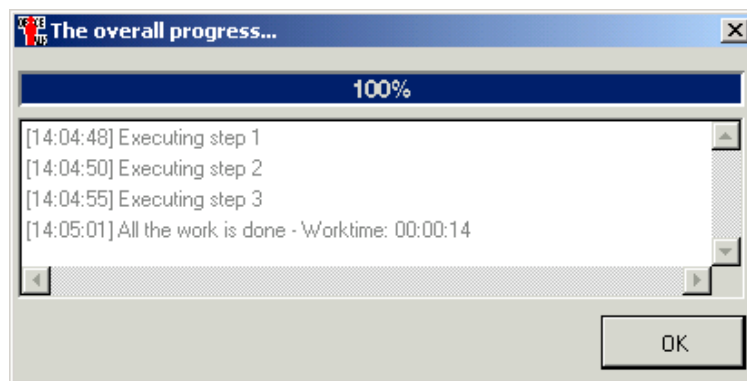
```

```
'Returns:
```

```
dialog.progressBar( "progressbar", increment: increment )
dialog.invalidate( "progressbar" )
```

Beispiel eines Dialoges mit Aktionen zum Start

Das Beispiel zeigt exemplarisch die Verwendung eines Dialoges mit Aktionen, die unmittelbar nach dem Öffnen ausgeführt werden. In der Postopen-Aktion des Startmakros *Test* wird mit dem Makro *WorkAll* (*dialog As Joops.Scripting.ScriptDialog*) die Arbeit ausgeführt. Die 3 Teile der auszuführenden Arbeit werden darin durch die dreimalige Verwendung des Makros *WorkStep* (*dialog As ScriptDialog, total As Integer, message As String*) ausgeführt. Der Benutzer wird dabei im Startmakro *Test* durch einen Fortschrittsbalken und eine Liste, in der die Teilaufgaben protokolliert werden, über den Fortschritt der Arbeiten informiert.



Test

```
'Tests the modal progressbar with a dummy work
'The work is divided into 3 single steps. The progress of each step
'is documented in a progressbar and a list of worked steps.
'All the work is done in the post load action
',
'Arguments:
',
'Returns:

Dim dialog As ScriptDialog
Dim theSteps As Array
Dim now As Time

dialog = New ScriptDialog
dialog.title( "The overall progress..." )
dialog.addProgressBar( "progressbar", left: 2, top: 12, width: 400,
height:
    20, total: 0, horizontal: True, reverse: False )
dialog.addList( Array(), named: "list", left: 2, top: 38, width: 400,
height: 100, horizontalScroll: True )
dialog.addAcceptButtonLeft( 323, top: 147, width: 80, height: 30,
default: True )
dialog.addPostOpenAction( "now = Time
    Call ProgressBarModal.WorkAll ( dialog )
    theSteps = dialog.values( "list" )
    theSteps( UBound( theSteps ) ) = "[" & Time & "]" " &
        "All the work is done - Worktime: " & Time
- now
```

```

        dialog.value( "list", put: theSteps )
        dialog.enableAccept" )
dialog.width( 407 )
dialog.height( 183 )
dialog.disableCancel
dialog.disableAccept
dialog.disable( "list" )
dialog.open

```

WorkAll (dialog As Joops.Scripting.ScriptDialog)

```

'Do the work and inform the user about progress
'
'Arguments:  dialog           The dialol with progressbar
'
'Returns:

Call ProgressBarModal.WorkStep ( dialog, 10, "Executing step 1" )
Call ProgressBarModal.WorkStep ( dialog, 20, "Executing step 2" )
Call ProgressBarModal.WorkStep ( dialog, 30, "Executing step 3" )

```

WorkStep (dialog As ScriptDialog, total As Integer, message As String)

```

'Do the required work of one step named with message
'
'Arguments:  dialog           The dialol with progressbar
'            total           The total amount of this step
'            message         The name of this step

'Returns:

Dim theSteps As Array
Dim theMessage As String
Dim i As Long

theMessage = "[" & Time & "]" & message
theSteps = dialog.values( "list" )
theSteps( UBound( theSteps ) ) = theMessage
dialog.value( "list", put: theSteps ) 'Actualize list with stepnames
dialog.value( "progressbarTotal", put: total ) 'Set the total amount
dialog.value( "progressbarCurrent", put: 0 ) 'Start at 0
For i = 1 To total Step 1
    Delay( 200 ) 'Increment processbar
    dialog.progressBar( "progressbar", increment: 1 )
Next

```

step

Die globale Variable bezeichnet den aktuell bearbeiteten Arbeitsschritt. Jeder Arbeitsschritt kennt eine Reihe von besonderen Diensten.

Syntax

actions

Der Dienst liefert alle Aktionen des Arbeitsschrittes in einen Array.

actionSymbol (<Zeichenkette>)

Mit dem Dienst wird der Ablaufhistorie der laufenden Aktion das Bildsymbol mit dem Namen <Zeichenkette> zugewiesen. In der Vorgangshistorie wird die Aktion mit diesem Bildsymbol angezeigt. Die Namen der Bildsymbole finden Sie im Dienst `symbolNames` der Systemvariable `action`.

comment(<Zeichenkette for: <Vorgangselement>)

Der Dienst fügt im Protokoll des Vorgangselementes <Vorgangselement> die Bemerkung mit der angegebenen Zeichenkette an. Für <Vorgangselement> schreiben Sie:

process	Der laufende Vorgang in dessen Kontext das Makro abläuft.
step	Der laufende Arbeitsschritt in dessen Kontext das Makro abläuft.
action	Die laufende Aktion in dessen Kontext das Makro abläuft.

comment(<Zeichenkette>)

Der Dienst ist ein Synonym für den Dienst `comment(<Zeichenkette for: <Vorgangselement>)` und fügt im Protokoll für den laufenden Arbeitsschritt die Bemerkung mit der angegebenen Zeichenkette an.

description

Der Dienst liefert die Beschreibung des Arbeitsschrittes als Zeichenkette.

description(<Zeichenkette>)

Mit dem Dienst belegen Sie die Beschreibung dieses ausgeführten Arbeitsschrittes mit <Zeichenkette>.

execution

Der Dienst liefert die Anzahl der Ausführungen des Arbeitsschrittes. Da die Ablauffolge der Arbeitsschritte eines Vorgangs auch rekursiv sein kann, kann der selbe Arbeitsschritt auch mehrfach ausgeführt werden.

history

Der Dienst liefert die Historie des Arbeitsschrittes in druckbarer Form.

history(<Zeichenkette>)

Mit dem Dienst belegen Sie die Historie dieses ausgeführten Arbeitsschrittes mit <Zeichenkette>.

isFirst

Der Dienst meldet `True`, wenn der Arbeitsschritt der erste im Ablaufpfad des Vorgangs ist, ansonsten `False`.

isEndOfPath

Der Dienst ist ein Alias für den Dienst *isLast*. Der Dienst antwortet auch mit `True`, wenn der Arbeitsschritt nur sich selbst als Nachfolger hat.

isLast

Der Dienst meldet `True`, wenn der Arbeitsschritt der letzte im Ablaufpfad des Vorgangs ist, ansonsten `False`.

isMandatory

Der Dienst meldet `True`, wenn die Ausführung des Arbeitsschrittes zwingend ist, ansonsten `False`.

isOptional

Der Dienst meldet `True`, wenn die Ausführung des Arbeitsschrittes optional ist, ansonsten `False`.

isStartOfPath

Der Dienst ist ein Alias für den Dienst *isFirst*.

maxCharComment

Der Dienst liefert die maximale Anzahl der Zeichen, die ein Kommentar enthalten darf. Ein Kommentar mit mehr Zeichen wird beim Speichern (`step.comment(...)`) automatisch und ohne Warnung abgeschnitten. Sie können im `ScriptDialog` die Eingabelänge mit dem Dienst *limit* (`<NameOderArrayMitNamen>`, `maxChar: <Variable>`) begrenzen.

nextSteps

Der Dienst ist ein Alias für den Dienst *succeedingSteps* und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion gelöscht wird.

preceedingResultNames

Der Dienst liefert die Namen der eingehenden Arbeitsschrittergebnisse in einem `Array`. Das sind die Namen der Arbeitsschrittergebnisse die von allen vorangehenden Arbeitsschritten zu diesem Arbeitsschritt zeigen.

processTime

Der Dienst liefert die kalkulierte Ausführungszeit und Dauer des Arbeitsschrittes.

preceedingSteps

Der Dienst liefert den oder die direkte Arbeitsschrittvorgänger des Arbeitsschrittes in einem `Array`.

resultNames

Der Dienst liefert die Namen der ausgehenden Arbeitsschrittergebnisse in einem Array. Das sind die Namen der Arbeitsschrittergebnisse des Arbeitsschrittes. Sie verweisen von diesem Arbeitsschritt zu den direkt nachfolgenden Arbeitsschritten.

startAfter(<Zahl>, for: <Ergebnisname>)

Der Dienst setzt im Arbeitsschrittergebnis mit dem Namen <Ergebnisname> die mit <Zahl> angegebene Wartezeit in Stunden, bis der Arbeitsschritt gestartet wird. <Zahl> muss dazu vom Type Integer und <Ergebnisname> von Typ String sein. Der Dienst liefert Null, wenn <Ergebnisname> keinen Eingangspfad (Arbeitsschrittergebnis) des Arbeitsschrittes bezeichnet.

startDate(<Startdatum>, time: <Uhrzeit>, for: <Ergebnisname>)

Der Dienst setzt im Arbeitsschrittergebnis mit dem Namen <Ergebnisname> das angegebene Startdatum. Dadurch wird das Startdatum des Arbeitsschrittes für den mit <Ergebnisname> benannten Eingangspfad bestimmt. <Datum> muss dazu vom Type Date, <Zeit> von Typ Time und <Ergebnisname> von Typ String sein. Der Dienst liefert Null, wenn <Ergebnisname> keinen Eingangspfad (Arbeitsschrittergebnis) des Arbeitsschrittes bezeichnet. Anwendung siehe Kapitel *Beispiel*.

startActions

Der Dienst liefert alle Startaktionen des Arbeitsschrittes in einen Array.

succeedingResultNames

Der Dienst ist ein Alias für den Dienst *resultNames*.

succeedingSteps

Der Dienst liefert den oder die direkte Arbeitsschrittnachfolger des Arbeitsschrittes in einem Array.

symbolNameString

Der Dienst liefert den Namen des Arbeitsschrittsymbols.

termActions

Der Dienst liefert alle Endeaktionen des Arbeitsschrittes in einen Array.

userActions

Der Dienst liefert alle Benutzeraktionen des Arbeitsschrittes in einen Array.

Beispiel

```
Dim theSteps As Array
Dim theStep As Step
Dim i As Integer

step.comment("Das ist der Protokolltext für die Aktion", for: action)
step.comment("Das ist der Protokolltext für den Vorgang", for:
process)
```



```
step.comment("Das ist der Protokolltext für den Arbeitsschritt")

'Startzeit des Arbeitsschittergebnisses bearbeiten für den
'Arbeitsschritt Akquisition - Abschließen auf heute + 5 Tage setzen
theSteps = step.nextSteps
For i = 1 To UBound(theSteps) Step 1
    theStep = theSteps(i-1)
    If theStep.displayName = "Akquisition - Abschließen"
        Then theStep.startDate(Date + 5, time: Time, for: "bearbeiten")
    End If
Next
```

StepHistory

Eine Arbeitsschritthistorie beinhaltet Informationen zu einem ausgeführten Arbeitsschritt. Der Dienst *steps* der Systemvariablen *ProcessHistory* liefert die Historien der ausgeführten Arbeitsschritte eines Vorgangs.

Syntax

actions

Der Dienst liefert die Historien der ausgeführten Aktionen des Arbeitsschrittes.

comment

Der Dienst liefert den im Skriptmakro erzeugten Kommentar zu dem Arbeitsschritt. Siehe Dienst *comment*(*<Zeichenkette>*) der Systemvariablen *step*.

consumed

Der Dienst liefert die Kosten der verbrauchten Ressourcen des Arbeitsschrittes.

delegated

Der Dienst liefert den Namen des Bearbeiters, an den der Arbeitsschritt delegiert wurde.

description

Der Dienst liefert die Beschreibung des Arbeitsschrittes.

endedAt

Der Dienst liefert Endedatum und Uhrzeit des Arbeitsschrittes.

initiatingWorker

Der Dienst liefert den Namen des Bearbeiters, der die Vorgangdelegation veranlasst hat. Der Name eines passiven Bearbeiters beginnt mit einem *.

initiatingWorkerName

Der Dienst liefert den Namen des Bearbeiters, der die Vorgangdelegation veranlasst hat. Der führende * bei Namen passiver Bearbeiter ist im Ergebnis nicht enthalten.

startedAt

Der Dienst liefert Startdatum und Uhrzeit des Arbeitsschrittes.

startingWorker

Der Dienst liefert den Namen des Bearbeiters, der den Arbeitsschritt ausgeführt hat. Der Name eines passiven Bearbeiters beginnt mit einem *.

startingWorkerName

Der Dienst liefert den Namen des Bearbeiters, der den Arbeitsschritt ausgeführt hat. Der führende * bei Namen passiver Bearbeiter ist im Ergebnis nicht enthalten.

targetStartAt

Der Dienst liefert das vorgesehene Startdatum des Arbeitsschrittes. Es kann vom tatsächlichen Startdatum abweichen.

stepscheduler

Die globale Variable bezeichnet den Scheduler. Er ist für die Überwachung und Ausführung eines Arbeitsschrittes verantwortlich. Der Scheduler kennt eine Reihe von besonderen Diensten. Die Namen der Dienste beginnen mit `stepscheduler..`

Syntax

adoptNextStep

Mit diesem Dienst wird der nächste auszuführende Arbeitsschritt vom selben Bearbeiter übernommen, wenn der Bearbeiter die entsprechenden Rechte besitzt (siehe *Beispiel-2*). In jedem Fall muss der Arbeitsschritt mit einem bekannten Arbeitsschrittergebnis beendet werden (siehe Anweisung *Return* und Business-Process-Management, Kapitel *Arbeitsschrittergebnis*).

anyEnvironment(<Name>)

Der Dienst liefert den Wert der Einstellung <Name>. Vorrangig wird der Wert der bearbeiterbezogenen Einstellung des angemeldeten Bearbeiters geliefert. Ist diese nicht vorhanden, wird der Wert der Einstellung aus den gemeinsamen Einstellungen geliefert.

asHTMLString(<Variable>)

Der Dienst wandelt den Inhalt der Variable <Variable> in eine **UTF8**-kodierte Zeichenkette, bei der jedes Zeichen **HTML**-konform dargestellt wird. D.h.: Jedem Zeichen wird das Zeichen % vorangestellt. <Variable> muss dazu einen Wert enthalten, der als Zeichenkette dargestellt werden kann. U.a. trifft das auf Variable vom Typ `String` zu. Die Ursprungszeichenkette wird dabei nicht verändert. Die gewandelte Zeichenkette wird als Ergebnis geliefert.

Hinweis: Zeichenketten, die mit Diensten der Systemvariablen `WSDClient` und `HTTPClient` an Webapplikationen übergeben werden, müssen mit diesem Dienst **HTML**-konform codiert werden.

asPlatformString(<Variable>)

Der Dienst wandelt den Inhalt der Variable <Variable>, die eine **UTF8**-kodierte **HTML**-konforme Zeichenkette enthalten muss, in eine plattformkodierte Zeichenkette. Die Ursprungszeichenkette wird dabei nicht verändert. Die gewandelte Zeichenkette wird als Ergebnis geliefert.

applicationFor(<Datei>)

Der Dienst liefert die Anwendung, mit der die angegebene Datei geöffnet werden kann. Dieses Ergebnis kann in der Funktion *Shell* als Argument `Programm` verwendet werden. Das Ergebnis des Dienstes ist vorrangig die Anwendungszuordnung aus Business-Process-Management, Kapitel *Einstellungen ohne Benutzerinterface*, Abschnitt *Applications*. Wenn hier keine Anwendungszuordnung getroffen wurde, wird die Zuordnung der Plattform geliefert. Auf Windows-Plattformen ist die Anwendungszuordnung i.d.R. durch die System-einstellung in Windows festgelegt. Auf Linux-Plattformen muss die Anwendungszuordnung in Business-Process-Management, Kapitel *Einstellungen ohne Benutzerinterface*, Abschnitt *Applications*, oder in der Funktion *Shell* festgelegt werden.

automation(<Vorgang>)

Der Dienst startet oder beendet die automatische Bearbeitung für den angegebenen Vorgang je nach gegenwärtigem Zustand. <Vorgang> muss dabei ein bereits gestarteter Vor-

gang sein und der angemeldete Bearbeiter muss die Berechtigung **autom. ausführen** für diesen Vorgang besitzen. Der Dienst liefert bei ordnungsgemäßer Ausführung als Ergebnis `True`, ansonsten `False`. Weitere Einzelheiten dazu finden Sie in der Dokumentation Business-Process-Management, Kapitel *Automatische Bearbeitung*. Sehen Sie dazu auch das *Beispiel-4*

Hinweis: Wenn es sich bei `<Vorgang>` um den aktuellen Vorgang handelt, und die automatische Bearbeitung beendet wird, wird der Automatikstatus (die visuelle Anzeige) erst beim nächsten automatischen Ausführungsintervall (siehe Kapitel Register Aufgaben in der Dokumentation Business-Process-Management) entfernt.

can(<Bearbeiter>, work: <Vorgang>)

Der Dienst antwortet mit `True` falls der Bearbeiter `<Bearbeiter>` das Rechte besitzt um den Vorgang `<Vorgang>` in irgend einer Art und Weise auszuführen. Das umfasst die Rechte **starten, warten ignorieren, beenden, übernehmen, ausführen, automatisch ausführen, delegieren**, die **Ausführungspriorität ändern** und den **Startzeitpunkt des nächsten Arbeitsschrittes ändern**. Weitere Einzelheiten zu den Bearbeitungsrechten finden Sie in der Dokumentation Business-Process-Management, Kapitel *Rechte*.

canWorker(<Bearbeiter>, handle: <Vorgang>)

Der Dienst ist ein Alias für den Dienst `can(<Bearbeiter>, work: <Vorgang>)` und sollte nicht mehr verwendet werden, da er ab der nächsten Hauptversion nicht mehr unterstützt wird.

consumed(<Name>, amount: <Zahl>)

Der Dienst berechnet die Menge der verbrauchten Ressourcen für eine Aktion und speichert das Ergebnis in der Vorgangshistorie. Dabei bezeichnet `<Name>` den Namen der Ressource und `<Zahl>` die Anzahl der verbrauchten Einheiten. Als Dezimaltrenner, falls erforderlich, müssen Sie beim Argument `<Zahl>` den Punkt `.` verwenden. Der Dienst antwortet mit `True`, wenn die Berechnung erfolgreich war. Wenn die Ressource mit dem angegebenen Namen `<Name>` nicht gefunden wurde, weil der Name falsch geschrieben ist, oder weil die Ressource in einem nicht sichtbaren Bearbeiter liegt (siehe Dokumentation Business-Process-Management, Kapitel *Sichtbereich*), wird eine Ausnahme geworfen (siehe Kapitel *Behandlung fehlerhafter Argumente*). Konnte die Berechnung aus anderen Gründen nicht durchgeführt werden, antwortet der Dienst mit `False`. Ein Anwendungsbeispiel dazu sehen Sie im *Beispiel-2*.

consumed(<Name>, amount: <Menge>, cost: <Betrag>, unit: <Einheit>)

Beschreibung siehe Dienst `consumed(<Name>, amount: <Menge>, cost: <Betrag>, unit: <Einheit>, description: <Beschreibung>)`.

consumed(<Name>, amount: <Menge>, cost: <Betrag>, unit: <Einheit>, description: <Beschreibung>)

Mit dem Dienst wird für den aktuell ausgeführten Arbeitsschritt eine Ressource in der Historie abgelegt. `<Name>` benennt die Ressource. `<Menge>` beziffert die Menge der verbrauchten Ressource. `<Betrag>` beziffert die Gesamtkosten der verbrauchten Ressource. `<Einheit>` benennt die Einheit der Ressource. "Tage", "Stunden", "Minuten" und "Sekunden" sind vordefinierte Einheiten. Sie können auch jede andere Einheit verwenden. `<Beschreibung>` beschreibt optional die Ressource genauer. Negative Werte für `<Menge>` und `<Betrag>` bestimmen eine Ressource nicht als Kosten, sondern als Einnahme.

Ein Beispiel:

```
stepscheduler.consumed("Planungskosten", amount: 1,3, cost: 870,00,
unit: „Tage“)
stepscheduler.consumed("Beratung", amount: 2,5, cost: -870,00, unit:
„Tage“)
```

currentEnvironment(<Name>)

Der Dienst liefert den Wert der Einstellung <Name> des angemeldeten Bearbeiters. <Name> ist dabei der vollständige Name der Einstellung. Namensteile werden durch / oder \ getrennt. <Name> bezeichnet die Einstellung aus den bearbeiterbezogenen Einstellungen des angemeldeten Bearbeiters. Ist die Einstellung dort nicht vorhanden, wird sie auf Windows-Plattformen aus dem Registry HKEY_CURRENT_USER\Software\JOOPS GmbH\OfficeTalk und auf Linux-Plattformen aus der Datei \$HOME/OfficeTalk.ini geliefert.

currentEnvironment(<Name>, put: <Zeichenkette>)

Der Dienst besetzt den Wert der Einstellung <Name> des angemeldeten Bearbeiters mit der Zeichenkette <Zeichenkette>. <Name> ist dabei der vollständige Name der Einstellung. Namensteile werden durch / oder \ getrennt. Wird für <Zeichenkette> Null angegeben, wird die Einstellung gelöscht.

databaseName

Der Dienst liefert den Namen des OfficeTalk eigenen Datenbank. Ist die Datenbank nicht bekannt, was unter normaler Verwendung nicht möglich ist, liefert der Dienst eine leere Zeichenkette.

defineStartDate

Der Dienst ist das Synonym für den Dienst *defineStartDate* (<Variable>).

defineStartDate (<Variable>)

Mit dem Dienst wird das, durch *startDate*(<Datum>, time: <Zeit>), definierte Ausführungsdatum des nächsten Arbeitsschrittes fest fixiert. Auch das Ausführungsdatum einer folgenden Aktion des Arbeitsschrittes mit einem unterschiedlichen Arbeitsschrittergebnis wird ignoriert (siehe *Beispiel-2*). Falls durch das definierte Startdatum der nächste Arbeitsschritt sofort auszuführen ist, wird dieser nach dem Ende des laufenden Arbeitsschrittes sofort gestartet. Soll der nächste Arbeitsschritt nicht sofort gestartet werden, muss <Variable> auf True gesetzt werden.

disableRecursion

Der Dienst verbietet für den weiteren Ablauf des Vorgangs eine Rekursion der Bearbeitungsreihenfolge der Arbeitsschritte.

disableNextStep

Wenn ein Arbeitsschritt beendet wurde, startet der Scheduler den nachfolgenden Arbeitsschritt, wenn dieser zur Ausführung ansteht. Der Dienst verhindert diese Ausführung. Er ist erforderlich, wenn das Ausführungsdatum des nächsten Arbeitsschrittes mit dem Dienst

enableRecursion

Mit dem Dienst erlauben Sie eine Rekursion der Bearbeitungsreihenfolge der Arbeitsschritte. Wenn der selbe Arbeitsschritt innerhalb eines Ablaufes mehrfach ohne Pause (Arbeitsschrittergebnis **sofort**) ausgeführt wird, nennt man das Rekursion. Generell deutet dies auf einen Ablauffehler hin und ist ohne Aufruf dieses Dienstes verboten.

environment(<Name>, in: <Kategorie>)

Der Dienst liefert den Wert der Einstellung <Name>. Der Dienst ist nur auf Windows Plattformen verfügbar und liefert nur Zeichenketteneinträge aus der zentralen Registry. <Name> ist dabei der vollständige Registryname der Einstellung. Namensteile werden

durch / oder \ getrennt. <Kategorie> ist der Name der Wurzel. Dafür schreiben Sie einen der zulässigen Namen HKEY_CLASSES_ROOT, HKEY_CURRENT_CONFIG, HKEY_CURRENT_USER oder HKEY_LOCAL_MACHINE.

environment(<Name>, in: <Kategorie>, put: <Zeichenkette>)

Der Dienst besetzt den Wert einer Einstellung mit der Zeichenkette <Zeichenkette>. Der Dienst ist nur auf Windows Plattformen verfügbar und kann nur Zeichenketteneinträge in der zentralen Registry setzen. <Name> ist dabei der vollständige Registryname der Einstellung. Namensteile werden durch / oder \ getrennt. <Kategorie> ist der Name der Wurzel. Dafür schreiben Sie einen der zulässigen Namen HKEY_CLASSES_ROOT, HKEY_CURRENT_CONFIG, HKEY_CURRENT_USER oder HKEY_LOCAL_MACHINE.

environment(<Schlüssel>, category: <Kategorie>, in: <Datei>)

Der Dienst liefert den Wert eines Schlüssels aus der mit <Datei> benannten Ini-Datei. <Kategorie> benennt dabei den Namen der Kategorie und <Schlüssel> den Namen des Schlüssels innerhalb der Kategorie. Als Ergebnis liefert der Dienst den Schlüsselwert als Zeichenkette (String) oder Null, wenn die Kategorie den Schlüssel nicht enthält. Der semantische Aufbau der Datei entspricht dem Aufbau einer Ini-Datei auf Windows-Plattformen:

```
[Kategorie]
Schlüssel1=Wert1
Schlüssel2=Wert2
```

environment(<Schlüssel>, category: <Kategorie>, put: <Wert>, in: <Datei>)

Der Dienst belegt den Wert eines Schlüssels in der mit <Datei> benannten Ini-Datei. <Kategorie> benennt dabei den Namen der Kategorie und <Schlüssel> den Namen des Schlüssels innerhalb der Kategorie. Ist <Wert> Null oder eine leere Zeichenkette, wird der Schlüssel vollständig aus der Ini-Datei entfernt. Der semantische Aufbau der Datei entspricht dem Aufbau einer Ini-Datei auf Windows-Plattformen.

Hinweis: <Wert> wird in der Ini-Datei immer als Zeichenkette gespeichert. Bei Bedarf wird <Wert> vorher in eine Zeichenkette (String) gewandelt.

format(<Variable>, as: <Formattyp>, with: <Formatanweisung>)

Der Dienst liefert den Inhalt von <Variable> als Zeichenkette formatiert. Dabei wird der Inhalt der Variable als Datentyp <Formattyp> angenommen und nach den Regeln in <Formatanweisung> formatiert. Einzelheiten zu den Argumenten <Formattyp> und <Formatanweisung> finden Sie im Dienst *format (<NameOderArrayMitNamen>, with: <Formatregel>)*. Ein kurzes Anwendungsbeispiel sehen Sie im *Beispiel-6*.

format(<Variable>, as: <Formattyp>)

Der Dienst liefert den Inhalt von <Variable> als Zeichenkette formatiert. Dabei wird der Inhalt der Variable nach den Standardregeln für den Datentyp <Formattyp> formatiert. Einzelheiten zu dem Argument <Formattyp> finden Sie im Dienst *format (<NameOderArrayMitNamen>, with: <Formatregel>)*.

format(<Variable>)

Der Dienst liefert den Inhalt von <Variable> als Zeichenkette formatiert. Dabei wird der Inhalt der Variable entsprechend den Standardregeln seines Datentyps formatiert. Einzelheiten zu den Formatierungsregeln finden Sie im Dienst *format (<NameOderArrayMitNamen>, with: <Formatregel>)*.

go(<Vorgang>, to: <Bearbeiter>)

Mit dem Dienst delegieren Sie die Bearbeitung des nächsten Arbeitsschrittes im Vorgang <Vorgang> an den Bearbeiter <Bearbeiter>. Damit wird damit ein aktiver Vorgang zur weiteren Ausführung an einen anderen Bearbeiter, als im nächsten Arbeitsschritt bestimmt, delegiert. <Vorgang> muss ein gestarteter (aktiver) Vorgang, und darf nicht der in Ausführung befindliche Vorgang sein! <Bearbeiter> kann ein passiver (Abteilung, Büro, usw.), oder ein aktiver Bearbeiter (Schreibtisch, Maschine) sein. <Bearbeiter> muss jedoch die erforderlichen Vorgangsrechte besitzen. Einzelheiten zu den Vorgangsrechten lesen Sie in der Dokumentation Business-Process-Management, Kapitel *Rechte*. Der Dienst beendet mit `True`, wenn die Delegation ausgeführt wurde, ansonsten mit `False`. Sehen Sie dazu auch das *Beispiel-5*.

go(<Vorgang>, to: <Bearbeiter>, comment: <Kommentar>)

In der Historie und in der Delegationsbenachrichtigung per E-Mail wird der angegebene Kommentartext eingetragen. Die weiteren Einzelheiten zum Dienst finden Sie im Kapitel *go(<Vorgang>, to:<Bearbeiter>)*.

goTo(<Bearbeiter>)

Mit dem Dienst delegieren Sie die Bearbeitung des nächsten Arbeitsschrittes an den Bearbeiter <Bearbeiter>. Mögliche Bearbeiter erhalten Sie u.a. mit dem Dienst *workers* oder *substitutes* kann ein passiver (Abteilung, Büro, usw.), oder ein aktiver Bearbeiter (Schreibtisch, Maschine) sein. <Bearbeiter> muss jedoch die erforderlichen Vorgangsrechte besitzen. Einzelheiten zu den Vorgangsrechten lesen Sie in der Dokumentation Business-Process-Management, Kapitel *Rechte*. Der Dienst beendet mit `True`, wenn die Delegation ausgeführt wurde, ansonsten mit `False`. Sehen Sie dazu auch das *Beispiel-3*. In jedem Fall muss der Arbeitsschritt mit einem bekannten Arbeitsschrittergebnis beendet werden (siehe Anweisung *Return* und Business-Process-Management, Kapitel *Arbeitsschrittergebnis*).

goTo(<Bearbeiter>, comment: <Kommentar>)

In der Historie und in der Delegationsbenachrichtigung per E-Mail wird der angegebene Kommentartext eingetragen. Die weiteren Einzelheiten zum Dienst finden Sie im Kapitel *goTo(<Bearbeiter>)*.

goToPredecessor

Mit dem Dienst delegieren Sie die Bearbeitung des nächsten Arbeitsschrittes an den Bearbeiter des vorangegangenen Arbeitsschrittes. Sie geben also die Ausführung des Vorgangs wieder an den Absender zurück. Dazu müssen Sie nicht wissen, wer der Bearbeiter des letzten Arbeitsschrittes war. Der letzte Bearbeiter ist immer die Maschine oder der Schreibtisch, von welchem der Arbeitsschritt konkret ausgeführt wurde. Der Dienst beendet mit `True`, wenn die Delegation ausgeführt wurde, ansonsten mit `False`. In jedem Fall muss der Arbeitsschritt mit einem bekannten Arbeitsschrittergebnis beendet werden (siehe Anweisung *Return* und Business-Process-Management, Kapitel *Arbeitsschrittergebnis*).

isAutomation(<Vorgang>)

Der Dienst antwortet mit `True`, falls die automatische Bearbeitung für den angegebenen Vorgang <Vorgang> gestartet ist. Weitere Einzelheiten dazu finden Sie in der Dokumentation Business-Process-Management, Kapitel *Automatische Bearbeitung*.

nextStepCategory(<CategoryVariable>, name: <NameVariable>)

Mit dem Dienst wird der Name des nächsten auszuführenden Arbeitsschrittes bestimmt. <CategoryVariable> und <NameVariable> sind die Zeichenketten für die Kategorie und die Bezeichnung des Arbeitsschrittes. Der Dienst liefert bei ordnungsgemäßer Ausführung

das Ergebnis `True`, ansonsten `False`. Durch den Dienst wird der Arbeitsschritt zukünftig in allen weiteren Verwendungen mit diesem Namen benannt. Das betrifft alle Dialoge und seine Historie. In Sonderfällen kann damit ein Arbeitsschritt durch einen konkreten Namen besser seine vorgesehene Aufgabe darstellen.

log(<Variable>)

Der Dienst schreibt den Inhalt von <Variable> in die Systemprotokolldatei, wenn die Option **Protokollierung** in den allgemeinen Einstellungen (siehe Kapitel *Einstellungen - Register Allgemeines* in der Dokumentation *Business-Process-Management*) aktiviert ist. Damit können Informationen zur Diagnose des Ablaufes in die Systemprotokolldatei geschrieben werden.

log(<Variable> into: <Dateiname>)

Der Dienst schreibt den Inhalt der Variable in die Datei <Dateiname>, wenn die Option **Protokollierung** in den allgemeinen Einstellungen (siehe Kapitel *Einstellungen - Register Allgemeines* in der Dokumentation *Business-Process-Management*) aktiviert ist. Damit können Informationen zur Diagnose des Ablaufes in eine Datei geschrieben werden.

nextStepWorkerFor(<Name>)

Der Dienst liefert den Bearbeiter, auf den das Arbeitsschrittergebnis mit dem Namen <Name>, verweist (siehe Dienst *can(<Bearbeiter>, work: <Vorgang>)*).

MD5String(<Variable>)

Mit dem Dienst wird für die Zeichenkette in der Variablen <Variable> eine digitale Signatur nach den MD5-Regeln (siehe RFC 1321) erstellt und als Zeichenkette geliefert. Der Inhalt von <Variable> muss `String` kompatibel sein.

MD5File(<Dateiname>)

Mit dem Dienst wird für die Datei <Dateiname> eine digitale Signatur nach den MD5-Regeln (siehe RFC 1321) erstellt und als Zeichenkette geliefert.

notify(<Eintrag>, with: <Datum>, in: <Gruppe>)

Mit dem Dienst übergeben Sie an alle Vorgänge, die auf die Beendigung des aktuellen Vorgangs warten, Vorgangsdaten. <Datum> ist dabei das Datum, <Eintrag> der Eintragsname des Datums und <Gruppe> der Gruppenname (siehe auch Dienst *item(<Eintrag>, with: <Datum>, in: <Gruppe>)* der Systemvariable *processdata*). Mit diesem Dienst können Sie die Übergabe von Ergebnissen der Vorgangsausführung an den Initiator, einen anderen Vorgang, realisieren.

notify(<Eintrag>, with: <Datum>, inEntry: <Gruppe>)

Mit dem Dienst ist ein Alias für den Dienst *notify(<Eintrag>, with: <Datum>, in: <Gruppe>)* und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

undefineStartDate

Mit dem Dienst wird eine Fixierung des Ausführungsdatum durch den Dienst *defineStartDate* wieder rückgängig gemacht.

openHistory

Der Dienst öffnet die Historie des aktuellen Vorgangs im Dialog **Historie für: <Vorgangsname>** und hält die Ausführung des Skriptmakros bis zum Schließen des Dialoges an. Der aktuelle Arbeitsschritt wird dabei nicht angezeigt.

openHistory(<Vorgang>)

Der Dienst öffnet die Historie des Vorgangs <Vorgang> im Dialog **Historie für: <Vorgangsname>** und hält die Ausführung des Skriptmakros bis zum Schließen des Dialoges an. Es kann sich hier um den aktuellen, oder einen anderen aktiven Vorgang des aktuellen Bearbeiters handeln. Der Bearbeiter muss das Vorgangsrecht **Sehen** für die gleichnamige Vorgangsvorlage besitzen. Ein kleines Beispiel:

```
Dim processes As Array  
processes = worker.activeProcesses  
stepscheduler.openHistory( processes(0) )
```

Die erste Anweisung liefert in processes alle laufenden Vorgänge des aktuellen Bearbeiters. Die zweite Anweisung öffnet die Historie des ersten Vorgangs aus dem Array.

openHistory(<Kategorienname>, name: <Bezeichnung>, logicalName: <zusätzlicher Name>)

Der Dienst öffnet die Historie der abgeschlossenen Vorgänge mit dem Namen <Kategorienname> und <Bezeichnung> im Dialog **Historie für: <Vorgangsname>** und hält die Ausführung des Skriptmakros bis zum Schließen des Dialoges an. Es werden nur die abgeschlossenen Vorgänge mit dem zusätzlichen Namen <zusätzlicher Name> angezeigt. Der Name kann die Wildcards * und ? enthalten. Die gleichnamige Vorgangsvorlage muss sich in der sichtbaren Bearbeiterhierarchie befinden und der aktuelle Bearbeiter muss das Vorgangsrecht **Sehen** für die Vorgangsvorlage besitzen, ansonsten öffnet sich der Historiendialog ohne Inhalt.

openProcessData

Der Dienst öffnet die Vorgangsdaten des aktuellen Vorgangs im Dialog **Die Daten des Vorgangs: <Vorgangsname>** und hält die Ausführung des Skriptmakros bis zum Schließen des Dialoges an.

openProcessData(<Vorgang>)

Der Dienst öffnet die Vorgangsdaten des Vorgangs <Vorgang> im Dialog **Die Daten des Vorgangs: <Vorgangsname>** und hält die Ausführung des Skriptmakros bis zum Schließen des Dialoges an. Es kann sich hier um den aktuellen, oder einen anderen aktiven Vorgang des aktuellen Bearbeiters handeln. Der Bearbeiter muss das Vorgangsrecht **Sehen** für die gleichnamige Vorgangsvorlage besitzen. Ein kleines Beispiel:

```
Dim processes As Array  
processes = worker.activeProcesses  
stepscheduler.openProcessData( processes(0) )
```

Die erste Anweisung liefert in processes alle laufenden Vorgänge des aktuellen Bearbeiters. Die zweite Anweisung öffnet die Vorgangsdaten des ersten Vorgangs aus dem Array.

processTemplates

Der Dienst liefert alle sichtbaren Vorgangsvorlagen des angemeldeten Bearbeiters als Array. Einzelheiten zum Sichtbereich finden Sie im Kapitel *Sichtbereich* der Dokumentation Business-Process-Management.

quote(<Zeichen>, in: <Zeichenkette>, with: <Quotezeichen>)

Der Dienst quotet in der Zeichenkette <Zeichenkette> das Zeichen <Zeichen> mit dem Zeichen <Quotezeichen>. Dieser Dienst ist für Sonderfälle bei der Kommunikation mit externen Systemen sinnvoll, wenn geklammerte Zeichenketten übergeben werden müssen.

Ein Beispiel:

```
stepscheduler.quote( ChrB(39), in:"Planung's", with: ChrB(39))
```

ergibt die Zeichenkette "Planung's"

resource(<Name>)

Der Dienst liefert die Ressource mit dem Namen <Name>. Die Ressource muss sichtbar sein.

resources

Der Dienst liefert alle Ressourcen, die für den angemeldeten Bearbeiter sichtbar, und damit verwendbar sind als Array.

resourceNames

Der Dienst liefert die Namen aller Ressourcen, die für den angemeldeten Bearbeiter sichtbar, und damit verwendbar sind als Array. Diese Namen können in dem Dienst *consumed*(<Name>, amount: <Zahl>) verwendet werden.

resultNames

Der Dienst liefert die Namen aller möglichen Arbeitsschrittergebnisse als Array. Damit können z.B. in einem Skriptmakro dem Benutzer die möglichen Arbeitsschrittergebnisse zur Auswahl angeboten werden. Der ausgewählte Name kann als Return-Wert des Makros verwendet werden und bestimmt somit den nächsten auszuführenden Arbeitsschritt (siehe Dienst *can*(<Bearbeiter>, work: <Vorgang>)). Ein Beispiel für diese Vorgehensweise sehen Sie im Makro **RunUserAction.Protocol**.

scheduleProcess(<Process>)

Mit dem Dienst wird nach dem Ende des aktuellen Arbeitsschrittes die Ausführung des Vorgangs <Process> initiiert. Wenn sich der Vorgang in der persönlichen Aufgabenliste des angemeldeten Bearbeiters befindet, und das Startdatum des nächsten Arbeitsschrittes erreicht ist, entspricht der Dienst dem Kontextmenü **Ausführen** für den Vorgang in der persönlichen Aufgabenliste. Wenn sich der Vorgang nicht in der persönlichen Aufgabenliste des angemeldeten Bearbeiters befindet, oder das Startdatum des nächsten Arbeitsschrittes nicht erreicht ist, hat der Dienst keine Auswirkung. Das Ergebnis des Dienstes ist *True*, wenn <Process> ausgeführt wird, ansonsten *False*.

shutdownOn(<Modus>)

Mit dem Dienst wird nach dem Ende des aktuellen Arbeitsschrittes oder des aktuellen Vorgangs OfficeTalk beendet. Für Modus schreiben Sie die Zeichenkette

EndOfStep	OfficeTalk wird nach dem Ende des laufenden Arbeitsschrittes beendet.
EndOfProcess	OfficeTalk wird nach dem Ende des laufenden Vorgangs beendet. Der Dienst wirkt nur, wenn er während dieses Ausführungszykuses verwendet wurde.

startAt.date

Der Dienst liefert das Startdatum für den nächsten Arbeitsschritt (siehe *Beispiel-2*).

startAt.time

Der Dienst liefert die Startzeit für den nächsten Arbeitsschritt (siehe *Beispiel-2*).

startDate(<Datum>, time: <Zeit>)

Mit dem Dienst wird das Ausführungsdatum und die Uhrzeit des nächsten Arbeitsschrittes bestimmt. Das Ausführungsdatum des Arbeitsschrittergebnisses der aktuellen Aktion wird damit ignoriert. Diese Einstellung wird durch das Startdatum einer eventuell nächsten Aktion übersteuert (siehe *Beispiel-2*).

stepResultNames

Mit dem Dienst ist ein Alias für den Dienst *resultNames* und sollte nicht mehr verwendet werden, da er in der nächsten Hauptversion entfernt wird.

substitutes

Der Dienst liefert die Stellvertreter, die laut ihrer Vorgangsrechte den aktuellen Vorgang ausführen dürfen, in einem `Array`. Siehe dazu auch die Dokumentation Business-Process-Management, Kapitel *Stellvertreter* und *Rechte*.

undefineAdoptNextStep

Dieser Dienst widerruft eine zuvor getroffene Übernahme des nächsten Arbeitsschrittes (siehe *Beispiel-2*).

userEnvironment(<Name>)

Der Dienst liefert die Belegung der Umgebungsvariable `<Name>`. Auf Windows-Plattformen werden Umgebungsvariable unter **Einstellungen – Systemsteuerung – System – Umgebungsvariable** definiert. Auf Linux-Plattformen werden Umgebungsvariable im Loginscript festgelegt. Der Dienst liefert als Ergebnis `Null`, wenn die Umgebungsvariable nicht gefunden wurde.

userChoose

Der Dienst liefert den Namen für die Beendigung eines Arbeitsschrittes, mit welcher der Anwender in einem Beendigungsdialog das Arbeitsschrittergebnis und damit den nächsten auszuführenden Arbeitsschritt und soweit die entsprechenden Rechte vorliegen, einen anderen Arbeitsschritt, den ausführenden Bearbeiter und das Ausführungsdatum festlegen kann. Dieser Name wird bei Verwendung der Anweisung `Return` benutzt (siehe Kapitel *Return*).

userSelection

Der Dienst liefert den Namen für die Beendigung eines Arbeitsschrittes, mit welcher der Anwender in einem Beendigungsdialog das Arbeitsschrittergebnis und damit den nächsten auszuführenden Arbeitsschritt festlegen kann. Dieser Name wird bei Verwendung der Anweisung `Return` benutzt (siehe Kapitel *Return*).

visibleWorkers

Der Dienst liefert alle sichtbaren Bearbeiter in einem `Array`.

workers

Der Dienst liefert alle sichtbaren Bearbeiter, die laut ihrer Vorgangsrechte den aktuellen Vorgang ausführen dürfen, in einem `Array`. Details zu den Vorgangsrechten finden Sie in der Dokumentation Business-Process-Management, Stichwort *Sichtbarkeitsregeln* und *Rechte*. Ein Anwendungsbeispiel sehen Sie im *Beispiel-3*.

workingWorker

Der Dienst liefert den wirklichen aktuellen Bearbeiter des Vorgangs. Wenn ein Aufgabenbereich aktiv ist, differiert dieser Bearbeiter von der Systemvariablen `worker`.

version

Der Dienst liefert die Version des laufenden Systems als Zeichenkette. Z.B.: "2.90". Damit kann geprüft werden, ob für ein Makro die erforderliche Version von OfficeTalk vorliegt.

Beispiel-1

Das Beispiel zeigt die Anwendung der Dienste *stepResultNames* und *nextStepWorkerFor(<Name>)*.

```
Dim ergebnisListe As Array
Dim ergebnis As String
Dim nextBearbeiterName As String

ergebnisListe = stepscheduler.stepResultNames
dialog = New Joops.Scripting.ScriptDialog
dialog.title( "Qualifizieren Sie das Ergebnis" )
dialog.addLabel( "Ergebnisart", left: 10, top: 5 )
dialog.addList( ergebnisListe, named: "ergebnis", left: 80, top: 5,
width: 200, height: 70 )
dialog.addAcceptButtonLeft( 200, top: 70 )
dialog.width( 300 )
dialog.height(30 )
dialog.open
ergebnis = dialog.value( "ergebnis" )
nextBearbeiterName = stepscheduler.nextStepWorkerFor( ergebnis ).name
MsgBox( nextBearbeiterName )
Return ergebnis
```

Beispiel-2

Das Beispiel zeigt die Anwendung der restlichen Dienste.

```
Dim datum As Date
Dim zeit As Time

datum = Date + 2
zeit = stepscheduler.startAt.time
stepscheduler.startDate(datum, time: zeit)
stepscheduler.defineStartDate
stepscheduler.adoptNextStep
stepscheduler.consumed("Produktionskosten", amount: 200.00)
```

Beispiel-3

Das Beispiel zeigt die Anwendung der Dienste *goTo(<Bearbeiter>)*, und *workers*.

```
Dim alleBearbeiter As Array
Dim dialog As ScriptDialog
Dim bearbeiter As Worker
```

... weitere Skriptaktivitäten ...

```
alleBearbeiter = stepscheduler.workers
dialog = New ScriptDialog
dialog.title("Wählen Sie den Bearbeiter")
dialog.addList(alleBearbeiter, named: "alle", left: 5, top: 5, width:
200, height: 200)
dialog.addAcceptButtonLeft( 160, top: 210 )
dialog.width(210)
dialog.height(250)
If dialog.open = True Then
    bearbeiter = dialog.value("alle")
    stepscheduler.goTo(bearbeiter)
End If
```

Beispiel-4

Das Beispiel zeigt die Anwendung der Dienste *automation(<Vorgang>)* und *isAutomation(<Vorgang>)*.

```
Dim vorgang As Process

... weitere Skriptaktivitäten ...

Try
    vorgang = Start "System" "Eskalation"
    stepscheduler.automation(vorgang)
    If isAutomation(vorgang) = True Then
        MsgBox("Automatische Bearbeitung für " & vorgang &
            "gestartet")
    End If
End Try
Catch
End Catch
```

Beispiel-5

Das Beispiel zeigt die Delegation eines neu gestarteten Vorgangs an einen anderen Bearbeiter, als im ersten Arbeitsschritt bestimmt ist.

Das fiktive Szenario:

Der erste Arbeitsschritt des Vorgangs **Auftrag - Bearbeitung** wird per Definition in der übergeordneten Abteilung **Vertrieb** für die weitere Bearbeitung abgelegt. Sie wollen aber, dass der Vorgang sofort an Ihren Kollegen **Klaus Martens**, der sich in der selben Abteilung wie Sie befindet, delegiert wird.

```
Dim vorgang As Process
Dim children As Array

... weitere Skriptaktivitäten ...

Try
    vorgang = Start "Auftrag" "Bearbeitung"
    children = worker.parent.children
```

```
count = children.size
For i = 1 To count Step 1
    If children(i - 1).fullName = "Martens Klaus"
        Then
            stepscheduler.go(vorgang, to: children(i - 1))
            MsgBox("Der Vorgang Auftrag-Bearbeitung wurde an Martens Klaus delegiert")
            Return True
        End If
    Next
End Try
Catch
End Catch
```

Beispiel-6

Das Beispiel zeigt die Formatierung einer Integervariable als Betrag mit Tausender- und Dezimaltrenner mit den Dienst *format(<Variable>, as: <Formattyp>, with: <Formatanweisung>)*. Dieser Dienst ist immer dann erforderlich, wenn Sie z.B. eine Zahlenvariable als Zeichenkette, formatiert als Betrag, benötigen.

```
Dim zahl As Integer
```

```
zahl = Call Script.Gehalt
MsgBox("Frau Meier erhält " & stepscheduler.format(zahl, as: "decimal",
with: "###,##0.00" & " Gehalt")
```


worker

Anweisungen des Skripts können über diese globale Variable auf den aktuellen Bearbeiter, der den Vorgang, ausführt, zugreifen.

Syntax

activeProcesses

Der Dienst liefert alle aktiven Vorgänge des Bearbeiters in einem Array.

Hinweis: Eingestellte Filter in der Aufgabenliste werden nicht berücksichtigt.

activeProcesses(<Vorgangsname>, timed: <Zeitraum>, priority: <Array>, logicalName: <zusätzlicher Name>)

Der Dienst liefert alle aktiven Vorgänge des Bearbeiters, die mit den angegebenen Werten übereinstimmen als Array. Mit dem Argument <Vorgangsname> werden die Namen der gesuchten Vorgänge benannt. Der Name muss im Format <Kategorie>-<Bezeichnung> angegeben werden. Das Argument <Zeitraum> bezeichnet den gewünschten Ausführungszeitraum der gesuchten Vorgänge. Folgende Zeiträume sind möglich:

Zeitraum	Bedeutung
alle	Alle Vorgänge
diese Woche	Alle Vorgänge, deren nächster Arbeitsschritt innerhalb diese Woche ausgeführt werden muss.
dieses Monat	Alle Vorgänge, deren nächster Arbeitsschritt innerhalb dieses Monats ausgeführt werden muss.
dieses Jahr	Alle Vorgänge, deren nächster Arbeitsschritt innerhalb dieses Jahres ausgeführt werden muss.
bis gestern	Alle Vorgänge, deren nächster Arbeitsschritt bis gestern ausgeführt werden musste.
bis heute	Alle Vorgänge, deren nächster Arbeitsschritt bis heute ausgeführt werden muss.
nur heute	Alle Vorgänge, deren nächster Arbeitsschritt heute ausgeführt werden muss.
ab heute	Alle Vorgänge, deren nächster Arbeitsschritt ab heute ausgeführt werden muss.

Für das Argument <Array> geben Sie die Unter- und Obergrenze der Ausführungsriorität der gesuchten Vorgänge an.

Für das Argument <zusätzlicher Name> geben Sie den zusätzlichen Namen der gesuchten Vorgänge an. Sie können hier die Wildcards * und ? verwenden.

Hinweis: Wenn Sie nur bestimmte Vorgänge wünschen, arbeitet dieser Dienst wesentlich schneller als der allgemeinere Dienst *activeProcesses*.

addMacro(<Makroname>, source: <Quelltext>, script: <Skriptname>)

Der Dienst fügt im Bearbeiter das Makro mit dem Quelltext <Quelltext> ein. <Makroname> benennt den Namen des Makros. <Skriptname> benennt das Skript, in dem das Makro eingefügt wird. Falls das Skript noch nicht vorhanden ist, wird es angelegt. Falls das Makro Argumente besitzt, sind diese Teil des vollständigen Namens ! Argumente des Makros sind in der ersten Zeile des Quelltextes ersichtlich. Das Ergebnis des Dienstes ist

True, wenn das Makro ordnungsgemäß angelegt wurde. Existiert das Makro bereits, wird dessen Quelltext überschrieben.

adress

Mit dem Dienst erhalten Sie das Adressobjekt des Bearbeiters. Für einzelne Adressfelder des Bearbeiters müssen Sie den Dienst *adress* mit dem entsprechenden Adressfelddienst kaskadieren. Z B. `worker.adress.firstName` liefert den Vornamen des Bearbeiters. Folgende Adressfelddienste sind verfügbar:

<code>description</code>	die Kurzbeschreibung
<code>firstName</code>	der Vorname
<code>surName</code>	der Nachname
<code>street</code>	die Straße incl. Hausnummer
<code>zipCode</code>	die Postleitzahl
<code>town</code>	der Ort
<code>eMail</code>	die E-Mail-Adresse
<code>phone</code>	die Telefonnummer
<code>fax</code>	die Faxnummer

allChildren

Der Dienst liefert alle in der Hierarchie untergeordneten Bearbeiter.

allChildren(Bearbeiterart)

Der Dienst liefert alle in der Hierarchie untergeordneten Bearbeiter, die der angegebenen Bearbeiterart entsprechen. Die Bearbeiterart wird als *String* übergeben.

Bearbeiterart	Beschreibung
Desk	Nur Schreibtische werden geliefert
Machine	Nur Maschinen werden geliefert
Department	Nur Abteilungen werden geliefert
Office	Nur Büros werden geliefert
Team	Nur Teams werden geliefert

allParents

Der Dienst liefert alle in der Hierarchie übergeordneten Bearbeiter bis zum Unternehmen.

children

Der Dienst liefert die in der Hierarchie direkt untergeordneten Bearbeiter.

company

Der Dienst liefert den in der Hierarchie obersten Bearbeiter, das Unternehmen.

description

Der Dienst liefert die Beschreibung des Bearbeiters.

identify

Der Dienst identifiziert den Bearbeiter durch die Anforderung seines Logins mit einen Identifizierungsdialog. Das Ergebnis der Identifizierung ist `Boolean`. Wenn das darin angegebenen Login mit dem Login des Bearbeiters übereinstimmen antwortet der Dienst `True`, ansonsten `False`.

identity

Der Dienst liefert die eindeutige Identität des Bearbeiters in der Datenbank als `Integer`.

isActive

Die Dienste liefern `TRUE`, wenn der Bearbeiter ein Schreibtisch oder eine Maschine ist.

isCompany, isDepartment, isOffice, isTeam, isDesk, isMachine

Die Dienste liefern `TRUE`, wenn der Bearbeiter vom entsprechenden Type ist. Damit können typabhängige Aktionen ausgeführt werden.

isLoggedIn

Die Dienste liefern `TRUE`, wenn der Bearbeiter angemeldet ist.

isMacro(<Makroname>, source: <Quelltext>, script: <Skriptname>)

Der Dienst liefert `True`, wenn das Makro mit dem Namen <Makroname> im Skript <Skriptname> des Bearbeiters vorhanden ist. Da die Argumente des Makros Teil des Namens sind, benötigt der Dienst auch den Quelltext.

isPassive

Die Dienste liefern `TRUE`, wenn der Bearbeiter ein Unternehmen, eine Abteilung, ein Büro oder ein Team ist.

labelGroup

Der Dienst liefert die Textgruppe des Bearbeiters als Zeichenkette. Die Textgruppe des Bearbeiters wird verwendet, um die Namen der Bezeichner in Vorgangsdaten variabel zu gestalten. Die Textgruppe wird indirekt auch von den Diensten `label (<Name>, ifNone: <Variable>)`, `label (<Name>)` und `addLabel (<Bezeichnung>, left: <Ganzzahl>, top: <Ganzzahl>, key: <Label>)` verwendet.

Hinweis: Die Datei der Textgruppen muss im Zeichenformat UTF-8 erstellt sein !

macroNamesIn(<Skriptname>)

Der Dienst liefert in einem `Array` die Namen der Makros aus dem Skript mit dem Namen <Skriptname> des Bearbeiters. Der Dienst liefert ein leeres `Array`, wenn das Skript keine Makros enthält, oder im Bearbeiter nicht existiert.

name

Mit dem Dienst erhalten Sie den vollständigen Namen des Bearbeiters:

Unternehmen:	Der Unternehmensname
Abteilung	Die Abteilungsbezeichnung

Büro:	Die Bürobezeichnung
Team:	Die Teambezeichnung
Schreibtisch:	Nachname und Vorname des Bearbeiters
Maschine:	Die Maschinenbezeichnung

parent

Der Dienst liefert den in der Hierarchie übergeordneten Bearbeiter.

processTemplates

Der Dienst liefert alle Vorgangsvorlagen des Bearbeiters als Array.

refreshActiveProcesses

Der Dienst aktualisiert die Liste der aktiven Vorgänge des Bearbeiters. Um alle aktuellen Vorgänge eines Bearbeiters zu erhalten, sollte vor der Verwendung des Dienstes *activeProcesses* dieser Dienst ausgeführt werden. Ein kleines Beispiel

```
Dim visibleWorkers As Array
Dim i As Integer
Dim theWorker As Worker
Dim processes As Array

visibleWorkers = stepscheduler.visibleWorkers
For i = 1 To visibleWorkers.size Step 1
    theWorker = visibleWorkers( i - 1 )
    If Not theWorker Is worker
        Then theWorker.refreshActiveProcesses
    End If
    processes = theWorker.activeProcesses
...
Next
```

scriptNames

Der Dienst liefert in einem Array die Namen aller Skripts des Bearbeiters. Der Dienst liefert ein leeres Array, wenn der Bearbeiter keine Skripts besitzt.

shortcut

Der Dienst liefert den Anmeldenamen für das Login des Bearbeiters.

visibleWorkers(<Variable>)

Der Dienst liefert alle sichtbaren Bearbeiter. Wenn Sie <Variable> mit **True** belegen, werden nur die aktiven Bearbeiter (Schreibtisch, Maschine) geliefert. Wenn Sie <Variable> mit **False** belegen, werden nur die passiven Bearbeiter (Unternehmen, Abteilung, Büro Team) geliefert.

WSDLClient

Mit der Systemvariable können Sie Services aus der Internetgemeinde, auch Webservice genannt, verwenden. Diese Webservices müssen das SOAP (**S**imple-**O**bject-**A**ccess-**P**roto-**c**ol)- Interface unterstützen und es muss eine WSDL-Beschreibung (**W**eb**S**ervice-**D**efinition-**L**anguage) dafür vorliegen. Diese Beschreibung wird in vielen Fällen vom Serviceprovider zur Verfügung gestellt. i.d.R. erhalten Sie vom Provider einen Zugriffscode zur Benützung des Webservices. Um die Ergebnisse eines Services sinnvoll weiter zu verarbeiten, müssen Sie deren Aufbau kennen (siehe Dienst *value(<Variable>)*).

Hinweis: Um Webservices verwenden zu können, muss mit dem Dienst *url(<Variable>)* zuerst die WSDL-Beschreibung geladen werden.

Syntax

```
Dim client As Joops.Scripting.WSDLClient
```

containsStruct

Der Dienst antwortet *True* , wenn das Ergebnis des Webservices ein *WSDLStruct* ist und *False*, wenn das Ergebnis ein scalarer Wert ist.

disconnect

Der Dienst trennt die Internetverbindung, die durch den Dienst *url(<Variable>)* aufgebaut wurde.

hasValue

Der Dienst antwortet *True* wenn ein Ergebnis vorliegt, ansonsten *False*.

request(<Variable>)

Mit dem Dienst wird der benannte Webservice ausgeführt. *<Variable>* gibt den Namen des Services an und muss vom Typ *String* sein. Falls bei der Ausführung des Dienstes ein Fehler auftritt, wird eine Ausnahme geworfen, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann der Fehler statt dessen programmatisch bearbeitet werden.

Hinweis: Siehe Dienst *url(<Variable>)*.

request(<Variable>, with: <Array>)

Mit dem Dienst wird der benannte Webservice mit Argumenten ausgeführt.

<Variable> gibt den Namen des Services an und muss vom Typ *String* sein.

<Array(<Argument>[,...]> enthält die für den Service erforderlichen Argumente. Die Reihenfolge der Argumente ist in der WSDL-Definition spezifiziert.

<Argument>

<skalares Datum> Der Argumentwert

{*Variable* | *Literal* | *Ausdruck* | *Funktion*}

<komplexes Datum> Der komplexe Argumentwert

Array(<Argumentname>, <Datum>)

<Argumentname> Der Name des Argumentes lt. WSDL-Spezifikation als *String*

<Datum> Der Argumentwert

{Variable | Literal | Ausdruck | Funktion}

Hinweis: Siehe auch Dienst *url(<Variable>)*.

Wenn ein Argument eine Zeichenkette (String) mit Sonderzeichen enthält, z.b. Umlaute, müssen diese Zeichen HTML-konform codiert werden (siehe Kapitel *asHTMLString(<Variable>)*).

saveSchemaInto(<Variable>)

Der Dienst speichert die geladene WSDL-Beschreibung in die mit <Variable> benannte Datei. Die WSDL-Beschreibung wird durch den Dienst *url(<Variable>)* geladen.

url(<Variable>)

Mit dem Dienst bestimmen Sie den Link zur WSDL-Beschreibung der Services. Für <Variable> können Sie einen lokalen (file:///...) oder entfernten Link (http://...), der die Beschreibung enthält, angeben. Die Beschreibung enthält den Namen, die Argumente und die Rückgabewerte des Webservices. Der Provider stellt i.d.R. diese Beschreibung zur Verfügung. Falls bei der Ausführung des Dienstes ein Fehler auftritt, wird eine Ausnahme geworfen, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann der Fehler statt dessen programmatisch bearbeitet werden.

Hinweis: Mit dem Dienst muss vor der Verwendung des Dienstes *request(<Variable>)* zuerst die WSDL-Beschreibung geladen werden. Dazu muss mit dem **Benutzer** und **Passwort** aus der Registerseite **HTTP** der allgemeinen Einstellungen in OfficeTalk eine Anmeldung beim Verbindungsprovider erfolgen. Falls eine der Angaben in der Registerseite fehlt, werden Sie mit dem Dialog **HTTP-Anmeldung** danach gefragt. Anschließend wird die WSDL-Beschreibung über den angegebenen Link geladen, was einige Zeit in Anspruch nehmen kann.

value

Der Dienst liefert das Ergebnis der Webservices. Der Dienst ist nur dann sinnvoll anzuwenden, wenn das Ergebnis des Services scalar ist.

value(<Variable>)

Der Dienst liefert das Ergebnis des Webservices mit dem Namen <Variable>. Die Variable muss vom Typ String sein. Der String kann die Wildcards * und ? enthalten. Webservices, die mit mehreren Teilergebnissen antworten, benennen diese Teilergebnisse. Ein Teilergebnis kann auch seinerseits wiederum eine Liste von Teilergebnissen enthalten, usw. Um ein Teilergebnis aus tieferer Ebene zu erhalten, müssen Sie entweder den nächsten Dienst verwenden, oder mit Hilfe der Systemvariablen *WSDLStruct* durch die Hierarchie wandern.

value(<Variable>, deep: <bool-Ausdruck>)

Der Dienst liefert das Ergebnis des Webservices mit dem Namen <Variable>. Der String kann die Wildcards * und ? enthalten. Wenn Sie für das Argument *deep: True* angeben, wird das erste Teilergebnis mit dem Namen <Variable> innerhalb der gesamte Ergebnishierarchie geliefert. Bei *False* für das Argument *deep:* (siehe Dienst *value(<Variable>)*).

values

Der Dienst liefert alle Ergebnisse des Webservices als Array. Weitere Einzelheiten dazu siehe Kapitel *values* der Systemvariablen *WSDLStruct*.

Beispiel

GoogleSpell

`Ein sehr einfaches Beispiel eines Webservices. Es benutzt die
`Rechtschreibung von Google. Für <Google-Key> ist die Autorisierung
`für die Verwendung des Webservices einzusetzen. Den Autorisierungs-
`schlüssel erhalten Sie durch die Anmeldung bei Google.

`

```
Dim client As Joops.Scripting.WSDLClient
client = New Joops.Scripting.WSDLClient
Try
    client.url( "http://api.google.com/GoogleSearch.wsdl" )
    client.request( "doSpellingSuggestion", with: Array( <Google-Key>,
"basebal" ) )
    MsgBox("Ergebnis: " & client.value)
End Try
Catch
    MsgBox("Fehler: " & error.Description)
End Catch
```

ELSEHallo

`Ein Beispiel zur Ausführung des Services HelloWorldResult
`im ELSEService

`

```
Dim cvv As WSDLClient
cvv = New Joops.Scripting.WSDLClient
cvv.url( "http://www.lieferscheinservice.de/ELSEService.asmx?WSDL" )
cvv.request( "HelloWorld", with: Array( "" ) )
return cvv.value("HelloWorldResult", deep: true)
```

ELSEAuthorization

`Ein Beispiel zur Autorisierung mit dem Service Authorization
`im ELSEService. Das Makro liefert das Bearbeitungsticket des Services

`

```
Dim cvv As WSDLClient
cvv = New Joops.Scripting.WSDLClient
cvv.url( "http://www.lieferscheinservice.de/ELSEService.asmx?WSDL" )
cvv.request( "Authorization", with: Array(Array (Array( "username" ,
"demo"), Array( "password" , "demo") ))
return cvv.value("Ticket", deep: true)
```

WSDLStruct

Mit der Systemvariable können Sie sich in Teilergebnissen eines Webservices bewegen. Um die Ergebnisse eines Webservices damit sinnvoll weiter zu verarbeiten, müssen Sie deren Aufbau kennen (siehe Dienst *value* und *value(<Variable>)* der Systemvariablen *WSDLClient*).

Syntax

```
Dim client As Joops.Scripting.WSDLStruct
```

containsStruct

Der Dienst antwortet mit *True*, wenn das Ergebnis selbst wieder eine *WSDLStruct* ist, *False* wenn das Ergebnis ein scalarer Wert ist.

value

Der Dienst liefert das Teilergebnis. Wenn das Ergebnis ein zusammengesetztes Teilergebnis ist, erhalten Sie wiederum eine Variable vom Typ *WSDLStruct*, ansonsten den scalaren Wert.

value(<Variable>)

Der Dienst liefert das Ergebnis mit dem Namen <Variable>. Die Variable muss vom Typ *String* sein. Der *String* kann die Wildcards *** und *?* enthalten. Ein Teilergebnis kann auch seinerseits wiederum eine Liste von Teilergebnissen enthalten, usw. Um ein Teilergebnis aus tieferer Ebene zu erhalten, können Sie sich mit dieser Systemvariable in eine tiefere Ebene steigen.

value(<Variable>, deep: <bool-Ausdruck>)

Der Dienst liefert das Ergebnis mit dem Namen <Variable>. Der *String* kann die Wildcards *** und *?* enthalten. Wenn Sie für das Argument *deep: True* angeben, wird das erste Teilergebnis mit dem Namen <Variable> innerhalb der gesamte Ergebnishierarchie geliefert. Bei *False* für das Argument *deep*: siehe Dienst *value(<Variable>)*.

values

Der Dienst liefert alle Ergebnisse als *Array*. Das *Array* enthält für scalare Datentypen der Wert selbst. Für Teilergebnisse enthält das *Array* eine Variable vom Typ *WSDLStruct*.

Besonderheiten

Der Dienst ist nur nach Ausführung des Dienstes *url(<Variable>)* erlaubt.

Beispiel

```
`Ein sehr einfaches Beispiel eines Webservices. Es sucht bei Google  
`Smalltalk Consultants. Für <Google-Key> ist die Autorisierung  
`für die Verwendung des Webservices einzusetzen. Den Autorisierungs-  
`schlüssel erhalten Sie durch die Anmeldung bei Google.  
`
```

```
Dim client As Joops.Scripting.WSDLClient  
Dim struct As Joops.Scripting.WSDLStruct  
Dim args As array
```



```
Dim results As Array

client = New Joops.Scripting.WSDLClient
args = Array( <Google-Key>,"Smalltalk Consultant", 0 ,10,
true,"",false, "", "", "")
client.url("http://api.google.com/GoogleSearch.wsdl")
struct = client.request("doGoogleSearch", with: args)
results = struct.value("resultElements")
struct = results(5)
MsgBox(struct.value("snippet")) `snippet des 5. Ergebnisses
MsgBox(client.value("snippet", deep: True)) `snippet des 1. Ergebnisses
results = struct.values
If results(0) Is WSDLStruct = True Then
    MsgBox("Teilergebnis: " & results(0).values)
Else
    MsgBox("Ergebnis: " & results(0))
End If
```

Dynamisch geladene Variablentypen

Um mit externen Systemen zu kommunizieren oder Sie zu steuern, können auf Windows-Plattformen Programmteile dynamisch geladen werden. Dadurch werden Datentypen/-Klassen geladen, die für die Kommunikation und Steuerung des externen Systems verwendet werden können.

Kommunikationsbibliothek

Mit der Direktive *Library* wird eine Kommunikationsbibliotheksdatei mit der Dateinamenserweiterung *.pcl* geladen. Die Bibliothek enthält systembezogene Variablentypen, auch Klassen genannt. Wenn eine Kommunikationsbibliothek geladen ist, können Sie die darin enthaltenen Variablentypen verwenden. Der Name des Variablentyps aus einer Bibliothek lautet `SmallCOM.<System>.<Type>`. Dabei ersetzen Sie `<System>` durch den Applikationsnamen des Systems und `<Type>` durch den gewünschten Typ. Entnehmen Sie die zu verwendenden Namen der Dokumentation zur Kommunikationsbibliothek. Bei der Eingabe der Anweisungen `Dim` und `New` während der Makroerstellung bietet Ihnen der Skripteditor über die Onlinehilfen eine Auswahlliste mit den Datentyp/Klassennamen aus der geladenen Kommunikationsbibliotheksdatei an. Ein kleines Beispiel für die Verwendung von geladenen Variablentypen:

PrintWordDocument

```
Library "..\Library\Microsoft Word Small.pcl"

Dim word As SmallCOM.Word.Application
Dim documents As SmallCOM.Word.Documents
Dim document As SmallCOM.Word.Document

word = New SmallCOM.Word.Application
documents = word.documents
document = documents.open( "..\Vorlagen\Anschreiben.doc" )

... weitere Anweisungen...

document.printOut

word.quit( False,originalFormat: Enum(SmallCOM.Word.WdOriginalFormat,
                                     wdWordDocument ),    routeDocument: False )
```

Automatisation

Auf Windows-Plattformen kann durch die Direktive *Interface* die Verbindung zu einem installierten System mit Automatisierungsschnittstelle hergestellt werden. Ob ein System automatisationsfähig ist, erfahren Sie von dessen Hersteller. Nach Ausführung der Direktive kann der Interfacename in `Dim`- und `New`-Anweisungen als Datentyp/Klasse verwendet werden. Der Skripteditor unterstützt keine Onlinehilfen für die verfügbaren Dienste. Informationen über die verfügbaren Dienste der Automatisation müssen Sie entsprechenden Dokumentationen des Herstellers entnehmen. Im Kapitel *Beispiel* finden Sie ein Beispiel für die Verwendung der *Interface*-Direktive.

Anweisungen

Mit Anweisungen wird den Ablauf eines Makros gesteuert. Anweisungen bestimmen die einzelnen Aktivitäten eines Makros.

= (Zuweisung)

Die Anweisung weist einer Variablen das Ergebnis eines Ausdrucks zu.

Syntax

`<Variable> = <Ausdruck>`

<Variable>

Der Name der Variablen

<Ausdruck>

{Variable | Literal | Ausdruck | Funktion | New-Anweisung | Open-Anweisung}

Der Ausdruck, mit dem die Variable belegt wird.

Besonderheiten

Bei der Zuweisung einer Variablen erfolgt eine automatische und typgerechte Konvertierung des Datums. Der Datentyp der Variablen links vom Gleichheitszeichen bestimmt die Konvertierungsart. Ist keine typgerechte Konvertierung möglich, wird eine Ausnahme geworfen (siehe Kapitel *Behandlung fehlerhafter Argumente*).

Konvertierungsbeispiele

```
Dim anInteger As Integer
```

```
Dim aLong As Long
```

```
Dim aFloat As Float
```

```
Dim aDouble As Double
```

```
Dim aString As String
```

```
Dim aDate As Date
```

```
Dim aTime As Time
```

```
anInteger = 34567898
```

```
aLong = 1999888776655
```

```
aFloat = 987234.789
```

```
aDouble = 7654987654323456.98
```

```
aString = "Das ist der Test"
```

```
aDate = Date
```

```
aTime = Time
```

```
aString = anInteger      '-> "34567898"
```

```
aString = aFloat         '-> "9.872.347,79"
```

```
aString = aDouble        '-> "7.654.987.654.323.456,98"
```

```
aString = aDate           '-> "1 Januar 2008"
aString = aTime           '-> "10:12:30"
anInteger = aDouble       '-> Verlust der Stellengenauigkeit/Fehlermeldung

aDouble = anInteger       '-> "34567898.00"
aString = aDouble         '-> "34.567.898,00"

anInteger = 1234.56       '-> Zuweisung erfolgt als Ganzzahl
aString = anInteger       '-> "1234"
```

Beispiel

```
Dim alter As Integer
Dim zahl As Integer
zahl = 10
alter = zahl + 10
```

Close

Die Anweisung schließt eine zuvor geöffnete Datei.

Syntax

Close <Variable>

1. <Variable>
Die Variable, mit der die Datei geöffnet wurde.

Beispiel

```
Dim datei As File
datei = Open "c:\FileIOtest.txt" For Output
Close datei
```

Do Until

Die Anweisung führt einen Anweisungsblock aus, bis eine beschriebene Bedingung eingetreten ist. Die Bedingung wird vor Ausführung der ersten Anweisung im Block geprüft.

Syntax

Do Until <Ausdruck> <Anweisung,...> Loop

1. Ausdruck {bool-Ausdruck|bool-Funktion}
Eine Ausdruck, dessen Ergebnis True oder False ergibt.
2. Anweisung
Ein oder mehrere Anweisungen, die abgearbeitet werden, solange das Ergebnis von Ausdruck True ist.

Beispiel

```
Dim zahl As Integer
zahl = 10
```

```
Do Until zahl < 5
    zahl = zahl - 1
MsgBox("4 ist noch nicht erreicht")
Loop
```

Do While

Die Anweisung führt einen Anweisungsblock aus, solange eine beschriebene Bedingung zutrifft. Die Bedingung wird nach Ausführung der letzten Anweisung im Block geprüft.

Syntax

```
Do While <Ausdruck> <Anweisung,...> Loop
```

1. <Ausdruck>
{bool-Ausdruck|bool-Funktion}
Ein Ausdruck der True oder False liefert
2. <Anweisung>
Ein oder mehrere Anweisung, die abgearbeitet werden, solange das Ergebnis von Ausdruck True ist.

Beispiel

```
Dim zahl As Integer
zahl = 1
Do While zahl < 5
    zahl = ( zahl + 1 )
Loop
```

Exit Do

Die Anweisung bricht die Ausführung des Anweisungsblocks *Do While* oder *Do Until* ab. Die Bearbeitungsschleife wird damit beendet.

Syntax

```
Exit Do
```

Beispiel

```
Dim zahl As Integer
zahl = 1
Do While zahl < 5
    zahl = ( zahl + 1 )
    If zahl > 3 Then
        Exit Do
    End If
Loop
```

Exit For

Die Anweisung bricht die Ausführung des Anweisungsblocks aus *For Next* ab. Die Bearbeitungsschleife wird damit beendet.

Syntax

```
Exit For
```

Beispiel

```
Dim zahl As Integer
For zahl = 1 To 10 Step 1
    If MsgBox( "For-Loop beenden", vbYesNo ) = 1 Then
        Exit For
    Next
```

For Next

Die Anweisung wiederholt einen Anweisungsblock und inkrementiert bei jedem Durchlauf den Startwert um die Schrittweite solange, bis der Startwert den Endwert erreicht hat.

Syntax

```
For <Startwert> To <Endewert> [ Step <Schrittweite> ]
    <Anweisung...>
Next
```

1. Startwert
<Assign-Anweisung>
Die Variable mit dem Startwert muss mit einer Assign-Anweisung initialisiert werden.
2. Endewert
{num.-Ausdruck|Variable|Literal}
Der Ausdruck liefert die Zahl, bis zu deren Erreichen die Schleife wiederholt wird. Die Überprüfung auf Erreichen findet zum Schleifenbeginn statt.
3. Schrittweite
<num. Ausdruck>
Der Ausdruck liefert den Wert, um den die Schrittweite bei jedem Durchlauf erhöht wird. Die Angabe der Schrittweite ist optional. Fehlt die Angabe, wird 1 als Schrittweite verwendet.
4. Anweisung
Der Anweisungsblock mit ein oder mehreren Anweisungen

Beispiel

```
Dim zahl1 As Integer
Dim zahl2 As Integer

zahl2 = 1
For zahl1 = ( 1 + 1 ) To 10 Step 1
    zahl2 = ( zahl2 + 3 )
```

```
If MsgBox( "For-Loop beenden", vbYesNo ) = 1 Then
    Exit For
End If
Next
```

Halt

Die Anweisung startet an den Debugger und öffnet seinen Dialog. Den weiteren Ablauf des Makros können Sie daraus steuern.

Syntax

```
Halt
```

Beispiel

```
Dim zahl1 As Integer
Dim zahl2 As Integer

zahl2 = 1
Halt
For zahl1 = ( 1 + 1 ) To 10 Step 1
    zahl2 = ( zahl2 + 3 )
    If MsgBox( "For-Loop beenden", vbYesNo ) = 1 Then
        Exit For
    End If
Next
```

If

Die Anweisung arbeitet je nach Ergebnis eines Ausdrucks einen Anweisungsblock ab.

Syntax

```
If <Ausdruck> Then <Anweisung...> [ Else <Anweisung...> ] End If
```

1. Ausdruck
{bool. Ausdruck|bool Funktion}
Wenn das Ergebnis des Ausdrucks True ist, werden die Anweisungen im Then-Block, ansonsten die Anweisungen im Else-Block, falls ein solcher angegeben ist, ausgeführt.
2. Anweisung
Ein oder mehrere Anweisungen

Beispiel

```
Dim zahl1 As Integer
Dim zahl2 As Integer

zahl1 = 10
zahl2 = 20
If ( zahl1 + zahl2 ) > 100 Then
```

```
MsgBox( „zu klein“)  
Else  
    MsgBox( „zu groß“)  
End If
```

Input

Die Anweisung liest eine Datei Elementweise und überträgt die einzelnen Elemente in die angegebenen Variablen. Die Datei muss im Textmodus beschrieben worden sein. i.d.R. sollte die Datei mit der Anweisung *Write* erstellt worden sein.

Syntax

```
Input <Datei>,<Variable>[,...]
```

1. Datei
 <Variable>
 Die Variable, mit der die Datei geöffnet wurde.
2. Variable
 <Variable>
 Die Variable, in die der Inhalt eines Dateielementes gelesen wird.

Beispiel

```
Dim datei As File  
Dim zahl As Integer  
Dim text As String  
datei = Open "c:\FileIOtest.txt" For Input  
Input datei, text, zahl  
Close datei  
MsgBox( Array( text, " " , zahl ) )
```

Line Input

Die Anweisung liest eine Datei zeilenweise in die angegebene Variable. Die Datei muss im Textmodus beschrieben worden sein. i.d.R. sollte die Datei mit der Anweisung *Print* erstellt worden sein.

Syntax

```
Line Input <Datei>,<Variable>
```

1. Datei
 <Variable>
 Die Variable, mit der die Datei geöffnet wurde.
2. Variable
 <Variable>
 Die Variable, in die der Inhalt der nächsten Zeile gelesen wird.

Beispiel

```
Dim datei As File  
Dim text As String  
datei = Open "c:\FileIOtest.txt" For Input
```



```
Line Input datei, text
Close datei
MsgBox( text )
```

New

Die Anweisung liefert ein neues Objekt (Instanz) vom angegebenen Datentyp/Klasse. Bei Bedarf muss der Datentyp/Klasse zuvor durch die *Library*-Direktive bekannt gemacht werden. Der Skripteditor bietet Ihnen bei Eingabe von *New*, gefolgt von zwei Leerzeichen, eine Liste der möglichen Variablentypen zur Auswahl an.

Syntax

```
New <Typ>
```

1. Typ
<Datentypname>
Der Name des Typs (der Klasse), von der eine Instanz erzeugt werden soll. Den Klassennamen von OfficeTalk Datentypen können Sie voll qualifiziert (z.B. Joops.OfficeTalk.Desk) oder nur mit Namen (z.B. Desk) angeben. Die Namen von Typen aus einer geladenen Kommunikationsbibliothek beginnen immer mit SmallCOM..

Besonderheiten

OfficeTalk Datentypen können Sie sowohl voll qualifiziert mit führendem Namensraum (z.B. Joops.OfficeTalk.Desk) als auch ohne Namensraum (z.B. Desk) angeben.

Datentypen die mittels der *Library*-Direktive eingeführt wurden beginnen mit SmallCOM.<Bibliothek>. Bei einer Microsoft-Word-Bibliothek beginnt der Name mit SmallCOM.Word. Entnehmen Sie der Bibliotheksdokumentation die zu verwendenden Namen.

Wenn Sie einen unbekannten Datentypnamen eingetippt haben, erhalten Sie beim Formatieren oder Kompilieren des Makros eine Auswahlliste mit den gültigen Datentypen. Daraus können Sie durch Auswahl den Namen korrigieren. Die Schreibweise der Klassennamen ist nicht case sensitiv.

Beispiel

```
Dim word As SmallCOM.Word.Application
word = New SmallCOM.Word.Application
word.visible( True )
```

Open

Die Anweisung öffnet eine Datei im gewünschten Modus.

Syntax

```
Open <Datei> For <Öffnungsmodus> [Access <Lesemodus>] |
Open <Datei> For Binary Access <Lesemodus>
```

1. Datei
{Variable|Literal}
Die Variable, die den Namen der zu öffnenden Datei enthält.

Der Dateiname kann als Präfix einen symbolischen Verzeichnisnamen enthalten. Z. B.: \$(temp)\Planung.txt. Dabei wird \$(temp) in das eigentliche Verzeichnis übersetzt. Einzelheiten dazu finden Sie im Kapitel *resolve* (<Variable>).

2. Öffnungsmodus

Input	Öffnet die Datei im Lesemodus.
Output	Öffnet die Datei im Schreibmodus. Ein alter Inhalt wird gelöscht.
Append	Öffnet die Datei im Ergänzungsmodus. Der alte Inhalt bleibt bestehen.
Binary	Öffnet die Datei im Binärmodus. Dafür ist der Operand Lesemodus erforderlich.

3. Lesemodus

Read	Öffnet die Datei im Lesemodus
Write	Öffnet die Datei im Schreibmodus
ReadWrite	Öffnet die Datei im Lese-/Schreibmodus

Beispiel

```
Dim datei As File
datei = Open "c:\FileIOTest.txt" For Output
```

Print

Die Anweisung schreibt in eine Datei zeilenweise die angegebene Variable. Für alle Dateimodi ausser Binary (siehe Kapitel *Open*) wird die Zeile mit dem Zeilenende CR abgeschlossen.

Syntax

```
Print <Datei>,<Inhalt>
```

1. Datei

<Variable>

Die Variable, mit der die Datei geöffnet wurde.

2. Inhalt

<Variable>

Die Variable, deren Inhalt in die Datei geschrieben werden wird. Ist die Variable ein Array von Variablen, werden die einzelnen Variablen mit einer Mindestspaltenbreite von 14 ausgegeben, wenn die Datei nicht mit dem Modus Binary (siehe Kapitel *Open*) geöffnet wurde. Wurde die Datei mit dem Modus Binary geöffnet, wird der Binärewert der Variable in die Datei geschrieben.

Besonderheiten

Dateien, die mit dem Modus Binary (siehe Kapitel *Open*) erstellt werden, können nicht mit den Anweisungen *Input* und *Line Input* gelesen werden.

Beispiel

```
Dim datei As File
Dim text As String
datei = Open "c:\FileIOTest.txt" For Output
text = "Das ist eine Zeile"
Print datei, text
Close datei
```

Return

Ein Startmakro muss mit einer `Return`-Anweisung abschließen und bestimmt dadurch den nächsten auszuführenden Arbeitsschritt. Das Argument der Anweisung ist eine Variable vom Typ `String`, und benennt den Namen des zu verwendenden Arbeitsschrittergebnisses für den nächsten auszuführenden Arbeitsschritt. Wird ein Makro durch eine *Call-Makro*-Anweisung aufgerufen, ist die `Return`-Anweisung abhängig vom erwarteten Ergebnis des rufenden Makros.

Syntax

```
Return <Ergebnis>
```

1. Ergebnis
{Ausdruck|Funktion|Variable|Literal}
Das Ergebnis des Ausdrucks wird an den Aufrufer des Makros retourniert. In einem Startmakros bestimmt dieser Wert i.d.R. den Namen des Arbeitsschrittergebnisses des nächsten auszuführenden Arbeitsschritt.

Hinweis: Wird das Startmakro mit der Anweisung `Return stepscheduler.userSelection` oder `Return "****SelectionByUser****"` endet, öffnet sich ein Dialog, in dem der Benutzer das Arbeitsschrittergebnis und damit alle weiteren Eigenschaften des nächsten Arbeitsschrittes festlegen kann (siehe auch Kapitel *userSelection*).

Wird das Startmakro mit der Anweisung `Return stepscheduler.userChoose` oder `Return "****ChooseByUser****"` endet (siehe auch Kapitel *userChoose*), öffnet sich der Standarddialog zur Beendigung eines Arbeitsschrittes. Darin kann der Benutzer das Arbeitsschrittergebnis und damit alle weiteren Eigenschaften des nächsten Arbeitsschrittes festlegen. Wenn der Benutzer als angemeldeter Bearbeiter die entsprechenden Arbeitsschrittrechte besitzt, kann er darüber hinaus aus allen Arbeitsschritten des Vorgangs den nächsten auszuführenden Arbeitsschritt, dessen Bearbeiter und sein Startdatum bestimmen. Wird dabei ein nicht direkter Arbeitsschrittnachfolger gewählt, ist zu beachten, dass die darin vorausgesetzte Arbeitsumgebung (z.B. Vorgangsdaten) gegeben ist. Mit diesen Möglichkeiten kann der Benutzer den Ablauf der Arbeitsschritte abweichend von der vorgegebenen Struktur der Vorgangsvorlage beeinflussen.

Beispiel

```
Return "Interresiert"
```

Select Case

Mit der Anweisung können umfangreiche Fallunterscheidungen übersichtlich beschrieben werden.

Syntax

```
Select Case <Ausdruck> Case <Case-Ausdruck> [...] [Case Else <Else-Ausdruck>] End Select
```

1. Ausdruck
{Ausdruck|Funktion|Variable|Literal}
Das Ergebnis dieses Ausdrucks wird mit den nachfolgenden Case-Abschnitten verglichen.
2. <Case-Ausdruck>
Case <Vergleichsausdruck>
 <Anweisung> [...]
 <Vergleichsausdruck>
 <Ausdruck|Funktion|Variable|Literal>

Wenn <Vergleichsausdruck> mit <Ausdruck> übereinstimmt, werden die Anweisungen in <Anweisung> ausgeführt.

3. <Else-Ausdruck>

Case Else

<Anweisung> [...]

Wenn kein <Case-Ausdruck> übereinstimmt, werden die Anweisungen in <Anweisung> ausgeführt.

Beispiel

```
Dim x As Integer
x = 100
Select Case x
    Case "Test"
        MsgBox("x enthält Test")
    Case worker.displayString
        MsgBox("x enthält den Namen des aktuellen Bearbeiters")
    Case 100
        MsgBox("x enthält den 100")
    Case Else
        MsgBox("keine Übereinstimmung mit der Variable X")
End Select
```

Try-Catch

Die Anweisung führt einen Try-Anweisungsblock aus. Tritt während der Ausführung der Anweisungen ein Fehler auf, wird der Catch-Anweisungsblock ausgeführt. Innerhalb des Catch-Blockes steht die Systemvariable *Error* (oder *error*) zur Verfügung. Die Anweisung ist das Pendant zur VisualBasic Anweisung `On Error GoTo <Label>`.

Syntax

```
Try <Anweisung...> End Try Catch <Anweisung...> End Catch
```

Wenn in den Anweisungen des Try-Blocks ein Fehler auftritt, werden die Anweisungen im Catch-Block ausgeführt.

Beispiel

```
Dim file As File
Dim a As String

Try
    Write file, a
End Try

Catch
    MsgBox( Error.Description )
    MsgBox( Error.Source )
End Catch
```

While

Siehe Anweisung *Do While*.

Syntax

```
While <Ausdruck> <Anweisung,...> Wend
```

Siehe Anweisung *Do While*.

Besonderheiten

Dateien, die mit dem Modus `Binary` (siehe Kapitel *Open*) erstellt werden, können nicht mit den Anweisungen *Input* und *Line Input* gelesen werden.

Beispiel

```
Dim zahl As Integer
zahl = 1
While zahl < 5
    zahl = ( zahl + 1 )
Wend
```

Write

Die Anweisung schreibt in eine Datei Elementweise die angegebenen Variablen. Für alle Dateimodi ausser `Binary` (siehe Kapitel *Open*) wird der letzte Wert mit dem Zeilenende `CR` abgeschlossen.

Syntax

```
Write <Datei>,<Wert>[,...]
```

1. Datei
<Variable>
Die Variable, mit der die Datei geöffnet wurde.
2. Wert
{Ausdruck|Funktion|Variable|Literal}
Das Ergebnis des Ausdrucks wird in die Datei geschrieben. Mehrere Variable werden durch , (Komma) getrennt, wenn die Datei nicht mit dem Modus `Binary` (siehe Kapitel *Open*) geöffnet wurde. Wurde die Datei mit dem Modus `Binary` geöffnet, wird der Binärewert der Variable in die Datei geschrieben.

Beispiel

```
Dim datei As File
Dim zahl As Integer
Dim text As String
datei = Open "c:\FileIOtest.txt" For Output
Write datei, text, zahl
Close datei
```

Funktionen

An vielen Stellen, an denen ein Ausdruck erforderlich ist, kann auch eine Funktion verwendet werden.

Array

Mit der Funktion kann eine Liste von Variablen gebildet werden.

Syntax

```
Array ( <Ausdruck>[,...] )
```

1. Ausdruck
{Ausdruck|Funktion|Variable|Literal}
Das n-te Arrayelement wird mit dem Ergebnis des Ausdrucks belegt.

Beispiel

```
Dim familienname As Array  
familienname = Array ("Meier", " Hans" )  
MsgBox ( familienname )
```

Zugriffe auf Arrayelemente

Mit der Indexfunktion kann eine Variable in das Array geschrieben oder aus dem Array gelesen werden. Beim Schreiben einer Variablen wird das Array bei Bedarf automatisch vergrößert.

Syntax

```
<Array>( <Position> )
```

1. Array
<Variable>
Der Name einer Arrayvariablen
2. Position
{Ausdruck|Funktion|Variable|Literal}
Gibt die Position der Variablen innerhalb des Arrays an. Die Zählbasis ist 0.

Beispiel

```
Dim familienname As Array  
familienname = Array ()  
familienname( 0 ) = "Meier "  
familienname( 1 ) = "Hans"  
MsgBox ( ( familienname( 1 ) & familienname( 2 ) ) )  
familienname( 1 ) = "Herbert"  
MsgBox ( ( familienname( 1 ) & familienname( 2 ) ) )
```

Bemerkung

Zusätzlich zu den Zugriffsfunktionen unterstützt ein Array Dienste zur einfacheren Verwaltung seines Inhaltes. Siehe dazu Kapitel *Dienste des Datentyps Array*.

Call-Dienst

Die Funktion ruft den Dienst eines Objektes. Die Variable enthält das Objekt. Zu den Dienstnamen lesen Sie bitte die Dokumentation der entsprechenden Kommunikationsbibliothek oder der entsprechenden Systemvariablen. Die Dienstnamen der Kommunikationsbibliotheken für MS-Office sind von den VBA Funktionen abgeleitet. Für diese Bibliotheken können Sie sich also den Namen eines Dienstes aus der entsprechenden VBA-Funktion bilden. Die VBA-Funktion

```
quit( Bool, originalFormat, routeDocument)
```

der Komponente Application in MS-Word lautet als Dienst

```
quit(False,originalFormat:Enum(SmallCOM.Word.WdOriginalFormat,
                                wdWordDocument ), routeDocument: False).
```

Der Skripteditor bietet Ihnen nach Eingabe von Variablenname., gefolgt von einem Leerzeichen, eine Liste der möglichen Dienste des Objektes zur Auswahl an. Der ausgewählte Dienst wird nach dem . eingefügt. Sie müssen nur noch die Argumentmuster incl. <> durch die konkreten Argumente ersetzen.

Behandlung fehlerhafter Argumente

I.d.R. werden die Argumente eines Dienstes nicht geprüft. Wird ein falsches Argument (z.B. unverträglicher Datentyp) verwendet, resultiert dies erst während der Ausführung in einer Ausnahme. Dadurch erhalten Sie eine Fehlermeldung, die nicht unbedingt sofort auf das falsche Argument hinweist. Wird jedoch der Dienst einer Systemvariablen ausgeführt und dieser Dienst verwendet ein falsches Argument, wird vor der eigentlichen Ausführung eine Ausnahme geworfen. Wenn der Dienst innerhalb eines Try-Catch-Blockes ausgeführt wird, wird die Ausführung des Makros mit dem Catch-Block fortgesetzt. Ansonsten wird das fehlerhafte Argument mit einer aussagefähigen Fehlermeldung, in der Sie zwischen **Ignorieren**, **Abbrechen** und **Debuggen** wählen können, moniert. In einigen Diensten wird auch eine Ausnahme bei Verwendung logisch falscher Argumente geworfen (siehe Dienst `scheduleAt(<Datum>, time: <Zeit>, comment: <Kommentar>)`).

Verletzung von Ausführungsbedingungen

Die Ausführung einiger Dienste unterliegt Bedingungen. Z.B.: Dürfen einige Dienste nur in einer vorgeschriebenen Umgebung verwendet werden. Werden bei der Ausführung des Dienstes diese Bedingungen verletzt, wird eine Ausnahme geworfen. Wenn die Verletzung der Bedingung innerhalb eines Try-Catch-Blockes auftritt, wird die Ausführung des Makros mit dem Catch-Block fortgesetzt. Ansonsten erhalten Sie eine aussagefähige Fehlermeldung, in der Sie zwischen **Ignorieren**, **Abbrechen** und **Debuggen** wählen können. Die Bedingungen zur Ausführung eines Dienstes sind bei seiner Beschreibung zu finden.

Syntax

```
<Variable>.<Dienst>[( <Argument>[,<Bezeichner>: <Argument>,...][...]]
```

1. Variable
Der Name der Variablen, die das Objekt enthält, welches den Dienst ausführen soll.
2. Dienst
Der Name des Dienstes
3. Bezeichner
Die Argumentbenennung
Mit Ausnahme des ersten Argumentes werden alle weiteren Argumente benannt. Die Benennung wird mit dem Zeichen : abgeschlossen.
4. Argument
{Ausdruck|Funktion|Variable|Literal}
Der Argumentwert des Dienstes.

Beispiel

Der aktuelle Bearbeiter kennt den Dienst „Name als String“

```
MsgBox( worker.name )
```

Die nachfolgende Kaskadierung liefert das Startdatum des Arbeitsschrittes‘

```
Dim datum As Date
```

```
datum = stepscheduler.startAt.date
```

Bemerkung

Die Schreibweise der Dienstenamen sind nicht case sensitiv. Dienste können, wie in dem Beispiel gezeigt, kaskadiert werden.

Call-Makro

Die Funktion ruft das Makro eines Skripts auf. Die Funktion liefert als Ergebnis das Argument der Return-Anweisung aus dem gerufenen Makro, oder Null, wenn das gerufene Makro nicht mit einer Return-Anweisung endet. Das Makro muss sich in einem sichtbaren Skript (siehe Dokumentation [Business-Process-Management](#), Kapitel *Sichtbereich*) befinden.

Syntax

```
Call <Scriptname>.<Macroname>[( <Argument>[,<Argument>,...]]
```

1. Scriptname
Der Name des Skripts. Das Skript kann im aktuellen Bearbeiter oder in einem innerhalb der sichtbaren Struktur übergeordneten Bearbeiter liegen.
2. Macroname
Der Name des Makros in dem Skript.
3. Argument
{Ausdruck|Funktion|Literal|Variable}
Das Argument des Makros.

Beispiel

```
Dim name As String
```

```
Dim alter As Integer
```

```
name = „Josef Springer“
```

```
alter = 10
```

```
kosten = Call Melden.Geburtstag ( name, ( alter + 20 ) )
```

```
MsgBox ( Array("die Party kostet: ", kosten, " Euro" ) )
```

Bemerkung

Das Makro wird innerhalb der aktuellen Bearbeiterhierarchie anhand der Signatur in der Call-Anweisung gesucht. Die Signatur besteht aus dem Skript- und Makroname, sowie der Argumentenreihenfolge und der Argumenttypen. Wenn das Makro nicht gefunden wird, erscheint eine entsprechende Fehlermeldung und die Funktion wird abgebrochen.

Für die Argumente der Call-Funktion wird eine automatische Konvertierung der Datentypen durchgeführt (z.B. Integer 1234.99 nach String "1.234,99"), wenn der Datentyp des Arguments in der Call-Funktion nicht mit dem Datentyp des Arguments im gerufenen Makro übereinstimmt. Falls eine Typenkonvertierung nicht möglich ist (z.B.: Inte-

ger nach Date), erscheint eine entsprechende Fehlermeldung und die Funktion wird abgebrochen.

Call-Supermakro

Die Funktion ruft das Makro eines Skripts des übergeordneten Bearbeiters auf. Das Makro wird dabei nicht im Bearbeiter des laufenden Makros gesucht, sondern das Makro wird im darrüberliegenden Bearbeiter der Bearbeiterhierarchie gesucht. Damit kann ein Makro innerhalb der Bearbeiterhierarchie mehrfach vorhanden sein. Mit den Funktionen `Call` und `Call Super` bestimmen Sie, welches dieser gleich benannten Makro ausgeführt werden soll. Die restliche Beschreibung finde Sie in der Funktion `Call`.

Syntax

```
Call Super <Scriptname>.<Macroname>[( <Argument>[,<Argument>,...]]
```

Beispiel

```
Dim name As String
Dim alter As Integer
name = „Josef Springer“
alter = 10
kosten = Call Super Melden.Geburtstag ( name, ( alter + 20 ) )
MsgBox ( Array("die Party kostet: ", kosten, " Euro" ) )
```

ChrB

Mit der Funktion kann aus einem Zahlenwert ein Zeichen gebildet werden.

Syntax

```
ChrB ( <Zahl> )
```

1. Zahl
{Ausdruck|Funktion|Variable|Literal}
Der Ausdruck liefert den Zahlenwert.

Beispiel

```
Dim zahl As Character
zahl = ChrB( 97 )
MsgBox ( zahl )
```

Chr

Mit der Funktion kann aus einem Zahlenwert eine Zeichenkette gebildet werden.

Syntax

```
Chr ( <Zahl> )
```

1. Zahl
{Ausdruck|Funktion|Variable|Literal}
Der Ausdruck liefert den Zahlenwert.

Beispiel

```
Dim zahl As String
zahl = Chr( 97 )
MsgBox ( zahl )
```

CurDir

Die Funktion liefert den gegenwärtig eingestellten Pfad als String. Bei Windows-Programmen ist dies i.d.R. der Startpfad.

Syntax

```
CurDir
```

Beispiel

```
MsgBox( CurDir )
```

Date

Die Funktion liefert das aktuelle Tagesdatum als Date Variable.

Syntax

```
Date
```

Beispiel

```
Dim datum As Date
datum = Date
MsgBox( datum )
```

Delay

Die Funktion wartet mit der Ausführung der nächsten Anweisung die angegebene Zeitspanne.

Syntax

```
Delay( <Delayzeit> )
```

1. Delayzeit
{Variabel|Literal|Ausdruck|Funktion}
Die zu wartende Zeitspanne in Millisekunden.

Beispiel

```
Shell "notepad" ""
Delay(2000)
MsgBox("Notepad ist gestartet")
```

Bemerkung

Die Funktion ist nützlich, wenn eine andere Anwendung gestartet wird, das Skriptmakro aber sofort weiter ausgeführt werden soll. Das nachfolgende Beispiel erläutert diese Anwendung: Ein Skriptmakro soll den Notizblock starten und mit einer Hinweismeldung weitermachen. Ohne die Delay-Funktion würde die Hinweismeldung sofort ausgegeben, aber

vom Notizblockfenster überdeckt. Mit Delay wartet das Skriptmakro eine halbe Sekunde auf das Öffnen des Notizblockfensters, und gibt dann die Hinweismeldung aus.

Eintragen

```
Shell "notepad" "c:\temp\Hinweis.txt"
```

```
Delay ( 500 )
```

```
MsgBox ( "Tragen Sie jetzt den Kommentar ein" )
```

```
Return True
```

Enum

Mit der Funktion kann eine Enum-Variable gebildet werden. Verschiedene Dienste in COM-Klassen aus Bibliotheken erfordern derartige Variable.

Syntax

```
Enum ( <Enum-Name>, <Enum-Typ> )
```

1. Enum-Name
<Variable>

Ist der Name der Enumeration aus der Bibliothek. Lesen Sie dazu in der Bibliotheksdokumentation nach. i.d.R. entspricht der Name dem Argumentnamen wie er in Typenbibliothek benannt ist.

2. Enum-Typ
<Variable>

Ist der Enum-Typ. Lesen Sie dazu in der Bibliotheksdokumentation nach. i.d.R. entspricht der Typ der Argumentvariante aus gleichnamigen Variante in der Typenbibliothek.

Beispiel

```
Dim WinWord As SmallCOM.Word.Application  
word = New SmallCOM.Word.Application  
word.quit( False, originalFormat: Enum(  
SmallCOM.Word.WdOriginalFormat, wdWordDocument ), routeDocument: False  
)
```

Filter

Mit der Funktion kann einen Array mit Zeichenketten gefiltert werden. Das Ergebnis der Funktion ist das Array mit den gefilterten Zeichenketten.

Syntax

```
Enum ( <Array>, <Filter> [, <Aufnehmen> [, <Vergleichsart> ]] )
```

1. Array
{Ausdruck|Funktion|Literal|Variable}
Ein Array mit zu filternden Zeichenketten.

2. Filter
{Ausdruck|Funktion|Literal|Variable}
Die Zeichenkette, die gefiltert wird.

3. Aufnehmen
<Boolean Variable>
True liefert die Zeichenketten, die <Filter> enthalten.
False liefert die Zeichenketten, die <Filter> nicht enthalten

4. <Vergleichsart>
Ist <Vergleichsart> nicht angegeben ist, wird <Filter> in <Array> mit vbBinaryCompare gesucht.
- | | |
|-----------------|--------------------------------------|
| vbTextCompare | vergleicht ohne Groß-Kleinschreibung |
| vbBinaryCompare | vergleicht mit Groß-Kleinschreibung |

InputBox

Die Funktion öffnet einen Dialog zur Eingabe eines einzeiligen Textes und liefert den eingegebenen Text als String Variable

Syntax

```
InputBox ( <Meldung>[, <Titel>[, <Vorbelegung>] )
```

1. Meldung
{Ausdruck|Funktion|Literal|Variable}
Der Ausdruck gibt den Meldungstext an.
2. Titel
{Ausdruck|Funktion|Literal|Variable}
Der Ausdruck liefert, falls angegeben, die Überschrift des Eingabedialoges
3. Vorbelegung
{Ausdruck|Funktion|Literal|Variable}
Der Ausdruck liefert eine Vorbelegung des Eingabetextes.

Beispiel

```
Dim antwort As String  
antwort = InputBox("Namen eingeben", "Titel der Eingabe", "Josef" )  
MsgBox( antwort )
```

IsDate

Die Funktion meldet als Boolean, ob die angegebene Variable vom Type Date ist.

Syntax

```
IsDate ( <Variable> )
```

1. Variable
{Ausdruck|Funktion|Literal|Variable}
Das Ergebnis des Ausdrucks wird geprüft.

Beispiel

```
Dim text As String  
Dim antwort As Boolean  
text = "Ja"  
antwort = IsDate ( text )  
If antwort = True Then  
... tu was ...
```

IsEmpty

Die Funktion meldet als Boolean, ob die angegebene Variable leer ist.

Syntax

```
IsEmpty ( <Variable> )
```

1. Variable
{Ausdruck|Funktion|Literal|Variable}
Das Ergebnis des Ausdrucks wird geprüft.

Beispiel

```
Dim leer As String
Dim antwort As Boolean
antwort = IsEmpty ( leer )
If antwort = True Then
... tu was ...
```

IsNull

Die Funktion meldet als Boolean, ob der Inhalt der angegebenen Variable Null ist.

Syntax

```
IsNull ( <Variable> )
```

1. Variable
{Ausdruck|Funktion|Literal|Variable}
Das Ergebnis des Ausdrucks wird geprüft.

Beispiel

```
Dim leer As Integer
Dim antwort As Boolean
leer = 0
antwort = IsNull ( leer )
If antwort = True Then
... tu was ...
```

IsNumeric

Die Funktion meldet als Boolean, ob der Inhalt der angegebenen Variablen numerisch ist.

Syntax

```
IsNumeric ( <Variable> )
```

1. Variable
{Ausdruck|Funktion|Literal|Variable}
Das Ergebnis des Ausdrucks wird geprüft.

Beispiel

```
Dim zahl As Integer
Dim antwort As Boolean
```

```
zahl = 1
antwort = IsNumeric ( zahl )
If antwort = True Then
... tu was ...
End If
```

InStr

Die Funktion sucht innerhalb einer Zeichenkette eine angegebene Zeichenkette und liefert als Ergebnis die Suchposition als Integer-Variable. Die Zählbasis ist 1.

Syntax

```
InStr ( <Zeichenkettel>, <Zeichenkette2> [ , <Vergleichsart> ] )
```

1. Zeichenkettel
{Ausdruck|Funktion|Literal|Variable}
Die Zeichenkette, innerhalb der gesucht wird.
2. Zeichenkette2
{Ausdruck|Funktion|Literal|Variable}
Die Zeichenkette, nach der gesucht wird.
3. <Vergleichsart>
Wenn die Vergleichsart nicht angegeben ist, wird vbBinaryCompare angenommen.

vbTextCompare	vergleicht ohne Groß-Kleinschreibung
vbBinaryCompare	vergleicht mit Groß-Kleinschreibung

Ergebnis

0	Zeichenkette2 wurde in Zeichenkettel nicht gefunden
>0	Die Startposition von Zeichenkette2 innerhalb Zeichenkettel
Null	Zeichenkettel oder Zeichenkette2 ist Null

Beispiel

```
Dim text1 As String
Dim text2 As String
Dim vergleich As Integer
text1 = "erster"
text2 = "letzter"
vergleich = InStr (text1, text2 )
MsgBox( vergleich )
```

InStrRev

Die Funktion gibt die Position des ersten Zeichens einer Zeichenfolge innerhalb einer anderen Zeichenfolge vom Ende der Zeichenfolge her gesehen zurück.

Syntax

```
InStrRev ( <Zeichenkette>, <Vergleich> [ , <Vergleichsart> ] )
```

1. <Zeichenkette>
<Ausdruck | Funktion | Literal | Variable>
Die Zeichenkette für den Vergleich.
2. <Vergleich>
<Ausdruck | Funktion | Literal | Variable>
Die Zeichenkette mit der verglichen wird.
3. <Vergleichsart>
Wenn die Vergleichsart nicht angegeben ist, wird vbBinaryCompare angenommen.

vbTextCompare	vergleicht ohne Groß-Kleinschreibung
vbBinaryCompare	vergleicht mit Groß-Kleinschreibung

Join

Die Funktion verbindet die Variablen eines Arrays zu einer einzigen Zeichenkette und liefert diese als Ergebnis.

Syntax

```
Join ( <Array>[,<Trenner>] )
```

1. <Array>
Die Variablen aus Array werden zu einer einzigen Zeichenkette verbunden.
2. <Trenner>
{Ausdruck | Funktion | Literal | Variable}
Falls angegeben, werden die einzelnen Teile aus Array durch die Zeichenkette Trenner getrennt. Falls nicht angegeben, wird ein Leerzeichen als Trenner verwendet.

Hinweis: Abweichend zur VisualBasic-Syntax kann <Trenner> nicht nur ein einzelnes Zeichen, sondern auch eine Zeichenkette sein.

Beispiel

```
Dim zeichen As Array
Dim zeichen As String
zeichen = Array ("h", "a", "l", "l", "o" )
text = Join (zeichen, " ")
MsgBox( text )
```

LCase

Die Funktion wandelt alle Zeichen in einer Zeichenkette in Kleinbuchstaben und liefert die gewandelte Zeichenkette.

Syntax

```
LCase ( <Zeichenkette> )
```

1. Zeichenkette
{Variabel | Literal | Ausdruck | Funktion}
Die Zeichenkette, in der alle Großbuchstaben in Kleinbuchstaben gewandelt werden.

Left

Die Funktion extrahiert aus einer Zeichenkette eine linke Teilzeichenkette.

Syntax

```
Left ( <Zeichenkette> , <Länge> )
```

1. Zeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Zeichenkette, aus der ein linker Teil extrahiert wird.
2. Länge
{Variabel|Literal|Ausdruck|Funktion}
Die Anzahl der zu extrahierenden Zeichen.

Beispiel

```
MsgBox ( "Left('Aber das ist toll', 4 ) : ", & Left("Aber das ist  
toll", 4 ) )
```

Len

Die Funktion meldet die Länge einer Zeichenkette als Integer.

Syntax

```
Len ( <Variable> )
```

1. <Variable>
{Ausdruck|Funktion|Literal|Variable}
Die Variable muß eine Zeichenkette enthalten.

Beispiel

```
Dim name As String  
Dim länge as Integer  
name = "Axel Bauer"  
länge = Len( name )  
MsgBox ( "Die Länge meines Namens beträgt: " & länge )
```

LTrim

Die Funktion löscht alle Leerzeichen am Anfang einer Zeichenkette und liefert die Zeichenkette.

Syntax

```
LTrim ( <Zeichenkette> )
```

1. Zeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Zeichenkette, in der alle linken Leerzeichen gelöscht werden.

Mid

Die Funktion extrahiert aus einer Zeichenkette eine Teilzeichenkette.

Syntax

```
Mid ( <Zeichenkette> ,<Start>[, <Länge>] )
```

1. Zeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Zeichenkette, aus der ein Teil extrahiert wird.
2. Start
{Variabel|Literal|Ausdruck|Funktion}
Die Position des ersten zu extrahierenden Zeichens. Die Zählbasis ist 1.
3. Länge
{Variabel|Literal|Ausdruck|Funktion}
Die Anzahl der zu extrahierenden Zeichen. Wenn das Argument fehlt, werden ab Start alle Zeichen extrahiert.

Beispiel

```
MsgBox ("Mid('Aber das ist toll', 6, 3 ) : ", & Mid("Aber das ist  
toll", 6, 3 ) )
```

MousePointer

Mit der Funktion können Sie den aktuellen Mauszeiger bestimmen. Sie können damit z.b. während lang andauernder Datenbankoperationen die „Eieruhr“ zeigen. Die Funktion antwortet mit dem Namen des zuletzt eingestellten Mauszeigers.

Syntax

```
MousePointer ( <Mauszeiger> | <Variable> )
```

1. Mit <Mauszeiger> oder <Variable> bestimmen Sie den Mauszeiger. Das ist ein vorgegebener Name oder eine Variable vom Typ String mit einem vorgegebenen Namen. Die Namen der Mauszeiger lauten:

vbHourglass	Eieruhr
vbNoDrop	durchgestrichener Kreis
vbDefault oder vbArrow	standard Mauszeiger (Pfeil)
vbCrosshair	zwei gekreuzte Balken
vbUpArrow	nach oben gerichteter Pfeil

Beispiel

```
Library "..\library\OracleForOfficeTalk.pcl"
Dim previousPointer As String
Dim befehl As SQLString
Dim session As SmallCOM.OracleForOfficeTalk.OfficeTalkSession
befehl = "SELECT KD_VORGANGS_NR, AUFTRART, AUFTRAGSKENNWORT,  
WUNSCH_LIEFERTERMIN, LT_LIEFER_DATUM, AU_DISPONENT from PKVK where  
KD_VORGANGS_NR = '1000' and KUNDEN_NR = '2000'"
previousPointer = MousePointer( vbHourglass )
session = Call Oracle.Open
```

```

session.createDynaset( befehl, queryOption: 0 )
If session.eof = False
    Then
        ... Ergebnisse abholen ...
    End If
session.closeSession
MousePointer( previousPointer )

```

MsgBox

Die Funktion öffnet einen Meldungsdialog und antwortet mit einem Integerwert, entsprechend dem geklickten Button. Dabei wird z.b. 1 für den linken Button und 3 für den dritten Button von links gemeldet. Der Argumentliste kann mit und ohne umschließende Klammern geschrieben werden.

Syntax

```

MsgBox { ( <Meldung> [, <Button>[, <Icon>[, <Titel>]]] ) |
        <Meldung> [, <Button>[, <Icon>[, <Titel>]]] }

```

1. Meldung

<Variabel|Literal|Ausdruck|Funktion>

Der anzuzeigende Text. Die Zeichenkette "<n>" trennt den nachfolgenden Text in die nächste Zeile. Die Zeichenkette "<t>" setzt einen Tabulator. Die Zeichen < und >, sollen sie auch angezeigt werden, müssen durch ein vorangestelltes % gequotet werden, ansonsten werden in der Meldung die Sonderzeichenketten "<t>" und "<n>" nicht korrekt interpretiert und direkt angezeigt.

2. Button

Zum Definieren der Buttons stehen Ihnen drei Varianten zur Verfügung:

<Buttonname> oder

(<Buttonname> , <Buttonnummer>) oder

(<Buttonnummer>)

<Buttonname>

Bezeichnet den Button mit seiner Beschriftung. Dafür schreiben Sie:

vbOKCancel **OK und Abbruch**

vbAbortRetryIgnore **Abbruch, Wiederholen und Ignorieren**

vbYesNoCancel **Ja, Nein und Abbruch**

vbYesNo **Ja, und Nein**

vbRetryCancel **Wiederholen und Abbruch**

<Buttonnummer>

Bezeichnet die Nummer des gedrückten Buttons der ohne die Meldungsausgabe retourniert wird.

Wenn OfficeTalk in Benutzerlevel **Beginner** oder höher arbeitet, wird ohne oder mit dem Icon vbInformation die Meldung nicht ausgegeben, sondern die angegebene Buttonnummer als Antwort geliefert.

Wenn OfficeTalk in Benutzerlevel **Normal** oder **Expert** arbeitet, wird mit dem Icon vbQuestion die Meldung nicht ausgegeben, sondern die angegebene Buttonnummer als Antwort geliefert.

Wenn OfficeTalk in Benutzerlevel **Expert** arbeitet, wird mit dem Icon vbExclamation die Meldung nicht ausgegeben, sondern die angegebene Buttonnummer als Antwort geliefert

Meldungen mit dem Icon `vbCritical` werden immer ausgegeben, ohne Rücksicht auf den Benutzerlevel und die angegebene `Buttonnummer`.

3. Icon

<code>vbCritical</code>	Halt Icon; Meldungen mit diesem Icon werden immer ausgegeben.
<code>vbQuestion</code>	Fragezeichen; Meldungen mit diesem Icon werden in Abhängigkeit der <code><Buttonnummer></code> und des Benutzerlevels ausgegeben.
<code>vbExclamation</code>	Ausrufezeichen; Meldungen mit diesem Icon werden in Abhängigkeit der <code><Buttonnummer></code> und des Benutzerlevels ausgegeben.
<code>vbInformation</code>	Infozeichen; Meldungen mit diesem Icon werden in Abhängigkeit der <code><Buttonnummer></code> und des Benutzerlevels ausgegeben

4. Titel

{Ausdruck|Funktion|Variable|Literal}
Der Titel der Meldungsbox

Beispiel

```
Dim antwort As Integer

antwort = MsgBox ("Hilfe<n>Gebt mir Geld", vbYesNo, vbQuestion, " Hil-
feansuchen" )

If antwort = 1 Then
... tu was ...
End If
```

Refresh

Die Funktion aktualisiert den Inhalt aller geöffneten Dialoge. Bei ActiveX Komponenten die mit einem Dienst einen Dialog öffnen und wieder schließen, kann es vorkommen, dass überdeckte Bereiche in OfficeTalk Dialogen beim Schließen nicht aktualisiert werden. Diese Bereiche bleiben grau. Um das zu verhindern, verwenden Sie nach dem verursachenden Dienst die Funktion `Refresh`.

Syntax

```
Refresh
```

Beispiel

```
Library "..\Library\MyApplication.pcl"

Dim prog As SmallCOM.MyProg.Application

prog = New SmallCOM.MyProg.Application
prog.auswahlAusListe
Refresh
```

Replace

Die Funktion ersetzt aus einer Zeichenkette eine Teilzeichenkette nach Vorgabe und liefert als Ergebnis die Zeichenkette mit den Ersetzungen.

Syntax

```
Replace ( <Zeichenkette> , <Teilzeichenkette> , <Ersetzen> [, <Start>  
[, <Anzahl> [, <Vergleichsart> ] ] ] )
```

1. Zeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Zeichenkette, in der die Ersetzung durchgeführt wird.
2. Teilzeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Teilzeichenkette, die ersetzt wird.
3. Ersetzen
{Variabel|Literal|Ausdruck|Funktion}
Die Teilzeichenkette, mit der alle <Teilzeichenkette> ersetzt werden.
4. Start
{Variabel|Literal|Ausdruck|Funktion}
Die Textposition ab der die Ersetzung stattfinden. Die Zählbasis ist 1.
5. Anzahl
{Variabel|Literal|Ausdruck|Funktion}
Die maximale Anzahl der Ersetzungen.
6. <Vergleichsart>
Ist <Vergleichsart> nicht angegeben ist, wird <Teilzeichenkette> in
<Zeichenkette> mit vbBinaryCompare gesucht.

vbTextCompare	vergleicht ohne Groß-Kleinschreibung
vbBinaryCompare	vergleicht mit Groß-Kleinschreibung

Hinweis: Die Ersetzungen finden nicht <Zeichenkette> in statt, sondern das Ergebnis der Funktion ist die Zeichenkette mit den Ersetzungen.

Right

Die Funktion extrahiert aus einer Zeichenkette eine rechte Teilzeichenkette.

Syntax

```
Right ( <Zeichenkette> , <Länge> )
```

1. Zeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Zeichenkette, aus der ein rechter Teil extrahiert wird.
2. Länge
{Variabel|Literal|Ausdruck|Funktion}
Die Länge der von rechts zu extrahierenden Zeichenkette.

Beispiel

```
MsgBox ( "Right('Aber das ist toll', 4 ) : ", & Right ( "Aber das ist  
toll", 4 ) )
```

RTrim

Die Funktion löscht alle Leerzeichen am Ende einer Zeichenkette und liefert die Zeichenkette.

Syntax

```
RTrim ( <Zeichenkette> )
```

1. Zeichenkette

```
{Variabel|Literal|Ausdruck|Funktion}
```

Die Zeichenkette, in der alle rechten Leerzeichen gelöscht werden.

Shell

Die Funktion startet ein Programm mit Argumenten. Ist beim Start des Programms ein Fehler aufgetreten, wird eine Ausnahme geworfen, wenn das Programm innerhalb von einer vorgegebenen Zeitspanne (ca.10 Millisekunden) gestartet werden konnte. Die Ausnahme mündet in einer entsprechenden Fehlermeldung. Mit der *Try-Catch*-Anweisung kann die Ausnahme programmatisch bearbeitet werden. Die Bearbeitung des Makros wird nach dem Start des Programms sofort fortgesetzt. Das Ergebnis der Funktion ist in jedem Fall `Null`.

Hinweis: Auf Windows-Plattformen wird das angegebene Programm mit Hilfe des Systemprogramms **CMD.EXE** ausgeführt.

Für das Argument `Programm` kann auch der Dienst *applicationFor*(<Datei>) verwendet werden.

Da die Funktion bei einem fehlerhaften Programmstart sowohl `Null` liefern, als auch eine Ausnahme werden kann, ist in jedem Fall die Verwendung einer *Try-Catch*-Anweisung zu empfehlen.

Syntax

```
Shell [Encoding <Zeichensatz>,< >] <Programm> <Argument>
```

1. Falls erforderlich, wird mit `Encoding` der Zeichensatz des Programmresultates festgelegt. Falls der Zeichensatz des Resultates **OEM** ist, muss für <Zeichensatz> **OEM** geschrieben werden. **Unicode** ist der Standard. Der Parameter ist nur auf Windows-Plattformen wirksam.
2. <Programm>

```
<Ausdruck|Funktion|Literal|Variable>
```

 Die Variable enthält das zu startende Programm.
3. <Argument>

```
<Ausdruck|Funktion|Literal|Variable>
```

 Der Inhalt von <Argument> wird dem Programm als Startparameter übergeben. Sind keine Startparameter erforderlich, muss ein leerer String ("") verwendet werden.

Hinweis: Wenn die Funktion nicht innerhalb einer *Try-Catch*-Anweisung verwendet wird, kann die Angabe von <Argument> entfallen.

Auf Windows-Plattformen wird das Argument <Programm>, wenn es Leerzeichen enthält, in doppelte Anführungszeichen gesetzt. Soll dies verhindert werden, muss das Argument mit einem Anführungszeichen beginnen.

Auf Windows-Plattformen können einige Kommandos nur mit dem Kommando `Start` ausgeführt werden. Ein Beispiel dafür ist das Kommando `mailto:`, um mit Hilfe des installierten Mailklienten eine E-Mail zu senden.

Z.B.: `Shell "" "Start mailto:hMeier@rechner.de" "" ""`

Beispiel

```
Dim prog As String
```

```
Dim fehler As Array
```

```
prog = "notepad"
Try
    Shell prog "c:\temp\josef.txt"
End Try
Catch
    fehler = Error.description
    MsgBox ("Fehlercode: " & fehler(0) & ": " & fehler(1))
End Catch
```

Bemerkung

Wenn <Programm> leer ist, und <Argument> einen Dateinamen enthält, dessen Namens-erweiterung auf Betriebssystemebene (auf Linux-Plattformen in OfficeTalk) einem Program-m zugeordnet ist, wird dieses Programm mit der Datei aus <Argument> gestartet.

Sollen dem Programm mehrere Startparameter übergeben werden, muss dies in Form ei-nes Array erfolgen. Die einzelnen Argumente werden dabei für den Aufruf jeweils in " " ge-setzt. Die Anweisung

```
Shell "MeinProgramm.exe" Array(1, "Ausgabe", 200)
```

wird also aufbereitet zu

```
"MeinProgramm.exe" "1" "Ausgabe" "200"
```

Shell Wait

Die Funktion startet ein Programm mit Argumenten. Die Bearbeitung des Makros wird erst nach Beendigung des gestarteten Programms fortgesetzt. Das Ergebnis der Funktion ist das Ergebnis aus stdout. Ist bei der Programmausführung ein Fehler aufgetreten, wirft die Funktion eine Ausnahme, die in einer entsprechenden Fehlermeldung mündet. Mit der *Try-Catch*-Anweisung kann die Ausnahme programmatisch bearbeitet werden.

Hinweis: Auf Windows-Plattformen wird das angegebene Programm mit Hilfe des System-programms **CMD.EXE** ausgeführt.

Syntax

```
Shell Wait [Encoding <Zeichensatz>,<,>] <Programm> <Argument>
```

1. Falls erforderlich, wird mit `Encoding` der Zeichensatz des Programmergebnisses fest-gelegt. Falls der Zeichensatz des Ergebnisses **OEM** ist, muss für <Zeichensatz> **OEM** geschrieben werden. **Unicode** ist der Standard. Der Parameter ist nur auf Windows-Plattformen wirksam.
2. <Zeichensatz>
Beschreibung siehe Kapitel *Shell*.
3. <Argument>
<Ausdruck|Funktion|Literal|Variable>
Der Inhalt von <Argument> wird dem Programm als Startparameter übergeben. Siehe auch Beschreibung im Abschnitt *Bemerkung*.

Beispiel

```
Dim fehler As Array
Try
```

```

    Shell Wait "" "Dokument.pdf"          'Adobe-Reader wird gestartet
End Try
Catch
    fehler = Error.description
    MsgBox ("Fehlercode " & fehler(0) & ": " & fehler(1))
End Catch

```

Space

Die Funktion eine Zeichenkette mit der angegebenen Anzahl von Leerzeichen.

Syntax

```
Space (<Anzahl> )
```

1. Anzahl
{Variabel|Literal}
Die Anzahl der Leerzeichen.

Split

Die Funktion trennt eine Zeichenkette in Teilzeichenketten auf, und liefert diese in einem Array. Die Zeichenkette wird an dem angegebenen Trennzeichen aufgeteilt.

Syntax

```
Split ( <Variable>[,<Trenner>[,<Vergleichsart>]] )
```

1. <Variable>
Die Variablen mit der Zeichenkette.
2. <Trenner>
{Ausdruck|Funktion|Literal|Variable}
Falls angegeben, wird die Zeichenkette an dem Trennzeichen getrennt. Falls nicht angegeben, wird das Leerzeichen als Trenner verwendet.
3. <Vergleichsart>
Ist <Vergleichsart> nicht angegeben ist, wird <Trenner> in <Variable> mit vbBinaryCompare gesucht.

vbTextCompare	vergleicht ohne Groß-Kleinschreibung
vbBinaryCompare	vergleicht mit Groß-Kleinschreibung

Hinweis: Das Ergebnis enthält die aufgetrennten Zeichenketten ohne <Trenner>.

Start

Die Funktion startet die benannte Vorgangsvorlage. Das Ergebnis der Funktion ist der gestartete Vorgang. Damit die Vorgangsvorlage gestartet werden kann, muss sie sich in einem sichtbaren Bearbeiter befinden, und der angemeldete Bearbeiter muss für die Vorgangsart (Name und Kategorie) das Vorgangsrecht **Starten** besitzen (siehe Kapitel *Vorgangsrechte* in der Dokumentation *Business-Process-Management*). Der Vorgang wird in der Aufgabenliste des Bearbeiter eingetragen, der als **ausführender Bearbeiter** im ersten Arbeitsschritt eingetragen ist. Wenn diesem Bearbeiter die erforderlichen Rechte zum Ausführen des Vorgangs fehlen, wird der Vorgang zwar in dessen Aufgabenliste eingetragen, er kann aber den Vorgang später nicht ausführen. Sein Administrator (ein übergeordneter Bearbeiter) muss zuerst die erforderlichen Rechte (mindestens **ausführen** oder **beenden**) für den Vorgang in diesem Bearbeiter eintragen. Als Ergebnis wird, falls erfolgreich, der ge-

startete Vorgang geliefert. Diesem können, wie im *Beispiel* gezeigt, u.a. Vorgangsdaten mitgegeben werden.

Konnte der Vorgang aus irgend einem Grund nicht gestartet werden, wird eine Ausnahme geworfen (siehe Kapitel *Behandlung fehlerhafter Argumente*).

Hinweis: Wenn der Vorgang in die persönliche Aufgabenliste des angemeldeten Bearbeiters eingetragen wurde, und das Startdatum des ersten Arbeitsschrittes erreicht ist, wird dessen Ausführung nach dem Ende des aktuellen Arbeitsschrittes gestartet.

Syntax

```
Start "<Kategorie>" "<Name>" [,Automatic]
```

1. Kategorie
{Ausdruck|Funktion|Literal|Variable}
Benennt die Kategorie der Vorgangsvorlage.
2. Name
{Ausdruck|Funktion|Literal|Variable}
Benennt den Namen der Vorgangsvorlage.
3. Mit dem optionalen Argument `Automatic` wird der gestartete Vorgang, ähnlich wie mit dem Kontextmenü **Automatisch ausführen**, automatisch ausgeführt. Einzelheiten zum automatischen Ausführen eines Vorgangs finden Sie im Kapitel *Automatisch Ausführen der Dokumentation Workflow*.

Beispiel

```
Dim gestartet As Process
Try
    gestartet = Start "Vertrieb" "Schulung"
    MsgBox("Der Vorgang wurde gestartet")
    gestartet.processdata.item ( "UMS", with: 200.20,
                                in: "Umsatz" )
End Try
Catch
    MsgBox("Der Vorgang konnte nicht gestartet werden")
End Catch
```

Bemerkung

Nach dem Abschluss des aktuellen Arbeitsschrittes wird der mit `Start` gestartete Vorgang sofort ausgeführt, wenn die dazu erforderlichen Bedingungen erfüllt sind. Siehe dazu Dokumentation *Workflow*, Kapitel *Automatischer Start der Ausführung*.

StrComp

Die Funktion vergleicht zwei Zeichenketten miteinander und liefert das Ergebnis des Vergleiches als Integer Variable

Syntax

```
StrComp ( <Zeichenkettel>, <Zeichenkette2> [ , <Vergleichsart>] )
```

1. <Zeichenkettel>
<Ausdruck|Funktion|Literal|Variable>
Die Zeichenkette für den Vergleich.
2. <Zeichenkette2>
<Ausdruck|Funktion|Literal|Variable>
Die zweite Zeichenkette für den Vergleich.
3. <Vergleichsart>
Wenn die Vergleichsart nicht angegeben ist, wird vbBinaryCompare angenommen.

vbTextCompare	vergleicht ohne Groß-Kleinschreibung
vbBinaryCompare	vergleicht mit Groß-Kleinschreibung

Ergebnis

-1	Zeichenkettel ist kleiner als Zeichenkette2
1	Zeichenkettel ist größer als Zeichenkette2
0	Zeichenkettel und Zeichenkette2 sind gleichwertig

Beispiel

```
Dim text1 As String
Dim text2 As String
Dim vergleich As Integer
text1 = "erster"
text2 = "letzter"
vergleich = StrComp (text1, text2, vbTextCompare )
MsgBox( text1 & " und " & text2 & " sind gleich: " & vergleich )
```

StrConv

Die Funktion wandelt eine Zeichenfolge in die angegebene Schreibweise um und liefert diese Zeichenkette.

Syntax

```
StrConv ( <Zeichenkette>, <Schreibweise> )
```

1. <Zeichenkette>
<Ausdruck|Funktion|Literal|Variable>
Die Zeichenkette für den Vergleich.
2. < Schreibweise >

vbUpperCase	Die Zeichenkette wird in Großbuchstaben umgewandelt.
-------------	--

<code>vbLowerCase</code>	Die Zeichenkette wird in Großbuchstaben umgewandelt.
<code>vbProperCase</code>	Jeweils der erste Buchstabe eines jeden Wortes der Zeichenkette wird in einen Großbuchstaben umgewandelt
<code>vbUnicode</code>	Die Zeichenkette wird in Unicode umgewandelt.
<code>vbFromUnicode</code>	Die Zeichenkette wird vom Unicode in die Voreinstellungen des Systems umgewandelt.

String

Die Funktion liefert eine Zeichenkette mit der angegebenen Anzahl von Zeichen.

Syntax

```
Len ( <Anzahl>, <Zeichen> )
```

1. Anzahl
{Variabel|Literal|Ausdruck|Funktion}
Die Anzahl der Zeichen.
2. Zeichen
{Variabel|Literal|Ausdruck|Funktion}
Das Zeichen aus dem die Zeichenkette erstellt wird.

StrReverse

Die Funktion vertauscht die Reihenfolge der Zeichen in einer Zeichenkette und liefert diese.

Syntax

```
StrReverse ( <Zeichenkette> )
```

1. Zeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Zeichenkette.

Time

Die Funktion liefert die aktuelle Uhrzeit als Time Variable

Syntax

```
Time
```

Beispiel

```
Dim zeit As Time  
zeit = Time  
MsgBox( zeit )
```

UBound

Die Funktion liefert die Anzahl der Elemente in einem Array.

Syntax

```
UBound ( <Variable> )
```

1. <Variable>
{Ausdruck|Funktion|Literal|Variable}
Der Ausdruck muss ein Array liefern.

Beispiel

```
Dim elemente As Array  
elemente = Array(100,200,300)  
MsgBox ("Die Anzahl der Elemente ist: " & UBound(elemente) )
```

Bemerkung

Wenn <Variable> kein Array enthält, liefert die Funktion Null als Ergebnis.

UCase

Die Funktion wandelt alle Zeichen in einer Zeichenkette in Großbuchstaben und liefert die gewandelte Zeichenkette.

Syntax

```
UCase ( <Zeichenkette> )
```

1. Zeichenkette
{Variabel|Literal|Ausdruck|Funktion}
Die Zeichenkette, in der alle Kleinbuchstaben in Großbuchstaben gewandelt werden.

Die Makrosyntax im Überblick

Die Syntax entspricht weitgehend der Microsoft®VisualBasic Syntax. Lesen Sie dazu auch die Dokumentation für VisualBasic.

Ausdrücke

Kaskadierung

Mehrere Ausdrücke können nacheinander geschrieben werden. Entsprechend der Operatorbindung und der Reihenfolgeregelung wird das Ergebnis gebildet. Ein Beispiel:

```
Dim zeile As String
```

```
Dim zahl As Integer
```

```
zeile = "Das " & "ist " & "der "beste " & "Weg"
```

```
MsgBox( zeile ) "Das ist der beste Weg"
```

```
zahl = 2 * 5 - 15 + 33
```

```
MsgBox( zahl ) -86
```

Bearbeitungsreihenfolge

Ausdrücke, soweit keine Bearbeitungsreihenfolge durch Klammern gegeben ist, werden von rechts nach links abgearbeitet. Durch entsprechendes Setzen von Klammern können Sie die Bearbeitungsreihenfolge, und damit das Ergebnis des Ausdrucks beeinflussen. Ein Beispiel:

```
Dim zahl As Integer
```

```
Dim ergebnis As Long
```

```
zahl = 5 - 10 + 20 + 10
```

```
MsgBox( "Ergebnis: " & zahl ) Ergebnis: -35
```

```
zahl = ( 5 - 10 ) + ( 20 + 10 )
```

```
MsgBox( "Ergebnis: " & zahl ) Ergebnis: 25
```

```
ergebnis = ((5 + 10) - (6 + 50)) + (11 - 3)
```

```
MsgBox("Das Ergebnis: " & ergebnis) Das Ergebnis: 49
```

Binäre Ausdrücke

Boolsche Operatoren

Operator Is

Der Operator vergleicht den Inhalt zweier Variablen und liefert `TRUE`, wenn beide Variablen das selbe Objekt enthalten oder `FALSE`, wenn die Variablen unterschiedliche Objekte enthalten. Ein kleines Beispiel soll das verdeutlichen:

```
Dim var1 As String
```

```
Dim var2 As String
```

```
var1 = "der Text"
```

```
var2 = var1
```

```
MsgBox("var1 Is var2: " & var1 Is var2) 'liefert: "var1 Is var2: True"
```

oder

```
Dim var1 As String
```

```
Dim var2 As String
```

```
var1 = "der Text"
```

```
var2 = "der Text"
```

```
MsgBox("var1 Is var2: " & var1 Is var2) 'liefert: "var1 Is var2:  
False"
```

Arithmetische Operatoren

Bei Ausdrücken mit arithmetischen Operatoren müssen alle Operanden selbst wieder arithmetisch Ausdrücke sein. Der linke Operand muss ein Literal oder eine Variable sein. Der rechte Operand kann ein Literal, eine Variable oder auch ein Ausdruck sein. Die erste Zuweisung ist korrekt wogegen die zweite Zuweisung beim Kompilieren einen Syntaxfehler ergibt:

Beispiel

```
Dim zahl As Integer
```

```
Dim ergebnis As Integer
```

```
zahl = 100
```

```
ergebnis = zahl + processdata.item("alter")
```

ist richtig

```
ergebnis = processdata.item("alter") + zahl
```

ergibt Syntaxfehler

Operator &

Linker Operand ist Array

Wenn der rechte Operand ein Array ist, liefert der Operator ein Array mit dem Inhalt der Arrays des linken und rechten-Operanden (`Array(Elemente linker Operand, Elemente rechter Operand)`). Wenn der rechte Operand kein Array ist, liefert der Operator ein Array mit den Elementen des Array des linken Operanden, ergänzt um den rechten Operanden.

Linker Operand ist kein Array

Der linke Operand muss ein Literal, eine Variable oder ein Ausdruck in Klammern sein. Der rechte Operand kann auch ein Ausdruck ohne Klammern sein. Falls einer der beiden Operanden keinen String liefert, wird dessen Ergebnis automatisch in einen String gewandelt. Die erste und dritte Zuweisung ist korrekt, wogegen die zweite Zuweisung beim Kompilieren einen Syntaxfehler ergibt.

Beispiel

```
Dim name As String
Dim A As Array
Dim B As Array
Dim C As Array
Dim result As Array
Dim dialog As Joops.Scripting.ScriptDialog
.....
name = "Meier" & dialog.valueNamed("vorname")           ist richtig
name = dialog.valueNamed("nachname") & "Herbert"       ergibt Syntaxfehler
name = (dialog.valueNamed("nachname")) & "Herbert"     ist richtig
A = Array("AA", "BB", "CC")
B = Array("DD", "EE", "FF")
C = Array(Array("D", "E", "F"), Array("G", "H", "I"))
result = A & B                                           ("AA", "BB", "CC", "DD", "EE", "FF")
result = A & B(0)                                       ("AA", "BB", "CC", "DD")
result = A & C                                           ("AA", "BB", "CC", "DD", Array("D", "E", "F"),
                                                         Array("G", "H", "I"))
```

Verbinden mehrerer Zeichenketten

Wenn Sie mehrere Zeichenketten zu einer Einzigigen verbinden möchten, können Sie neben dem Operator & auch die Funktion Array verwenden.

Beispiel

```
Dim zeile As String
zeile = Array("Das ", "ist der ", "schnellste ", "Weg")
zeile = "das " & "ist " & "der " & "schwierige " & "Weg"
```

Hinweise für die Verwendung von Variablen

Der Speicher einer Variable wird beim Ende des Makros von der automatischen Speicher-verwaltung freigegeben. Als Ausnahme von dieser Regel gelten die globalen Variablen, Argumentevariablen des Makros und die Variable der Return-Anweisung.

Wird ein Dialogelement mit dem Inhalt einer Variablen versorgt und die Dialogvariable selbst ist über den Makroablauf hinaus gültig (Dialogvariable ist Argument des Makros), darf der Speicher der Variablen am Ende des Makros von der automatischen Speicher-verwaltung nicht freigegeben werden! Für diese Variable wird die Speicherfreigabe verhindert, indem sie explizit mit Null belegt, oder in der Anweisung Return verwendet wird.

Die Syntax des Makros

Nachfolgend die vollständige Syntax der Makros.

<Makroname [(<Argument[,...]>)]

[<Direktive>]

[<Variable-Deklaration>]

<Anweisung>

<Argument>

<Name> As <Typ>

<Name> Name des Arguments (der Variablen, max. 128 Zeichen lang)

<Typ> Datentyp des Arguments oder der Variablen

{String|Date|Time|Array|ByteString|Character|Integer|Long|
Float|Double|Boolean|Worker|Desk|Machine|Department|Team|
Company|Office|Process|Processdata|Step|Action|
ScriptDialog|Object}

<Direktive>

Library <Literal>

<Variable-Deklaration>

{Public Dim|Dim} <Name> As <Typ>

<Name> Name der Variable (max. 128 Zeichen lang)

<Typ> Datentyp der Variable

{String|Date|Time|Array|ByteString|Character|Integer|Long|
Float|Double|Boolean|Worker|Desk|Machine|Department|Team|
Company|Office|Process|Processdata|Step|Action|
ScriptDialog|Object}

<Anweisung>

{Funktion|Call Dienst|New|Return|If|Until|While|For Next|
Call Script|ExitDo|ExitFor|Print|Write|LineInput|Input|Open|Close|
Try-Catch|Halt}

Die Syntax des Ausdrucks

An vielen Stelle, an denen eine Variable eingesetzt wird, kann auch ein Ausdruck oder eine Funktion, mit dem erforderlichen Ergebnistyp, eingesetzt werden. Nachfolgend die vollständige Syntax der Ausdrücke und Funktionen.

<Ausdruck>

{Binärer Ausdruck|Unärer Ausdruck}

Binärer Ausdruck: {num. Ausdruck|bool Ausdruck|text Ausdruck}

num. Ausdruck: {Date|Time|num.Term|num.Operator|num.Term}

num. Term: {num. Ausdruck|num.Variable}

num. Operator: {+ | - | * | / | \ | mod | ^}

bool Ausdruck: <Term> <bool. Operator> <Term>

Term: {Ausdruck | Funktion | Literal | Variable}

bool. Operator: {= | < > | < | > | <= | >= | And | Or | Xor | Is}

text Ausdruck: <Term> <Operator> <Term>

Term: {Ausdruck | Funktion | Literal | Variable}

<Term> **muß eine Zeichenkette liefern !**

Operator: {&}

Unärer Ausdruck: {<Call Dienst Anweisung | Unärer bool Ausdruck>}

Unärer bool Ausdruck: Not <Term>

<Funktion>

{Numerische Funktion | Bool Funktion | Zeichenkette Funktion |
Zeichen Funktion | Spezial Funktion | Enum Funktion | Shell Funktion |
Start Funktion}

Numerische Funktion: <MsgBox | StrComp | UBound>

Bool Funktion: {IsEmpty | IsNull | IsNumeric | IsDate}

Zeichenkette Funktion: {Join | CurDir | InputBox}

Zeichen Funktion: {ChrB | Chr}

Spezial Funktion: {Array | <Arrayvariable> (<Position>) | Date | Time}

Enum Funktion: <SmallCOM. <Bibliothek>. <Enum>>

Shell Funktion: <Shell>

Start Funktion: <Start>

<Literal>

<Numerisches-Literal | Zeichenketten-Literal>

<Numerisches-Literal>: {Ziffer[Ziffer...] | . }

Der Punkt wird als Dezimaltrenner verwendet !

<Ziffer> {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

<Zeichenketten-Literal>: "Zeichen[Zeichen...]"

<Variable>

<Variablenname>