



local

Deklaration eines lokalen Datenbereichs

Verwandte

Siehe Befehle,

global,

main, sub

Die Definition von Variablen ist unter global beschrieben.

Prozedurlokale Variable

Pro Prozedur kann nur ein lokaler Datenbereich deklariert werden. Die Gültigkeit der Variablen beschränkt sich dabei auf die Funktionen der aktuellen Prozedur.

Lebensdauer

Der prozedurlokale Datenbereich wird in dem Moment angelegt, in dem eine Funktion der Prozedur aufgerufen wird. Er bleibt solange erhalten, bis die Prozedur wieder verlassen wird.

Prozedur A

```
local{ tTest : int;}sub x{ B:a();}sub y{ x();}sub z{ }
```

sub a{ A:z();}
Wird die Funktion 'y' in der Prozedur 'A' aufgerufen, wird der prozedurlokale Datenbereich angelegt. Jetzt ruft 'A:y' zunächst 'A:x' auf, dort wird 'B:a' aufgerufen, die wiederum 'A:z' aufruft. Dabei bleibt der am Anfang angelegte Datenbereich erhalten. Er wird erst in dem Moment wieder entfernt, wenn keine Funktion von 'A' mehr aktiv ist, in unserem Beispiel also nach dem Beenden von 'A:y'. Ein prozedurlokaler Datenbereich ist daher immer nur einmal vorhanden.

Prozedur B

Funktionslokale Variable

Bei einer Funktion kann zusätzlich ein lokaler Datenbereich angelegt werden (sowohl bei sub als auch bei main).

Beispiel:

```
sub ButtonCtrl : int;local{ tCounter : int; tSize : int;}{...}
```

Die Deklaration entspricht dabei der Deklaration prozedurlokaler Variablen. Die Gültigkeit der Variablen beschränkt sich allerdings nur auf die Funktion, in der die Deklaration steht.

Nach der Deklaration sind die Variablen mit dem Wert 0 bzw. einer Entsprechung initialisiert, d. h. alle numerischen Variablen haben den Wert 0, alphanumerische Variablen den Wert "", Zeit-Variablen 00:00:00.0 und Datumsvariablen 0.0.0.

In einem Datenbereich können maximal 4 MB zur Deklaration von Variablen verwendet werden. Wird dieser Wert überschritten, erfolgt die Fehlermeldung Datenlimit überschritten. Die Fehlermeldung kommt auch dann, wenn die Startposition einer Variable nicht innerhalb der ersten 2 MB des Bereichs liegt. Möglicherweise müssen große Arrays dynamisch angelegt werden.

NULL

NULL-Wert

Siehe ComCall()

Variablen und Datenbankfelder in CONZEPT 16 können mit dieser Konstante geleert werden:

Datentyp Wert

<u>alpha</u>	"
<u>byte</u>	0
<u>word</u>	
<u>int</u>	
<u>bigint</u>	
<u>float</u>	0.0
<u>decimal</u>	
<u>decimal</u>	_DecimalUndef
<u>date</u>	0.0.0
<u>time</u>	0:0:0.0
<u>logic</u>	false

Zusammengesetzte Datentypen können ebenfalls auf diese Weise mit einer Anweisung geleert werden. Durch das Leeren wird bei den Datentypen date und decimal der Wert auf "nicht definiert" gesetzt. Bei den anderen Datentypen steht ein gültiger Wert in der Variablen.

Der Bezeichner eines Arrays kann nicht auf den Wert NULL gesetzt werden.

Beispiele:

```
local{ tHdlFrame      : int; tZeichensatz : font; tRechteck      : rect; tFeld          : int[20];}
```



Beim Aufruf von Methoden eines COM-Objekts, die optionale Argumente erwarten, kann mit dem Schlüsselwort NULL ein Argument übersprungen werden.

Konstanten für Feld- und Variablentypen

Konstanten für Feld- und Variablentypen

Siehe Befehlsgruppen

Bei allen Funktionen, die mit Feld- oder Variablentypen arbeiten, sind folgende Typcodes definiert:

- TypeAlpha Alphanumerisch
- TypeBigInt Ganzzahlig (64 Bit)
- TypeByte Ganzzahlig (8 Bit)
- TypeCaltime Kalenderzeit
- TypeColor Farbe
- TypeDate Datum
- TypeDecimal Dezimal
- TypeEvent Ereignis
- TypeFloat Gleitkomma
- TypeFont Schriftart
- TypeHandle Deskriptor
- TypeInt Ganzzahlig (32 Bit)
- TypeLogic Logisch
- TypeNone Kein Typ
- TypeOther Unbekannter Typ
- TypePoint Punkt
- TypeRange Markierter Bereich
- TypeRect Rechteck
- TypeRTFTab Tabulator
- TypeTime Zeit
- TypeWord Ganzzahlig (16 Bit)

_TypeAlpha
Konstante für einen alphanumerischen Typ
Wert 2

Verwandte
Befehle,
Siehe FldInfoByName(),
KeyFldInfo(),
LinkFldInfo(),
ClnType

Das Feld / Die Variable / Die Spalte ist vom Typ alpha.

_TypeBigInt

Konstante für einen ganzzahligen Typ (64 Bit)

Wert 11

Verwandte

Befehle,

Siehe FldInfoByName(),

KeyFldInfo(),

LinkFldInfo(),

ClnType

Das Feld / Die Variable / Die Spalte ist vom Typ bigint.

_TypeByte

Konstante für einen ganzzahligen Typ (8 Bit)

Wert 8

Siehe Verwandte
Befehle

Die Variable ist vom Typ byte.

`_TypeCaltime`

Konstante für einen Kalenderzeit-Typ

Wert 20

Siehe [Verwandte](#)
[Befehle](#)

Die Variable ist vom Typ [caltime](#).

_TypeColor

Konstante für einen Farben-Typ

Wert 40

Siehe Verwandte
Befehle

Die Variable ist vom Typ color.

_TypeDate

Konstante für einen Datums-Typ

Wert 17

Verwandte

Befehle,

Siehe FldInfoByName(),

KeyFldInfo(),

LinkFldInfo(),

ClnType

Das Feld / Die Variable / Die Spalte ist vom Typ date.

_TypeDecimal
Konstante für einen Dezimal-Typ
Wert 12

Verwandte
Befehle,
Siehe FldInfoByName(),
KeyFldInfo(),
LinkFldInfo(),
ClnType

Das Feld / Die Variable / Die Spalte ist vom Typ decimal.

_TypeEvent

Konstante für einen Ereignistyp

Wert 34

Verwandte

Siehe Befehle,

event

Das Feld / Die Variable / Die Spalte ist vom Typ event.

_TypeFloat

Konstante für einen Gleitkomma-Typ

Wert 13

Verwandte

Befehle,

Siehe FldInfoByName(),

KeyFldInfo(),

LinkFldInfo(),

ClnType

Das Feld / Die Variable / Die Spalte ist vom Typ float.

_TypeFont

Konstante für einen Schriftart-Typ

Wert 35

Verwandte

Siehe Befehle,

font

Das Feld / Die Variable / Die Spalte ist vom Typ font.

_TypeHandle
Konstante für einen Deskriptor-Typ
Wert 36
Siehe Verwandte
Befehle
Die Variable ist vom Typ handle.

_TypeInt

Konstante für einen ganzzahligen Typ (32 Bit)

Wert 10

Verwandte

Befehle,

Siehe FldInfoByName(),

KeyFldInfo(),

LinkFldInfo(),

ClnType

Das Feld / Die Variable / Die Spalte ist vom Typ int.

_TypeLogic

Konstante für einen logischen Typ

Wert 16

Verwandte

Befehle,

Siehe FldInfoByName(),

KeyFldInfo(),

LinkFldInfo()

Das Feld / Die Variable / Die Spalte ist vom Typ logic.

Der Wert von den Befehlen FldInfo(), FldInfoByName(), KeyFldInfo() und LinkFldInfo() zurückgegeben.

`_TypeNone`

Konstante für keinen Typ

Wert 0

Siehe [Verwandte](#)

[Befehle](#)

Die Variable hat keinen Typ.

`_TypeOther`

Konstante für einen unbekannten Typ

Wert 37

Siehe [Verwandte](#)

[Befehle](#)

Die Variable ist von einem unbekannten Typ.

_TypePoint

Konstante für einen Point-Typ

Wert 32

Konstanten-

Siehe und

Feldtypen

Die Variable ist vom Typ point.

_TypeRange

Konstante für einen Range-Typ

Wert 38

Siehe Verwandte
Befehle

Die Variable ist vom Typ range.

_TypeRect

Konstante für einen Rect-Typ

Wert 33

Siehe Verwandte
Befehle

Die Variable ist vom Typ rect.

_TypeRTFTab
Konstante für einen RTFTab-Typ
Wert 39
Siehe Verwandte
Befehle
Die Variable ist vom Typ rtftab.

_TypeTime
Konstante für einen Zeit-Typ
Wert 18

Verwandte

Befehle,

Siehe FldInfoByName(),
KeyFldInfo(),
LinkFldInfo(),
ClnType

Das Feld / Die Variable / Die Spalte ist vom Typ time.

_TypeWord

Konstante für einen ganzzahligen Typ (16 Bit)

Wert 9

Siehe Verwandte
Befehle

Das Feld / Die Variable / Die Spalte ist vom Typ word.

Der Wert wird von den Befehlen FldInfo(), FldInfoByName(), KeyFldInfo() und LinkFldInfo() zurückgegeben.

Datentypen

Liste aller Datentypen

Befehlsgruppen,

Siehe Liste aller

Prozedurbefehle

Allgemeine Variablentypen

Bezeichnung im Editor	Bezeichnung im Datenstruktureditor
<u>alpha</u>	alphanumerisch
<u>byte</u>	ganzzahlig 8
<u>word</u>	ganzzahlig (kurz)
<u>int</u>	ganzzahlig (lang)
<u>bigint</u>	ganzzahlig 64
<u>float</u>	Gleitkomma
<u>decimal</u>	Dezimal
<u>date</u>	Datum
<u>time</u>	Zeit
<u>logic</u>	logisch

Strukturelle Variablentypen

event Ereignis

font Zeichensatz

point Punkt

range Markierter Bereich

rect Rechteck

rtftab Tabulator

Spezielle Variablentypen

calttime Kalenderzeit

handle Deskriptor

color Farbe

alpha - Datentyp
Alphanumerischer Typ

Verwandte

Siehe Befehle,

Edit

Bei der Deklaration von alpha kann optional noch eine Längenangabe folgen, wobei die Länge im Bereich von 1 bis 8192 Stellen liegt.

Beispiel:

```
aString : alpha(640);
```

Sofern keine Länge angegeben wird, werden 80 Stellen als Länge angenommen. Gültige Zeichen in alphanumerischen Werten liegen im Bereich der ASCII-Tabelle von 1 bis 255.

Wird einem Feld oder einer Variablen vom Typ alpha eine zu lange Zeichenkette zugewiesen, kommt es zu dem Laufzeitfehler "Alphawert zu lang". Über die Funktion ErrIgnore() mit der Option ErrStringOverflow kann die Generierung eines Laufzeitfehlers global abgeschaltet werden. In diesem Fall wird die Zeichenkette abgeschnitten.

Konstante

Eine alphanumerische Konstante wird in einfache Hochkommata eingeschlossen. Um in alphanumerischen Konstanten auch ein Hochkomma verwenden zu können, ist dieses als doppeltes Hochkomma " " anzugeben.

Beispiel:

```
'Dies ist eine sogenannte ''Zeichenkette'''
```

Operationen und Funktionen

Operator Funktion

+ Zeichenketten verbinden

Beispiel:

```
Zeile # Zeichenkette1 + Zeichenkette2 + Zeichenkette3
```

Nach der Deklaration ist die Variable mit dem Wert " initialisiert.

byte - Datentyp

Ganzzahliger Typ (8-bit ohne Vorzeichen)

Verwandte Befehle,

Siehe int,

Zahlendarstellung

Die zulässigen Werte liegen im Bereich von 0 bis 255. Anstelle von byte kann auch int8 verwendet werden.

Der Typ byte ist im wesentlichen nur bei Arrays von Bedeutung, da mit diesem Typ im Gegensatz zu int Speicherplatz gespart werden kann, sofern die zu erwartenden Werte im jeweiligen Wertebereich von byte liegen.

Nach der Deklaration ist eine Variable mit dem Wert 0 initialisiert.

word - Datentyp

Ganzzahliger Typ (16-bit ohne Vorzeichen)

Verwandte Befehle,

Siehe int,

Zahlendarstellung

Die zulässigen Werte liegen im Bereich von 0 bis 65535. Anstelle von word kann auch int16 verwendet werden.

Der Typ word ist im Wesentlichen nur bei Arrays von Bedeutung, da mit diesem Typ im Gegensatz zu int Speicherplatz gespart werden kann, sofern die zu erwartenden Werte im jeweiligen Wertebereich von word liegen.

Nach der Deklaration ist die Variable mit dem Wert 0 initialisiert.

int - Datentyp

Ganzzahliger Typ (32-bit mit Vorzeichen)

Verwandte Befehle,

Siehe IntEdit,

Zahlendarstellung

Die zulässigen Werte liegen im Bereich von $-2.147.483.647$ ($-2^{31}+1$) bis $+2.147.483.647$ ($2^{31}-1$). Anstelle von int kann auch long oder int32 verwendet werden.

Konstante

Ganzzahlige Konstanten bestehen in der dezimalen Notation nur aus Ziffern und dem optionalen Minuszeichen. In der hexadezimalen Notation muss die Konstante mit 0x beginnen und kann auch die Buchstaben A bis F enthalten. Konstanten können mit \l abgeschlossen werden.

Beispiele:

18991-77\l0x1B0F

Nach der Deklaration ist die Variable mit dem Wert 0 initialisiert.

bigint - Datentyp
Ganzzahliger Typ (64-bit mit Vorzeichen)

Verwandte Befehle,

Siehe BigIntEdit,

Zahlendarstellung

Die zulässigen Werte liegen im Bereich von $-9.223.372.036.854.775.807 (-2^{63}+1)$ bis $+9.223.372.036.854.775.807 (2^{63}-1)$.

Anstelle von bigint kann auch int64 verwendet werden.

Zur Speicherung von Variablen oder Datenbankfeldern dieses Typs werden 8 Byte benötigt.

Konstanten

Konstanten bestehen nur aus Ziffern und dem optionalen Minuszeichen und werden immer mit \b abgeschlossen.

Beispiele:

18991\b-77\b0x1B0F\b


Nach der Deklaration ist die Variable mit dem Wert 0\b initialisiert.

Methoden von bigint

Liste aller Methoden von bigint

Siehe bigint

- vmServerTime
- vmSystemTime

vmServerTime() : 
 date / time / caltime
 Serverzeit ermitteln

		Aktuelles
	<u>date</u> /	Datum /
Resultat	<u>time</u> /	aktuelle
	<u>caltime</u>	Uhrzeit vom
		Server

Siehe Methoden von date,
Methoden von time,
Methoden von caltime,
Methoden von bigint,
DbControl()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.


Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit des CONZEPT 16-Datenbankservers ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment      : caltime; tToday      : date; tNow      : time; tTimestamp : bigint;}
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes auf die Zeitzone des Client-Rechners gesetzt.

vmSystemTime() : 
 date / time / caltime
 Systemzeit ermitteln

Aktuelles
 Resultat date / time Datum /
 / caltime aktuelle
 Uhrzeit
 Methoden von date,
 Methoden von time,
 Siehe Methoden von caltime,
 Methoden von bigint,
 SysDate(), SysTime()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.

Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment : caltime; tToday : date; tNow : time; tTimestamp : bigint; }{ tM
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes gesetzt.

float - Datentyp

Gleitkommatyp (64-Bit mit Vorzeichen und 15 Stellen Genauigkeit)

Verwandte

Befehle, FloatEdit,

Siehe Zahlendarstellung,

Rundungsfehler

(Blog)

Die zulässigen Werte liegen im Bereich von -1.0E307 bis +1.0E307.

Konstante

Eine Gleitkommakonstante enthält immer einen Dezimalpunkt, der die Vorkomma- von den Nachkommastellen trennt. Um eine Verwechslung mit Konstanten des Typs decimal zu vermeiden, können die Konstanten mit `\f` abgeschlossen werden.

Beispiele

```
6712.012110.010.0\f-0.08821
```

Hinweise zur Verwendung

Bei der Verwendung von Gleitkommawerten ist immer damit zu rechnen, dass es zu Rundungsfehlern kommt. Ein Vergleich zweier Gleitkommawerte auf Gleichheit (`=`) ist somit zu vermeiden. Stattdessen sollte der Vergleich auf einen Bereich ausgedehnt oder der Typ decimal verwendet werden:

```
if ((ValueA > ValueB - Delta) and (ValueA < ValueB + Delta)){ // Values are equal ...}else{ //
```

Der Wert Delta sollte dabei so klein wie möglich gewählt werden. Er wird aber wesentlich durch die Berechnung der Werte beeinflusst.



Bei eindeutigen Schlüsseln sollten keine Gleitkommafelder verwendet werden. Durch die Umwandlung vom dezimalen in das binäre Zahlensystem entstehen Ungenauigkeiten, die unter Umständen dazu führen können, dass ein Datensatz nicht gespeichert oder gefunden werden kann.

Nach der Deklaration ist die Variable mit dem Wert `0\f` initialisiert.

decimal - Datentyp
 Dezimaltyp (256-bit mit Vorzeichen)

Verwandte Befehle,

Siehe DecimalEdit,
Zahlendarstellung

Dieser Typ speichert einen Gleitkommawert im Bereich von $-(10^{63}-1)$ bis $10^{63}-1$. Im Gegensatz zur Speicherung im Binärformat beim float-Typ werden Werte im dezimalen Format abgelegt, wodurch eine exakte Repräsentation aller Werte des Wertebereichs gegeben ist. Die Genauigkeit ist mit 58 Stellen fast 4 mal so hoch wie beim float-Typ. Durch das dezimale Format werden Rundungsfehler bei Berechnung vermieden. Außerdem kann eine Variable vom Typ decimal den Wert "undefiniert" enthalten.

Variablen oder Felder diesen Typs belegen 32 Bytes im Hauptspeicher. In der Datenbank werden die Werte mit variabler Länge gespeichert. Dabei belegt der Wert "undefiniert" 1 Byte, alle anderen Werte 2 Bytes plus die halbe Anzahl von signifikanten Ziffern.

Beispiele:

<u>_DecimalUndef</u>	belegt 1 Byte
0	belegt 2 Bytes
1.0	belegt 3 Bytes (1 Ziffer)
17.43	belegt 4 Bytes (4 Ziffern)
2100	belegt 3 Bytes (2 Ziffern)
0.0000214	belegt 4 Bytes (3 Ziffern)
17867001	belegt 6 Bytes (8 Ziffern)

In Prozeduren wird die Typbezeichnung decimal verwendet. Es findet keine automatische Umwandlung zwischen decimal und anderen Typen statt. Für die Umwandlung stehen entsprechende Cnv...-Befehle zur Verfügung. Für dezimale Konstanten wird in Prozeduren die Zeichenfolge \m an den Wert angehängen. Alternativ kann die Exponentialdarstellung verwendet werden.

Beispiele:

```
17\m-1.7681\m0.0000005\m5.433E5\m
```

Bei Berechnungen mit Dezimal-Werten stehen fast alle mathematischen Funktionen zur Verfügung, die auch für den float-Typ vorhanden sind (Ausgenommen trigonometrische Funktionen). Da die Berechnungen nicht direkt vom Prozessor ausgeführt werden können und mit hoher Genauigkeit durchgeführt werden, brauchen die Operationen mehr Rechenzeit als beim float-Typ. Dies ist besonders bei komplexeren Funktionen wie LogN() oder Exp() zu berücksichtigen.

Berechnungen mit dem undefinierten Wert _DecimalUndef liefern immer ein undefiniertes Ergebnis.

Tritt bei den Berechnungen ein ungültiges Ergebnis auf, ist das Resultat _DecimalError. Dieser Wert ist nur temporär vorhanden, bei weiteren Berechnungen

oder beim Speichern wird er automatisch in DecimalUndef gewandelt. Ungültige Ergebnisse können beispielsweise bei Division durch 0, durch einen Überlauf (Betrag des Wertes $> 10^{63}-1$) oder einen Unterlauf (Betrag des Wertes $< 10^{-63}$) entstehen. Ein ungültiges Ergebnis führt zu einem Laufzeitfehler. Diese können mit ErrIgnore(ErrDecimal, true) komplett abgeschaltet werden.

Nach der Deklaration einer Variablen besitzt sie den Wert 0\m.

Für Felder diesen Typs gibt es die Konstante TypeDecimal.

date - Datentyp

Datumstyp (32-Bit)

Verwandte

Befehle,

Siehe Eigenschaften,

Methoden,

DateEdit

Die zulässigen Werte liegen im Bereich vom 1.1.1900 bis 31.12.2154.

Konstante

Bei Datumswerten enthält die Konstante zwei Punkte, die die einzelnen Teile der Konstanten trennen. Das Datum wird immer in der Form Tag.Monat.Jahr dargestellt. Ein leeres Datum wird mit 0.0.0 dargestellt.

Beispiele:

16.3.8131.10.199301.07.2004

Der Datumswert kann ebenfalls über die Eigenschaften vpYear, vpMonth und vpDay gesetzt werden.

Nach der Deklaration ist die Variable mit dem Wert 0.0.0 initialisiert.

Eigenschaften von date

Liste aller Eigenschaften von date

Siehe date

- vpDay
- vpDayOfWeek
- vpLeapYear
- vpMonth
- vpWeek
- vpWeekYear
- vpYear

vpYear

Jahr eines Datums

Typ int

Eigenschaften

von date,

Siehe Eigenschaften

von caltime,

vpDate

In dieser Eigenschaft kann das Jahr gesetzt bzw. abgefragt werden.

Wird die Eigenschaft für eine Variable vom Typ caltime gesetzt, können Werte im Bereich 1601 bis 30000 eingetragen werden. In diesem Fall kann das Datum ebenfalls mit der Eigenschaft vpDate gesetzt oder abgefragt werden. Hier gilt dann der Wertebereich des Datentyps date.

Wird die Eigenschaft für eine Variable vom Typ date gesetzt, können Werte im Bereich 1900 bis 2154 angegeben werden.

Wird ein Wert außerhalb dieses Bereiches angegeben, wird der Laufzeitfehler ErrValueRange generiert.

vpMonth

Monat eines Datums

Typ int

Eigenschaften

von date,

Siehe Eigenschaften

von caltime,

vpDate

In dieser Eigenschaft kann der Monat des Datums gesetzt oder abgefragt werden. Zulässige Werte liegen im Bereich 1 bis 12. Wird ein Wert außerhalb dieses Bereiches angegeben, erfolgt der Laufzeitfehler ErrValueRange.

Das vollständige Datum kann ebenfalls mit der Eigenschaft vpDate gesetzt oder abgefragt werden.

vpDay

Tag eines Datums

Typ int

Eigenschaften

von date,

Siehe Eigenschaften

von caltime,

vpDate

In dieser Eigenschaft kann der Tag des Datums abgefragt oder gesetzt werden.

Zulässige Werte liegen im Bereich 1 bis 31.

Diese Eigenschaft kann erst gesetzt werden, nachdem die Eigenschaften vpYear und vpMonth gesetzt wurden. Wird der Wert 31 angegeben, wird der Wert automatisch auf den letzten Tag des Monats gesetzt.

```
cMoment->vpYear # 2000;cMoment->vpMonth # 2;cMoment->vpDay # 31; // Day wird auf 29 gesetzt
```

Das vollständige Datum kann ebenfalls mit der Eigenschaft vpDate gesetzt oder abgefragt werden.

vpLeapYear

Schaltjahr eines Datums ermitteln

Typ logic

Siehe Eigenschaften von date,

Eigenschaften von caltime

Diese Eigenschaft kann nur abgefragt werden. Die Eigenschaft hat den Wert true, wenn das angegebene Datum ein Schaltjahr ist, sonst ist der Wert false.

Beispiel:

```
cMoment->vpDate # 01.01.2000;if (cMoment->vpLeapYear){ // Schaltjahr ...}
```

vpWeek

Kalenderwoche eines Datums

Typ int

Eigenschaften von date,

Siehe Eigenschaften von
calttime, vpWeekYear

Diese Eigenschaft kann nur abgefragt und nicht gesetzt werden. In der Eigenschaft steht die Kalenderwoche des gesetzten Datums. Das zu dieser Kalenderwoche gehörende Jahr kann in der Eigenschaft vpWeekYear abgefragt werden.

vpWeekYear

Jahr einer Kalenderwoche

Typ int

Eigenschaften von

Siehe date, Eigenschaften
von caltime, vpWeek

Diese Eigenschaft kann nur abgefragt und nicht gesetzt werden. In der Eigenschaft steht das Jahr der Kalenderwoche des gesetzten Datums. Die Kalenderwoche kann in der Eigenschaft vpWeek abgefragt werden.

Beispiel

```
tDate # 30.12.2007;tWeek # tDate->vpWeek; // 52tWeekYear # tDate->vpWeekYear; // 2007tDate # 31.1
```

vpDayOfWeek

Wochentag eines Datums

Typ int

Eigenschaften von

Siehe date, Eigenschaften
von caltime

Diese Eigenschaft kann nur abgefragt werden. Sie liefert die Nummer des Wochentages zurück.

Beispiel:

```
switch (cMoment->vpDayOfWeek){ case 1, 2, 3, 4, 5 : { // Werktag ... } case 6, 7 : {
```

Methoden von date

Liste aller Methoden von date

Siehe date

- vmDayModify
- vmEasterDate
- vmMonthModify
- vmServerTime
- vmSystemTime

vmSystemTime() : 
 date / time / caltime
 Systemzeit ermitteln

Aktuelles
 Resultat date / time Datum /
 / caltime aktuelle
 Uhrzeit

Methoden von date,
Methoden von time,
 Siehe Methoden von caltime,
Methoden von bigint,
SysDate(), SysTime()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.

Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment      : caltime; tToday      : date; tNow        : time; tTimestamp : bigint; }{ tM
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes gesetzt.

vmServerTime() : 
date / time / caltime
Serverzeit ermitteln

Aktuelles
date / Datum /
Resultat time / aktuelle
caltime Uhrzeit vom
Server

Methoden von date,
Methoden von time,
Siehe Methoden von caltime,
Methoden von bigint,
DbControl()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.

Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit des CONZEPT 16-Datenbankservers ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment      : caltime; tToday      : date; tNow      : time; tTimestamp : bigint;}
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes auf die Zeitzone des Client-Rechners gesetzt.

vmMonthModify(int1)
: date / caltime
Monat modifizieren
int1 Differenz
Resultat date / Veränderter
caltime Datumswert
Methoden von date,
Siehe Methoden von time,
Methoden von caltime,
vmDayModify

Diese Methode kann für die Datentypen date und caltime aufgerufen werden.

Mit dieser Methode kann der Monat eines Datumswertes verändert werden. Die Differenz des Wertes wird als (int1) übergeben. Hier sind sowohl positive als auch negative Werte zulässig.

Überschreitet die Differenz eine Jahresgrenze, wird das Jahr mit verändert.

Ist der in dem Datum enthaltene Tag nicht in dem veränderten Monat enthalten, wird der Tag auf den letzten Tag des neuen Monats gesetzt.

Beispiele:

```
cMoment->vpDate # 30.03.2002;cMoment->vmMonthModify(-1); // ein Monat zurück = 28.02.2002
```

Mögliche Laufzeitfehler:

ErrValueRange Das neue Datum liegt außerhalb des gültigen Bereiches von date oder caltime.

vmDayModify(int1)
: date / caltime
Tag modifizieren
int1 Differenz
Resultat date / Veränderter
caltime Datumswert
Methoden von date,
Siehe Methoden von
caltime,
vmMonthModify

Diese Methode kann für die Datentypen date und caltime aufgerufen werden.

Mit dieser Methode kann der Tag eines Datumswertes verändert werden. Die Differenz des Wertes wird als (int1) übergeben. Hier sind sowohl positive als auch negative Werte zulässig.


Überschreitet die Differenz eine Monats- oder Jahresgrenze, wird ebenfalls der Monat bzw. das Jahr mit verändert.

Beispiele:

```
cMoment->vpDate # 04.03.2002;cMoment->vmDayModify(-7); // eine Woche zurück = 25.02.2002
```

Mögliche Laufzeitfehler:

_ErrValueRange Das neue Datum liegt außerhalb des gültigen Bereiches von date oder caltime.

vmEasterDate(int1) : 
 date / caltime
 Ostersonntag ermitteln
 int1 Jahr

Resultat date / Ostersonntag des
caltime angegebenen
 Jahres

Siehe Methoden von date,
Methoden von caltime

Diese Methode kann für die Datentypen date und caltime aufgerufen werden.

Mit dieser Methode wird der Ostersonntag für das in (int1) angegebene Jahr zurückgegeben. Das Jahr kann im Bereich 1700 bis 2199 angegeben werden.

Alle andere kirchlichen Feiertage beziehen sich auf den Ostersonntag. Mit folgendem Define können die anderen Feiertage berechnet werden:

```
define{ // Konstanten zur Berechnung des entsprechenden Datums mAschermittwoch : -46 mKarfreit
```

Beispiele:

```
cHimmelfahrt->vmEasterDate(2006);cHimmelfahrt->vmDayModify(mHimmelfahrt);
```

time - Datentyp

Zeittyp (32-Bit)

Verwandte

Befehle,

Siehe Eigenschaften,

Methoden,

TimeEdit

Die zulässigen Werte liegen im Bereich von 00:00:00.00 bis 23:59:59.99. Ein leerer Zeitwert wird durch 24:00:00.00 dargestellt.

Nach der Deklaration ist die Variable mit dem Wert 00:00:00.0 initialisiert.

Konstante

Bei Zeitwerten enthält die Konstante mindestens einen Doppelpunkt. Die Zeit wird immer im Format Stunden:Minuten:Sekunden.Hundertstelsekunden dargestellt, wobei die Angabe von Sekunden und Hundertstelsekunden optional ist.

Beispiele:

```
local{ tTime : time;}{ ... tTime # 16:15; tTime # 1:20:56; tTime # 22:01:00.78; tTime->vmS}
```

Eigenschaften von time

Liste aller Eigenschaften von time

Siehe time

- vpHours
- vpMilliseconds
- vpMinutes
- vpSeconds

vpHours

Stunden einer Uhrzeit

Typ int

Eigenschaften von

Siehe time,

Eigenschaften von

caltime, vpTime

Über diese Eigenschaft können die Stunden der Uhrzeit gesetzt oder abgefragt werden. Es können Werte im Bereich 0 bis 23 angegeben werden.

Bei der Angabe eines Wertes außerhalb dieses Bereiches wird der Laufzeitfehler __ErrValueRange generiert.

Die vollständige Uhrzeit kann ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden.

vpMinutes

Minuten einer Uhrzeit

Typ int

Eigenschaften von

Siehe time,

Eigenschaften von

caltime, vpTime

Mit dieser Eigenschaft können die Minuten eines Zeitwertes gesetzt oder abgefragt werden. Die zulässigen Werte liegen im Bereich 0 bis 59. Wird ein Wert außerhalb dieses Bereiches angegeben, wird der Laufzeitfehler ErrValueRange generiert.

Die vollständige Uhrzeit kann ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden.

vpSeconds

Sekunden einer Uhrzeit

Typ int

Eigenschaften von

Siehe time, Eigenschaften
von caltime, vpTime

Mit dieser Eigenschaft können die Sekunden eines Zeitwertes gesetzt oder abgefragt werden. Die zulässigen Werte liegen in dem Bereich 0 bis 59. Wird ein Wert außerhalb dieses Bereiches angegeben, erfolgt der Laufzeitfehler ErrValueRange.

Die vollständige Uhrzeit kann ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden.

vpMilliseconds

Millisekunden einer Uhrzeit

Typ int

Eigenschaften von time,

Siehe Eigenschaften von
caltime, vpTime

In dieser Eigenschaft können die Millisekunden einer Uhrzeit gesetzt oder abgefragt werden. Die zulässigen Werte liegen im Bereich 0 bis 999. Wird ein Wert außerhalb dieses Bereiches angegeben, wird der Laufzeitfehler ErrValueRange generiert.

Wird die Eigenschaft bei einer Variablen vom Typ time gesetzt, werden nur die 100stel übertragen. Die 1000stel werden abgeschnitten.

Bei Variablen vom Typ caltime kann die vollständige Uhrzeit ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden. Hier können allerdings nur 100stel Sekunden angegeben werden. Die 1000stel werden dabei abgeschnitten.

Methoden von time

Liste aller Methoden von time

Siehe time

- vmSecondsModify
- vmServerTime
- vmSystemTime

vmSystemTime() : 
 date / time / caltime
 Systemzeit ermitteln

Aktuelles
 Resultat date / time Datum /
 / caltime aktuelle
 Uhrzeit
Methoden von date,
Methoden von time,
 Siehe Methoden von caltime,
Methoden von bigint,
SysDate(), SysTime()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.


Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment      : caltime; tToday      : date; tNow        : time; tTimestamp : bigint; }{ tM
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes gesetzt.

vmServerTime() : 
 date / time / caltime
 Serverzeit ermitteln

		Aktuelles
	<u>date</u> /	Datum /
Resultat	<u>time</u> /	aktuelle
	<u>caltime</u>	Uhrzeit vom
		Server

Siehe Methoden von date,
Methoden von time,
Methoden von caltime,
Methoden von bigint,
DbControl()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.


Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit des CONZEPT 16-Datenbankservers ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment      : caltime; tToday      : date; tNow      : time; tTimestamp : bigint;}
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes auf die Zeitzone des Client-Rechners gesetzt.

vmSecondsModify(int1) 

: time / caltime

Sekunden modifizieren

int1 Differenz

Resultat time / Veränderter
caltime Zeitwert

Siehe Methoden von time,
Methoden von caltime

Die Methode kann für Variablen der Typen time und caltime aufgerufen werden.

Mit dieser Methode kann ein Zeitwert verändert werden. Die Differenz des Wertes wird als (int1) übergeben. Hier sind sowohl positive als auch negative Werte zulässig.

Wird diese Methode für eine Variable vom Typ caltime aufgerufen, wirken sich Überschreitungen von Tagen, Monaten oder Jahren auch auf das Datum aus.

Beispiele:

```
cMoment->vmSystemTime();cMoment->vmSecondsModify(-3600); // Uhrzeit vor einer Stunde
```

Mögliche Laufzeitfehler:

ErrValueRange Die Tagesgrenze wurde überschritten und das neue Datum liegt außerhalb des gültigen Bereiches von caltime.

logic - Datentyp

Logischer Typ (8-Bit)

Verwandte

Siehe Befehle,

Checkbox

Bei diesem Typ werden als logische Konstanten die Schlüsselwörter y bzw. true für "wahr" und n bzw. false für "falsch" benutzt. Siehe auch Vergleichsoperatoren.

event - Datentyp - Struktur

Diese Struktur beschreibt ein Ereignis

Verwandte

Befehle,

Siehe NULL,

Liste aller

Ereignisse

Die Struktur event besteht aus folgenden Feldern:

Element Typ Verwendung

ID int Nummer des Ereignisses

Obj int Deskriptor des Objektes, welches das Ereignis ausgelöst hat.

Pos point Mausposition relativ zur oberen linken Ecke des angeklickten Objektes.

In der Programmierung kann ein Teil der Struktur mit <Name der Variablen>:<Name des Feldes> referenziert werden.

Beispiele:

```
// Name des auslösenden Objektes ermitteltName # aEvt:Obj->wpName;// X-Koordinate der Maus ermittelt
```

font - Datentyp - Struktur

Struktur zum Auslesen und Setzen einer Schriftart

Verwandte Befehle,

Siehe Eigenschaft Font,

NULL,

Schriftartkonstanten

Mit der Struktur font können Schriftarten gelesen und gesetzt werden. Die Struktur font besteht aus folgenden Feldern:

Element Typ Verwendung

Name alpha Name der Schriftart

Size int Größe der Schriftart in zehntel Punkten

Attributes int Attribute der Schriftart

In der Programmierung kann ein Teil der Struktur mit <Name der Variablen>:<Name des Feldes> referenziert werden. Die Angabe der Größe der Schriftart erfolgt in zehntel Punkten.

Beispiel:

```
tFont # $Button->wpFont;// unterstreichen ein- oder ausschaltenif ((tFont:Attributes & _WinFontAt
```


point - Datentyp - Struktur

Diese Struktur beschreibt einen Punkt

Verwandte

Siehe Befehle,

PointMake(),

NULL

Die Struktur point besteht aus folgenden Feldern:

Element Typ Verwendung

x int Horizontale Position des Punktes

y int Vertikale Position des Punktes

In der Programmierung kann ein Teil der Struktur mit <Name der Variablen>:<Name des Feldes> referenziert werden.

Beispiel

```
local{ tPunkt : point;}{ ... tPunkt:x # 100; tPunkt:y # 80; // alternativ: tPunkt # PointM
```

range - Datentyp - Struktur

Diese Struktur beschreibt eine Markierung

Verwandte

Siehe Befehle,

RangeMake(),

NULL

Die Struktur range besteht aus folgenden Feldern:

Element Typ Verwendung

min int Anfang der Markierung

max int Ende der Markierung

Die Elemente stellen die Positionen der Markierung dar. Die Markierung beginnt nach dem in min angegebenen Zeichen und endet nach dem in max angegebenen Zeichen.

In der Programmierung kann ein Teil der Struktur mit <Name der Variablen>:<Name des Feldes> referenziert werden.

Ein markierter Text kann mit Hilfe von Formatierungsanweisungen formatiert werden.

Durch die Angabe der gleichen Werte in den Elementen min und max, wird die Eingabemarke an die entsprechende Position gesetzt. Die Eingabemarke kann mit (-1,-1) an das Ende des Textes gesetzt werden.

Beispiele:

```
// Bereich mit rotem HintergrundtRange:min # 0;tRange:max # 30;// alternativtRange # RangeMake(0,
```

rect - Datentyp - Struktur

Diese Struktur beschreibt ein Rechteck

Verwandte

Siehe Befehle,
RectMake(),
NULL

Die Struktur rect besteht aus folgenden Elementen:

Element Typ Verwendung

left int Position der linken Kante
top int Position der oberen Kante
right int Position der rechten Kante
bottom int Position der unteren Kante

In der Programmierung kann ein Teil der Struktur mit <Name der Variablen>:<Name des Feldes> referenziert werden.

Ein Rechteck kann mit den Befehlen WinPropGet() oder WinPropSet() und über die Eigenschaft Area ausgelesen bzw. zugewiesen werden.

Beispiel

```
local{ tRect : rect;}{ ... tRect:left   # 100; tRect:top     # 80; tRect:right  # 140; tRe
```

rtftab - Datentyp - Struktur

Diese Struktur beschreibt einen Tabulator

Verwandte

Siehe Befehle,

RtfTabMake(),

NULL

Die Struktur rtftab besteht aus folgenden Feldern:

Element Typ Verwendung

TabPos int Position des Tabulators

TabType int Typ des Tabulators

In der Programmierung kann ein Teil der Struktur mit <Name der Variablen>:<Name des Feldes> referenziert werden.

Beispiele:

```
tRtfTab:TabPos # PrtUnitLog(4.0, _PrtUnitCentimetres);tRtfTab:TabTyp # _WinRtfTabCenter;
```

caltime - Datentyp
Kalenderzeit

Verwandte
Befehle,
Eigenschaften,
Methoden,
Siehe NULL, UTC
und lokale
Zeit (Blog),
Eigenschaften
und Methoden
(Blog)

Dieser Datentyp enthält Informationen zum Datum und zur Uhrzeit. Eine Variable dieses Typs verfügt über mehrere Eigenschaften und Methoden. Der Wertebereich geht vom 01.01.1601 bis zum 31.12.30000 in 100 Nanosekunden. Eine Variable wird wie folgt definiert:

```
cMoment : caltime;
```

Eigenschaften

Eigenschaften der Variablen werden mit dem Operator "->" angesprochen. Folgende Eigenschaften sind vorhanden:

- vpYear Jahreszahl
- vpMonth Monat
- vpDay Tag
- vpHours Stunden
- vpMinutes Minuten
- vpSeconds Sekunden
- vpMilliseconds Millisekunden
- vpDate Datumswert
- vpTime Zeitwert
- vpBiasMinutes Zeitzoneabweichung

Nach der Deklaration ist die Variable mit dem Wert NULL initialisiert. Alle Eigenschaften haben den Wert 0 bzw. 0.0.0 und 00:00:00.0.





Wird das Datum manuell initialisiert (nicht mit vmSystemTime(), vmServerTime() oder vpDate), muss dieses in der Reihenfolge vpYear, vpMonth, vpDay gesetzt werden.

Folgende Eigenschaften können nur ausgelesen und nicht gesetzt werden:

- vpLeapYear Schaltjahr
- vpWeek Kalenderwoche
- vpWeekYear Jahr der Kalenderwoche
- vpDayOfWeek Wochentag

Beispiele:

```
// Datum setzen cMoment->vpYear # 2002; cMoment->vpMonth # 8; cMoment->vpDay # 29; // alternativ:
```

-  Die Eigenschaft vpBiasMinutes kann nur abgefragt werden, wenn die Methode vmSystemTime aufgerufen oder der Wert zuvor gesetzt wurde.
-  Beim Vergleichen von caltime-Werten wird der Datums- und Zeitwert ohne die Zeitzonenabweichung (vpBiasMinutes) verglichen. Somit entspricht beispielsweise 15.03.2013 15:52:23 UTC+1 = 15.03.2013 16:52:23 UTC+2.

Methoden

Die Eigenschaften einer Variablen können mit Methoden verändert werden. Die Methoden werden mit dem Operator "->" aufgerufen. Folgende Methoden sind für diesen Datentyp definiert:

- vmSystemTime() Systemzeit ermitteln
- vmServerTime() Systemzeit des Servers ermitteln
- vmEasterDate() Datum des Ostersonntag ermitteln
- vmMonthModify() Monat ändern
- vmDayModify() Tag ändern
- vmSecondsModify() Uhrzeit ändern

Beispiele:

```
// Systemzeit ermitteln und Zeitzone abfragen cMoment->vmSystemTime(); iTimeBias # cMoment->vpBiasM
```

Eigenschaften von caltime

Liste aller Eigenschaften von caltime

Siehe caltime

- vpBiasMinutes
- vpDate
- vpDay
- vpDayOfWeek
- vpHours
- vpLeapYear
- vpMilliseconds
- vpMinutes
- vpMonth
- vpSeconds
- vpTime
- vpWeek
- vpWeekYear
- vpYear

vpYear

Jahr eines Datums

Typ int

Eigenschaften

von date,

Siehe Eigenschaften

von caltime,

vpDate

In dieser Eigenschaft kann das Jahr gesetzt bzw. abgefragt werden.

Wird die Eigenschaft für eine Variable vom Typ caltime gesetzt, können Werte im Bereich 1601 bis 30000 eingetragen werden. In diesem Fall kann das Datum ebenfalls mit der Eigenschaft vpDate gesetzt oder abgefragt werden. Hier gilt dann der Wertebereich des Datentyps date.

Wird die Eigenschaft für eine Variable vom Typ date gesetzt, können Werte im Bereich 1900 bis 2154 angegeben werden.

Wird ein Wert außerhalb dieses Bereiches angegeben, wird der Laufzeitfehler ErrValueRange generiert.

vpMonth

Monat eines Datums

Typ int

Eigenschaften

von date,

Siehe Eigenschaften

von caltime,

vpDate

In dieser Eigenschaft kann der Monat des Datums gesetzt oder abgefragt werden. Zulässige Werte liegen im Bereich 1 bis 12. Wird ein Wert außerhalb dieses Bereiches angegeben, erfolgt der Laufzeitfehler ErrValueRange.

Das vollständige Datum kann ebenfalls mit der Eigenschaft vpDate gesetzt oder abgefragt werden.

vpDay

Tag eines Datums

Typ int

Eigenschaften

von date,

Siehe Eigenschaften

von caltime,

vpDate

In dieser Eigenschaft kann der Tag des Datums abgefragt oder gesetzt werden.

Zulässige Werte liegen im Bereich 1 bis 31.

Diese Eigenschaft kann erst gesetzt werden, nachdem die Eigenschaften vpYear und vpMonth gesetzt wurden. Wird der Wert 31 angegeben, wird der Wert automatisch auf den letzten Tag des Monats gesetzt.

```
cMoment->vpYear # 2000;cMoment->vpMonth # 2;cMoment->vpDay # 31; // Day wird auf 29 gesetzt
```

Das vollständige Datum kann ebenfalls mit der Eigenschaft vpDate gesetzt oder abgefragt werden.

vpDate

Datum

Typ date

Verwandte

Befehle,

Siehe vpYear,

vpMonth,

vpDay

Mit dieser Eigenschaft kann das Datum eines calttime-Datentypes abgefragt oder gesetzt werden.

Wurde der Datumswert zuvor über die Eigenschaften vpYear, vpMonth und vpDay gesetzt, kann das Datum nur dann abgefragt werden, wenn es im Wertebereich des Datentyps date (01.01.1900 bis 31.12.2154) liegt. Erfolgt dennoch eine Abfrage, wird der Laufzeitfehler ErrValueRange generiert.

vpHours

Stunden einer Uhrzeit

Typ int

Eigenschaften von

Siehe time,

Eigenschaften von

caltime, vpTime

Über diese Eigenschaft können die Stunden der Uhrzeit gesetzt oder abgefragt werden. Es können Werte im Bereich 0 bis 23 angegeben werden.

Bei der Angabe eines Wertes außerhalb dieses Bereiches wird der Laufzeitfehler __ErrValueRange generiert.

Die vollständige Uhrzeit kann ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden.

vpMinutes

Minuten einer Uhrzeit

Typ int

Eigenschaften von

Siehe time,

Eigenschaften von

caltime, vpTime

Mit dieser Eigenschaft können die Minuten eines Zeitwertes gesetzt oder abgefragt werden. Die zulässigen Werte liegen im Bereich 0 bis 59. Wird ein Wert außerhalb dieses Bereiches angegeben, wird der Laufzeitfehler ErrValueRange generiert.

Die vollständige Uhrzeit kann ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden.

vpSeconds

Sekunden einer Uhrzeit

Typ int

Eigenschaften von

Siehe time, Eigenschaften
von caltime, vpTime

Mit dieser Eigenschaft können die Sekunden eines Zeitwertes gesetzt oder abgefragt werden. Die zulässigen Werte liegen in dem Bereich 0 bis 59. Wird ein Wert außerhalb dieses Bereiches angegeben, erfolgt der Laufzeitfehler ErrValueRange.

Die vollständige Uhrzeit kann ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden.

vpMilliseconds

Millisekunden einer Uhrzeit

Typ int

Eigenschaften von time,

Siehe Eigenschaften von
caltime, vpTime

In dieser Eigenschaft können die Millisekunden einer Uhrzeit gesetzt oder abgefragt werden. Die zulässigen Werte liegen im Bereich 0 bis 999. Wird ein Wert außerhalb dieses Bereiches angegeben, wird der Laufzeitfehler ErrValueRange generiert.

Wird die Eigenschaft bei einer Variablen vom Typ time gesetzt, werden nur die 100stel übertragen. Die 1000stel werden abgeschnitten.

Bei Variablen vom Typ caltime kann die vollständige Uhrzeit ebenfalls mit der Eigenschaft vpTime gesetzt oder abgefragt werden. Hier können allerdings nur 100stel Sekunden angegeben werden. Die 1000stel werden dabei abgeschnitten.

vpTime

Uhrzeit

Typ time

Verwandte

Befehle,

Siehe vpHours,

vpMinutes,

vpSeconds,

vpMilliseconds

Mit dieser Eigenschaft kann die Uhrzeit eines caltime-Datentypes abgefragt oder gesetzt werden.

Der Zeitwert kann auch über die Eigenschaften vpHours, vpMinutes, vpSeconds und vpMilliseconds gesetzt oder abgefragt werden.

vpLeapYear

Schaltjahr eines Datums ermitteln

Typ logic

Siehe Eigenschaften von date,

Eigenschaften von caltime

Diese Eigenschaft kann nur abgefragt werden. Die Eigenschaft hat den Wert true, wenn das angegebene Datum ein Schaltjahr ist, sonst ist der Wert false.

Beispiel:

```
cMoment->vpDate # 01.01.2000;if (cMoment->vpLeapYear){ // Schaltjahr ...}
```

vpWeek

Kalenderwoche eines Datums

Typ int

Eigenschaften von date,

Siehe Eigenschaften von
calttime, vpWeekYear

Diese Eigenschaft kann nur abgefragt und nicht gesetzt werden. In der Eigenschaft steht die Kalenderwoche des gesetzten Datums. Das zu dieser Kalenderwoche gehörende Jahr kann in der Eigenschaft vpWeekYear abgefragt werden.

vpWeekYear

Jahr einer Kalenderwoche

Typ int

Eigenschaften von

Siehe date, Eigenschaften
von caltime, vpWeek

Diese Eigenschaft kann nur abgefragt und nicht gesetzt werden. In der Eigenschaft steht das Jahr der Kalenderwoche des gesetzten Datums. Die Kalenderwoche kann in der Eigenschaft vpWeek abgefragt werden.

Beispiel

```
tDate # 30.12.2007;tWeek # tDate->vpWeek; // 52tWeekYear # tDate->vpWeekYear; // 2007tDate # 31.1
```

vpDayOfWeek

Wochentag eines Datums

Typ int

Eigenschaften von

Siehe date, Eigenschaften
von caltime

Diese Eigenschaft kann nur abgefragt werden. Sie liefert die Nummer des Wochentages zurück.

Beispiel:

```
switch (cMoment->vpDayOfWeek){ case 1, 2, 3, 4, 5 : { // Werktag ... } case 6, 7 : {
```

vpBiasMinutes

Zeitzoneabweichung zur UTC

Typ int

Siehe Verwandte Befehle,

vmSystemTime()

In dieser Eigenschaft steht die Abweichung der Zeit auf Grund der Zeitzone von der UTC (Coordinated Universal Time). Die Angabe erfolgt in Minuten.

Die zulässigen Werte liegen im Bereich vom -720 bis +720.

Die mitteleuropäische Sommerzeit weicht um +120 Minuten von der UTC ab.

Die Eigenschaft wird nur gesetzt, wenn eine Zuweisung aus einer calttime-Variablen erfolgt, in der die Eigenschaft bereits gesetzt ist, oder die Methode vmSystemTime() oder vmServerTime() aufgerufen wird. Die Zeitzone muss korrekt gesetzt sein, wenn der Wert mit dem Befehl CnvBC() gewandelt werden soll.

Methoden von caltime

Liste aller Methoden von caltime

Siehe caltime

- vmDayModify
- vmEasterDate
- vmMonthModify
- vmSecondsModify
- vmServerTime
- vmSystemTime

vmSystemTime() : 
 date / time / caltime
 Systemzeit ermitteln

Aktuelles
 Resultat date / time Datum /
 / caltime aktuelle
 Uhrzeit

Methoden von date,
Methoden von time,
 Siehe Methoden von caltime,
Methoden von bigint,
SysDate(), SysTime()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.

Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment      : caltime; tToday      : date; tNow      : time; tTimestamp : bigint; }{ tM
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes gesetzt.

vmServerTime() : 
 date / time / caltime
 Serverzeit ermitteln

		Aktuelles
	<u>date</u> /	Datum /
Resultat	<u>time</u> /	aktuelle
	<u>caltime</u>	Uhrzeit vom
		Server

Siehe Methoden von date,
Methoden von time,
Methoden von caltime,
Methoden von bigint,
DbControl()

Die Methode kann für Variablen der Typen caltime, date, time und bigint aufgerufen werden.

Mit dieser Methode kann das aktuelle Datum und / oder die aktuelle Uhrzeit des CONZEPT 16-Datenbankservers ermittelt werden. Wird diese Methode auf eine Variable des Typs bigint angewendet, erhält die Variable einen 64-Bit Zeitstempel (Zeiteinheiten nach 1601, eine Zeiteinheit beträgt 100 Nanosekunden).

Beispiele:

```
local{ tMoment      : caltime; tToday      : date; tNow      : time; tTimestamp : bigint;}
```



Bei Variablen vom Typ caltime wird zusätzlich die Eigenschaft vpBiasMinutes auf die Zeitzone des Client-Rechners gesetzt.

vmMonthModify(int1)
: date / caltime
Monat modifizieren
int1 Differenz
Resultat date / Veränderter
caltime Datumswert
Methoden von date,
Siehe Methoden von time,
Methoden von caltime,
vmDayModify

Diese Methode kann für die Datentypen date und caltime aufgerufen werden.

Mit dieser Methode kann der Monat eines Datumswertes verändert werden. Die Differenz des Wertes wird als (int1) übergeben. Hier sind sowohl positive als auch negative Werte zulässig.

Überschreitet die Differenz eine Jahresgrenze, wird das Jahr mit verändert.

Ist der in dem Datum enthaltene Tag nicht in dem veränderten Monat enthalten, wird der Tag auf den letzten Tag des neuen Monats gesetzt.

Beispiele:

```
cMoment->vpDate # 30.03.2002;cMoment->vmMonthModify(-1); // ein Monat zurück = 28.02.2002
```

Mögliche Laufzeitfehler:

ErrValueRange Das neue Datum liegt außerhalb des gültigen Bereiches von date oder caltime.

vmDayModify(int1)
: date / caltime
Tag modifizieren
int1 Differenz
Resultat date / Veränderter
caltime Datumswert
Methoden von date,
Siehe Methoden von
caltime,
vmMonthModify

Diese Methode kann für die Datentypen date und caltime aufgerufen werden.

Mit dieser Methode kann der Tag eines Datumswertes verändert werden. Die Differenz des Wertes wird als (int1) übergeben. Hier sind sowohl positive als auch negative Werte zulässig.


Überschreitet die Differenz eine Monats- oder Jahresgrenze, wird ebenfalls der Monat bzw. das Jahr mit verändert.

Beispiele:

```
cMoment->vpDate # 04.03.2002;cMoment->vmDayModify(-7); // eine Woche zurück = 25.02.2002
```

Mögliche Laufzeitfehler:

_ErrValueRange Das neue Datum liegt außerhalb des gültigen Bereiches von date oder caltime.

vmSecondsModify(int1) 

: time / caltime

Sekunden modifizieren

int1 Differenz

Resultat time / Veränderter
caltime Zeitwert

Siehe Methoden von time,
Methoden von caltime

Die Methode kann für Variablen der Typen time und caltime aufgerufen werden.

Mit dieser Methode kann ein Zeitwert verändert werden. Die Differenz des Wertes wird als (int1) übergeben. Hier sind sowohl positive als auch negative Werte zulässig.


Wird diese Methode für eine Variable vom Typ caltime aufgerufen, wirken sich Überschreitungen von Tagen, Monaten oder Jahren auch auf das Datum aus.

Beispiele:

```
cMoment->vmSystemTime();cMoment->vmSecondsModify(-3600); // Uhrzeit vor einer Stunde
```

Mögliche Laufzeitfehler:

ErrValueRange Die Tagesgrenze wurde überschritten und das neue Datum liegt außerhalb des gültigen Bereiches von caltime.

vmEasterDate(int1) : 
 date / caltime
 Ostersonntag ermitteln
 int1 Jahr

Resultat date / Ostersonntag des
caltime angegebenen
 Jahres

Siehe Methoden von date,
Methoden von caltime

Diese Methode kann für die Datentypen date und caltime aufgerufen werden.

Mit dieser Methode wird der Ostersonntag für das in (int1) angegebene Jahr zurückgegeben. Das Jahr kann im Bereich 1700 bis 2199 angegeben werden.

Alle andere kirchlichen Feiertage beziehen sich auf den Ostersonntag. Mit folgendem Define können die anderen Feiertage berechnet werden:

```
define{ // Konstanten zur Berechnung des entsprechenden Datums mAschermittwoch : -46 mKarfreit
```

Beispiele:

```
cHimmelfahrt->vmEasterDate(2006);cHimmelfahrt->vmDayModify(mHimmelfahrt);
```

handle
Deskriptor

Siehe Datentypen,
ComCall()

Der Begriff handle hat im CONZEPT 16-Editor die folgenden Bedeutungen:

- Datentyp
- Typumwandlung bei COM-Aufrufen

Datentyp

Der Datentyp handle ist ein spezieller Datentyp für Deskriptoren. Variablen vom Typ handle sind vollständig kompatibel zum Datentyp int. Alle entsprechenden Spezifikationen zum Datentyp handle können der Dokumentation unter int entnommen werden.

Durch die Verwendung des eigenen Datentyps für Deskriptoren ist es möglich, eigene sub-Funktionen, die einen Deskriptor als ersten Parameter erwarten, mit dem Pfeiloperator aufzurufen. Aufrufe in Form von

```
Function(tHandle, tA, tB)
```

können also auch im folgenden Format angegeben werden:

```
tHandle->Function(tA, tB)
```

Typumwandlung bei COM-Aufrufen

Mit dem Schlüsselwort handle wird bei der Übergabe von Parametern an Methoden oder Eigenschaften von COM-Objekten ein Deskriptor gekennzeichnet.

Beispiel:

```
tCellStart # tComWorksheet->cphCell(1, 1);tCellEnd    # tComWorkSheet->cphCell(10, 2);tComRange #
```

In diesem Beispiel wird ein Bereich markiert, der anschließend mit Werten gefüllt oder ausgelesen werden kann. Wird in diesem Fall handle weggelassen, würde der Deskriptor falsch interpretiert werden.

color

Farb-Datentyp

Verwandte

Befehle,

Siehe Eigenschaften,

Methoden,

ColorEdit

Der Datentyp color enthält Informationen für die Verwendung von Farben in CONZEPT 16. Eine Variable mit diesem Datentyp verfügt über verschiedene Eigenschaften und Methoden. Eine Variable wird wie folgt deklariert:

```
tColor : color;
```

Der Default-Wert der Variable ist schwarz, alle Eigenschaftswerte werden mit 0 initialisiert.

Eigenschaften

Ist eine Variable mit dem Typ color deklariert, kann ein Farbwert über die verschiedenen Eigenschaften gesetzt oder abgefragt werden.

• vpColorSystem

Die _WinCol...-Konstanten, sowie berechnete Farbwerte können auch mit diesem Datentyp verwendet werden. Die entsprechende Konstante oder der Farbwert werden der Eigenschaft vpColorSystem vom Typ int zugewiesen oder aus dieser Eigenschaft gelesen.

```
local{ tColor : color; tColValue : int;}{ tColor->vpColorSystem # _WinColWhite; tCo
```

• **Farbanteilen**

Die Farbe im Datentyp color wird aus vier Farbkomponenten zusammengesetzt. Die einzelnen Farbanteile können über folgende Eigenschaften gesetzt und abgefragt werden:

- ◆ vpColorR Rot-Anteil der Farbe
- ◆ vpColorG Grün-Anteil der Farbe
- ◆ vpColorB Blau-Anteil der Farbe
- ◆ vpColorA Alpha-Anteil der Farbe

Die Anteile werden als byte-Werte angegeben. Der Alpha-Anteil der Farbe bestimmt die Deckkraft. Der Wert 0 bedeutet deckend, der Wert 255 bedeutet transparent.

```
// Weiß, ohne transparenztColor->vpColorR # 255;tColor->vpColorG # 255;tColor->vpColorB #
```

Die Zugriffe und die Abfrage kann kombiniert werden, um zum Beispiel bestimmte Windows-Farben zu ermitteln.

```
local{ tColor : color; tColValue : int;}{ tColor->vpColorSystem # _WinColActiveWindow; tCo
```

Methoden

Für den Datentyp stehen folgende Methoden zur Verfügung:

- **vmColorSystem()**

Mit dieser Methode wird die Farbe auf den übergebenen Farbwert und die Transparenz gesetzt. Die Methode entspricht der Anweisung ColorMake().

- **vmColorRGBA()**

In den Parametern dieser Methode werden die Werte der einzelnen Farbkanäle und die Transparenz übergeben. Die entsprechenden Eigenschaften werden gesetzt.

In den folgenden Beispielen, werde Farben mit Hilfe der Methoden gesetzt:

```
// Weiß, ohne transparenz tColor->vmColorSystem(_WinColWhite, 0); tColor->vmColorRGBA(255, 255, 255, 0);
```

Eigenschaften von color

Liste aller Eigenschaften von color

Siehe color

- vpColorA
- vpColorB
- vpColorG
- vpColorR
- vpColorSystem

vpColorSystem


Farbwert

Typ int

Verwandte

Siehe Befehle,
color

Mit dieser Eigenschaft kann der Farbwert über eine der _WinCol...-Konstanten gesetzt werden. Durch Setzen der Eigenschaft werden die Eigenschaften vpColorR, vpColorG und vpColorB ebenfalls geändert. Das Setzen einer dieser Eigenschaften ändert auch den Inhalt der Eigenschaft vpColorSystem.

 Da die Deckkraft beim Setzen der Farbe mit ColorMake() ignoriert wird, enthält die durch vpColorSystem ermittelte Farbinformation ebenfalls keine Deckkraft. Wird die Eigenschaft auf einen unzulässigen Farbwert gesetzt, wird der Laufzeitfehler ErrValueInvalid erzeugt.

vpColorR

Farbanteil rot

Typ byte

Verwandte

Siehe Befehle,

color

Über diese Eigenschaft kann der rote Farbanteil einer Farbe gesetzt und abgefragt werden. Die Eigenschaft wird ebenfalls geändert, wenn die Eigenschaft vpColorSystem gesetzt wird.

vpColorG

Farbanteil grün

Typ byte

Verwandte

Siehe Befehle,

color

Über diese Eigenschaft kann der grüne Farbanteil einer Farbe gesetzt und abgefragt werden. Die Eigenschaft wird ebenfalls geändert, wenn die Eigenschaft vpColorSystem gesetzt wird.

vpColorB

Farbanteil blau

Typ byte

Verwandte

Siehe Befehle,

color

Über diese Eigenschaft kann der blaue Farbanteil einer Farbe gesetzt und abgefragt werden. Die Eigenschaft wird ebenfalls geändert, wenn die Eigenschaft vpColorSystem gesetzt wird.

vpColorA

Transparenz der Farbe

Typ byte

Siehe Verwandte
Befehle, color

Über diese Eigenschaft kann die Transparenz (Alpha-Kanal) einer Farbe gesetzt und abgefragt werden. Bei einem Wert von 0 ist die Farbe deckend (nicht transparent), 255 bedeutet transparent. Ein Wert von 128 entspricht somit einer Transparenz von 50%.

Methoden von color

Liste aller Methoden von color

Siehe color

- vmColorRGBA
- vmColorSystem

vmColorSystem(int1, byte2) :



color

Farbe mit Transparenz erzeugen

int1 Farbwert

byte2 Transparenz

Resultat color Farbe mit Transparenz

Siehe Methoden von color,
 ColorMake()

Mit dieser Methode kann aus einer _WinCol...-Farbkonstante eine Farbe mit Transparenz erzeugt werden. Die Farbkonstante wird in (int1), die Transparenz in (byte2) angegeben.

Beispiel:

```
tColor->vmColorSystem(_WinColWhite, 128); // Weiß mit 50% Transparenz
```

Mögliche Laufzeitfehler:

_ErrValueInvalid In (int1) wurde ein ungültiger Farbwert angegeben.

vmColorRGBA(byte1, byte2,
byte3, byte4) : color



Farbe mit Transparenz erzeugen

byte1 Farbanteil rot

byte2 Farbanteil grün

byte3 Farbanteil blau

byte4 Transparenz

Resultat color Farbe mit Transparenz

Siehe Methoden von color,
 ColorRgbMake()

Mit dieser Methode kann aus den verschiedenen Farbkanälen und der Transparenz eine Farbe erzeugt werden. Die Intensität der einzelnen Farben werden in der Reihenfolge Rot, Grün und Blau, gefolgt von der Transparenz angegeben.

Die Methode entspricht der Anweisung ColorRgbMake().

NULL

NULL-Wert

Siehe ComCall()

Variablen und Datenbankfelder in CONZEPT 16 können mit dieser Konstante geleert werden:

Datentyp Wert

<u>alpha</u>	"
<u>byte</u>	0
<u>word</u>	
<u>int</u>	
<u>bigint</u>	
<u>float</u>	0.0
<u>decimal</u>	
<u>decimal</u>	_DecimalUndef
<u>date</u>	0.0.0
<u>time</u>	0:0:0.0
<u>logic</u>	false

Zusammengesetzte Datentypen können ebenfalls auf diese Weise mit einer Anweisung geleert werden. Durch das Leeren wird bei den Datentypen date und decimal der Wert auf "nicht definiert" gesetzt. Bei den anderen Datentypen steht ein gültiger Wert in der Variablen.

Der Bezeichner eines Arrays kann nicht auf den Wert NULL gesetzt werden.

Beispiele:

```
local{ tHdlFrame      : int; tZeichensatz : font; tRechteck      : rect; tFeld          : int[20];}
```



Beim Aufruf von Methoden eines COM-Objekts, die optionale Argumente erwarten, kann mit dem Schlüsselwort NULL ein Argument übersprungen werden.

Array

Array

local, global,

Siehe Verwandte

Befehle,

Datentypen

Eine Variable kann auch als Array deklariert werden. In diesem Fall folgt nach dem Typ die Anzahl der Elemente in eckigen Klammern. Sofern sich die Anzahl der benötigten Elemente erst zu Laufzeit ergibt, kann ein Array auch als dynamisch deklariert werden, indem keine Anzahl angegeben wird.

Beispiel:

```
global common{ gValues      : int[1000];           // 1000 Elemente von int  gNameTab      : alpha(100)
```

Ein einzelnes Array kann nicht größer als 4 MB deklariert werden. Die Deklaration einer Variablen muss in den ersten 2 MB des Bereiches erfolgen, sonst erfolgt die Fehlermeldung Datenlimit überschritten. Möglicherweise müssen große Arrays dynamisch angelegt werden.

Ein dynamisches Array muss zur Laufzeit mit dem Befehl VarAllocate() mit der gewünschten Anzahl von Elementen im Speicher angelegt werden.

Die einzelnen Elemente eines Array werden später durch <namen>[<index>] angesprochen (z. B. gSortTab[22]). Dabei hat das erste Element immer die Nummer 1, die Nummer des letzten Elements entspricht der Anzahl von Elementen im Array.



Mehrdimensionale Arrays können mit eindimensionalen Arrays realisiert werden. Ein Beispiel dazu befindet sich in der FAQ-Sektion der Dokumentation.

Konstanten für einfache Datentypen

Liste der Konstanten für Datentypen, die nicht zusammengesetzt sind

Im folgenden eine Liste der Datentypen, die nicht aus mehreren Komponenten bestehen. Die Datentypen werden von verschiedenen Anweisungen zurückgegeben, die Informationen über die Datenstruktur oder Objekte ermitteln.

- TypeAlpha
- TypeBigInt
- TypeByte
- TypeDate
- TypeDecimal
- TypeFloat
- TypeInt
- TypeLogic
- TypeTime
- TypeWord

Zahlendarstellung

Darstellung von Zahlen im Speicher

Siehe Datentypen

Der Computer arbeitet auf Basis des Binärsystems. Alle Zahlen werden zur Speicherung in ein binäres Zahlenformat umgewandelt. Je nach verwendeter Umwandlung werden Zahlen unterschiedlich in Binärform repräsentiert.

Grundsätzlich kann man die Darstellung von natürlichen Zahlen (ganze Zahlen) und reelle Zahlen (als Festkomma- oder Gleitkommazahlen) unterscheiden.

Ganze Zahlen (byte, word, int / long und bigint)

Die Datentypen byte und word werden ohne Vorzeichen gespeichert. Die Werte werden in Potenzen von 2 aufgeteilt und als einzelne Bits in ein Byte geschrieben. Das Bit mit der kleinsten Signifikanz steht dabei am weitesten rechts. Im Falle des Datentyps word werden zur Speicherung zwei Byte benötigt. Das weniger signifikante Byte steht dabei an der kleineren Adresse im Speicher ("Little endian").

Die Datentypen int / long bzw. bigint werden als 4 Byte bzw. 8 Byte Werte mit Vorzeichen im Speicher abgelegt. Die Speicherung von negativen Zahlen erfolgt dabei im 2er-Komplement. Das höchstwertigste Bit wird dabei als Vorzeichen verwendet.

Beispiele

dezimale Darstellung	Datentyp	binäre Darstellung
10	<u>byte</u>	00001010
10	<u>word</u>	00001010 00000000
10	<u>int</u>	00001010 00000000 00000000 00000000
-10	<u>int</u>	11110110 11111111 11111111 11111111
10	<u>bigint</u>	00001010 00000000 00000000 00000000 00000000 00000000 00000000 00000000

Aufgrund der Darstellung kann es zu Problemen beim Überlauf kommen, wenn zwei Zahlen addiert werden, die ein Übertrag in das höchstwertigste Bit zur Folge haben.

Reale Zahlen (float und decimal)

Gleitkommazahlen (float) werden als 8 Byte-Werte abgelegt. Dabei wird eine Mantisse und ein Exponent gespeichert.

$10.0 = 1.0 \cdot 10^1$ oder in binärer Darstellung $10.0 = 1.010 \cdot 2^{0011}$. Die Mantisse wird normalisiert. Somit ist der Wert der Mantisse immer größer oder gleich 1 und kleiner 2. Die Speicherung der 1 vor dem Komma kann dadurch entfallen. Im Speicher wird die Zahl wie folgt abgelegt:

Vorzeichen	Exponent	Mantisse
63	62-52	51-0
0	0000000 0011	0100 00000000 00000000 00000000 00000000 00000000 00000000

Kontakt

Mit Hilfe des Exponenten wird das Komma innerhalb der Mantisse verschoben. Sollen reale Zahlen dargestellt werden, wird das Komma nicht über die gesamte Mantisse verschoben:

$$10.5 = 1.0101 * 2^{0011}$$

Es bleibt also nach dem Verschieben des Kommas ein gesetztes Bit hinter dem Komma stehen. Dieses Bit repräsentiert den Wert 2^{-1} also 0.5. Durch die Darstellungsform können bestimmte Zahlen, die im Dezimalsystem einfach dargestellt werden können, nicht fehlerfrei in das Binärsystem übertragen werden. Zum Beispiel:

$$10.6 = 1.0101001100110001... * 2^{0011}$$

Wäre die Mantisse mit den hier berechneten Stellen zu Ende, entspräche die Zahl dem Wert 10.5999658203125. Sie ist somit fehlerbehaftet. Eine Rundung kann das unmittelbare Problem beseitigen, wird allerdings mit dieser Zahl eine umfangreiche Berechnung durchgeführt und anschließend gerundet, kann es zu Abweichungen kommen. Dies ist auch der Grund, warum float-Felder nicht als eindeutige Schlüsselfelder verwendet werden dürfen. Werden andere Rundungsstrategien verwendet (zum Beispiel beim Zugriff über andere Programme über die ODBC- oder Programmierschnittstelle), kann der Datensatz nicht mehr gefunden werden (es sei denn, die Datensätze werden sequenziell gelesen).

Bei dem Datentyp decimal wird nicht die gesamte Zahl in die binäre Darstellung überführt, sondern nur die einzelnen Ziffern. Die Speicherung der Ziffern erfolgen in 32 Byte. Die Ziffern werden dabei komprimiert gespeichert, sodass Werte mit 58 Stellen Genauigkeit gespeichert werden können.

Zahl Exponent (zur Basis 10) Ziffern

10.6 0001 0000 0110

Ein Fehler aufgrund der Darstellungsform kann dann nicht mehr auftreten.

Objektreferenz (\$)
Objektreferenzierung

Objektname,

Siehe Suchpfad
für Objekte

Oberflächenobjekte können zur Laufzeit prozedural mit dem \$-Zeichen über ihren Namen referenziert werden. Daher sollte auf eine eindeutige Vergabe von Objektnamen innerhalb des selben Fensters geachtet werden.

Wird ein Objekt über seinen Namen referenziert, wird in dem aktuellen Dialog nach dem Objekt gesucht. Bei Ereignissen wird zunächst von dem Objekt aus gesucht, dass das Ereignis ausgelöst hat, und anschließend in dem enthaltenden Dialog. Die Suche wird beendet, sobald das erste Objekt mit dem entsprechenden Namen gefunden wird.

Wird kein Objekt gefunden, wird 0 zurückgeliefert. Was beispielsweise bei Abfragen von Eigenschaften zum Laufzeitfehler _ErrHdlInvalid und zum anschließenden Abbruch der Prozedur führt.

Das Startobjekt bestimmt, wie lange die Suche dauert und welche Objekte durchsucht werden. Der Suchpfad wird automatisch gesetzt, wenn ein Ereignis ausgelöst oder ein Frame-Objekt geladen wird. Alle Namensreferenzen werden dann innerhalb des Fenster-Objekts aufgelöst.

Mit dem Befehl WinSearchPath() wird das Startobjekt der Suche definiert. Wurde innerhalb eines Ereignisses ein Objekt mit dem Befehl WinOpen() mehrfach geladen, ist der Suchpfad zunächst auf das zuletzt geladene Fenster gesetzt. Da die Namen der Objekte nicht mehr eindeutig sind, muss, um ein Objekt aus dem zuerst geladenen Fenster anzusprechen, der Suchpfad auf dieses Fenster gesetzt werden.



Die Suche wird bei jeder Namensreferenz erneut durchgeführt. Wird ein Deskriptor innerhalb einer Funktion mehrfach verwendet (zum Beispiel in einer Schleife), sollte der Deskriptor in einer Variablen gespeichert und diese als Objektreferenz genutzt werden.

Mit dem with-Konstrukt wird die Existenz der Objekte, die mit \$: referenziert werden, bereits bei der Übersetzung geprüft. Die Namensreferenzen werden bei Verwendung des with-Konstruktes schneller aufgelöst, da beim Übersetzen der Prozedur der Pfad des Objektes gespeichert wird.

Beispiele

```
// Caption des Objekts 'lblName' auf 'Name' setzen$lblName->wpCaption # 'Name';// Caption des Obj
```

Befehle für globale Variablen

Befehle für globale Variablen

Siehe Befehlsgruppen,
Befehlsliste

- VarAllocate
- VarCopy
- VarFree
- VarInfo
- VarInstance
- VarName



VarAllocate(name1[,int2]) : int

Globalen Datenbereich oder dynamisches Array anlegen

name1 Name des globalen Datenbereichs

int2 Anzahl der zu allozierenden Elemente

Resultat int Deskriptor der Datenbereichs-Instanz

Siehe Verwandte Befehle, VarInfo(), VarFree(),
VarInstance(), VarCopy()

Hiermit wird ein globaler Datenbereich (global) oder ein dynamisches Array im Hauptspeicher angelegt. Bei dynamischen Arrays ist darauf zu achten, dass die maximale Größe eines Arrays 4 MB nicht überschreiten darf.

Sofern ein globaler Datenbereich mehrfach angelegt wird, muss der zurückgelieferte Deskriptor gespeichert werden, um ein Umschalten zwischen den verschiedenen Instanzen des Datenbereichs zu ermöglichen (es kann nicht gleichzeitig auf verschiedene Instanzen eines Datenbereichs zugegriffen werden). Durch das Anlegen einer neuen Instanz wird diese sogleich zur aktuellen Instanz.

Beispiele:

```
// Der globale Datenbereich 'Common' wird angelegtVarAllocate(Common);// Das dynamische Array 'Va
```



Die eindeutige Identifikation des Datenbereichs erfolgt über den Namen der Prozedur und den Namen des Datenbereichs. Ein angelegter Datenbereich, kann somit nur in der gleichen Prozedur freigegeben werden, in der der Datenbereich auch angelegt wurde.

VarCopy(name1, int2, int3)



Globalen Datenbereich kopieren

name1 Name des globalen
Datenbereichs

int2 Datenbereichs-Deskriptor
(Quelle)

int3 Datenbereichs-Deskriptor
(Ziel)

Verwandte Befehle,

Siehe VarAllocate(),
VarInstance()

Mit diesem Befehl kann der Inhalt eines globalen Datenbereichs kopiert werden.

Der globale Datenbereich (name1) muss bei Quell- (int2) und Ziel-Datenbereich (int3) identisch sein.



Dynamische Arrays werden **nicht** kopiert. Im Zieldatenbereich vorhandene dynamische Arrays bleiben unverändert.

Beispiel:

```
global MyData{ // Datenbereichsstruktur gID : int; gName : alpha;}main local { tMyDataS
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Deskriptor ungültig



VarFree(name1)

Globalen Datenbereich oder dynamisches Array freigeben.

name1 Name des globalen
Datenbereichs oder
des dynamischen
Arrays

Verwandte Befehle,

VarAllocate(),

Siehe VarInfo(),

VarInstance(),

ProcessMemoryKB

Mit dieser Funktion wird ein angelegter globaler Datenbereich oder ein angelegtes dynamisches Array wieder aus dem Hauptspeicher entfernt.

Bei mehrfachen Instanzen eines Datenbereichs wird die aktuelle Instanz freigegeben. Anschließend ist keine Instanz aktiv. Um auf eine andere Instanz zuzugreifen, muss diese zunächst mit VarInstance() aktiviert werden.


Befindet sich innerhalb eines Datenbereiches ein dynamisches Array, wird das Array zusammen mit dem Datenbereich aus dem Speicher entfernt.

Beispiele:

```
global Common{ gAlpha      : alpha(100); gDynArray : int[]; }...VarAllocate(Common);VarAllocate(gD
```



Die eindeutige Identifikation des Datenbereichs erfolgt über den Namen der Prozedur und den Namen des Datenbereichs. Ein angelegter Datenbereich, kann somit nur in der gleichen Prozedur freigegeben werden, in der der Datenbereich auch angelegt wurde.

VarInfo(name1) : int 

Variableninformation

Name eines globalen
Datenbereichs, eines
name1 Arrays oder einer
alphanumerischen
Variablen

Resultat int Variableninformation

Verwandte Befehle,
Siehe VarAllocate(), VarFree(),
VarInstance(),
VarName()

Mit VarInfo() besteht die Möglichkeit, den Status eines globalen Datenbereichs, eines Arrays oder einer alphanumerischen Variablen zu ermitteln.

Folgende Resultate sind definiert:

globaler Datenbereich

0 - nicht angelegt
> 0 - Deskriptor der Datenbereichs-Instanz

Array

0 - nicht angelegt (bei einem dynamischen Array)
> 0 - Anzahl der Elemente im Array

Bei einer **alphanumerischen Variable** wird die maximale Länge der Variablen zurückgegeben.

Beispiel

```
if (VarInfo(Common) > 0){ VarFree(Common);}
```



VarInstance(alpha1, int2)

Instanz eines globalen Datenbereichs auswählen

alpha1 Name eines globalen
Datenbereichs

int2 Deskriptor der
Datenbereichs-Instanz

Verwandte Befehle,

Siehe VarAllocate(),
VarFree(), VarInfo(),
VarCopy()

Mit dieser Funktion wird eine mit VarAllocate() angelegte Instanz ausgewählt. Alle nachfolgenden Zugriffe auf die Variablen des Datenbereichs beziehen sich dann auf diese Instanz.

Beispiel:

```
tHdlSave # VarInfo(Common);          // aktuelle Instanz sichern
VarInstance(Common, tHdlNew);
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Deskriptor ungültig



obj -> VarName() : alpha

Name des globalen Datenbereichs ermitteln

obj Deskriptor des globalen
Datenbereichs

Resultat alpha Name des Datenbereichs

Siehe Verwandte Befehle, VarInfo()

Der Befehl liefert den Namen des in (obj) übergebenen globalen Datenbereiches zurück. Im Namen ist der jeweilige Prozedurname enthalten.

Wird in der Prozedur "Def.Application" der globale Datenbereich "Common" definiert, liefert der Befehl "Def.Application:Common" zurück.

Beispiel:

```
// Der globale Datenbereich 'Common' wird angelegtVar # VarAllocate(Common);// Name ermitteltNa
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Deskriptor ungültig

Befehle zur Prozedursteuerung

Liste der Befehle zur Prozedursteuerung

Siehe Befehlsgruppen,
Befehlsliste

Befehle

- break
- Call
- CallOld
- cycle
- do...while
- exit
- for...loop...while/until
- if...else
- main
- return
- RmtCall
- sub
- switch...case...default
- try
- trysub
- while
- {...}

{...}

Anweisungs- / Deklarations- / Definitionsblock

Siehe [Verwandte](#)
[Befehle](#)

Syntax:

```
{ [[ <Anweisung> ]] }
```

Mit den Zeichen { und } werden mehrere Anweisungen, Definitionen oder Deklarationen zu einem Block zusammengefasst.

Beispiele:

```
define{ mPI : 3.14159265 ...}local{ tNumber : int; tSum : int; ...}if (tNumber > 400){ i
```



Im PASCAL-Style werden Anweisungsblöcke mit BEGIN / END markiert. Weitere Informationen befinden sich im Abschnitt [Style-Unterschiede](#).

break



Schleifenabbruch

Verwandte

Siehe Befehle,

cycle

Innerhalb einer Schleife (for...loop...while/until, do...while, while) kann mit dieser Anweisung die Schleife beendet werden.

Nach break wird die erste Anweisung hinter der Schleife durchgeführt. break wird meistens benutzt, um komplexe Schleifen einfach zu verlassen, ohne dass die Schleifenbedingung extrem lang wird.

cycle



Schleife wiederholen

Verwandte

Siehe Befehle,



break

Innerhalb einer Schleife (for...loop...while/until, do...while, while) können mit dieser Anweisung die restlichen Anweisungen in der Schleife übersprungen werden. Die Schleife wird hierbei am Anfang fortgesetzt.



In einer do...while-Schleife wird die while-Bedingung nicht geprüft.

Die Schleife wird dabei nicht verlassen, sondern erneut durchgeführt. cycle wird meistens benutzt, um innerhalb einer Schleife eine extreme Schachtelung von if...else-Anweisungen zu vermeiden.

Call(alpha1[, ...]) : var 
 Prozedurfunktion aufrufen
 alpha1 Prozedurname
 ... Funktionsargumente
 Resultat var Rückgabewert der 
 Prozedurfunktion

Siehe [Verwandte Befehle](#), [CallOld\(\)](#),
[RmtCall\(\)](#)

Mit dieser Funktion kann eine variable Prozedurfunktion oder Prozedur aufgerufen werden. (alpha1) bezeichnet den Namen der Funktion, wobei die Notation <Prozedurname:Funktionsname> zu verwenden ist. Wird der Funktionsname weggelassen, findet ein Aufruf der main-Funktion statt. Es können bis zu 64 weitere Argumente für die Funktion übergeben werden. Die Richtigkeit der Argumenttypen wird erst zur Laufzeit überprüft.

Steht der Name der aufgerufenen Funktion zum Übersetzungszeitpunkt fest, kann der Name der Prozedur und der Funktion direkt im Sourcecode in der Form <Prozedurname:Funktionsname> angegeben werden. Bei der Übersetzung kann dann auch die Existenz der Prozedur / Funktion überprüft werden. Folgende Anweisungen sind identisch:

```
Call('MyProc:MyFunc');MyProc:MyFunc();
```

Das Resultat der aufgerufenen Funktion wird von dem Befehl zurückgegeben, sodass der Befehl Call() auch in Ausdrücken verwendet werden kann. Der Typ des Rückgabewertes wird erst zur Laufzeit überprüft.

Beispiel:

```
FuncName # 'Sum:Calc_' + CnvAI(CalcTyp, 0);tResult # Call(FuncName, Val1, Val2);
```

Falls der Typ des Ausdrucks variabel oder nicht eindeutig bestimmt werden kann, muss eine Typdeklaration in der Form "typ(Call(...))" verwendet werden.

Beispiel:

```
FsiWrite(tHdl, int(Call(xy)));if (alpha(Call(a1)) > 'xyz') ...
```

Stimmt zur Laufzeit der Typ nicht überein oder liefert die aufgerufene Funktion keinen Wert, erfolgt ein Laufzeitfehler.

Der Aufruf von Prozedurfunktionen kann auch wie folgt realisiert werden:

```
// Diff in derselben Prozedur oder main in der Prozedur Diff aufrufenDiff(SaveVal, NewVal);// Tes
```

Die Verarbeitung bleibt auf der Maschine, die diesen Befehl ausführt. Wird also in einer Prozedur, die mit RmtCall() aufgerufen wurde, die Anweisung Call() durchgeführt, wird die so aufgerufene Prozedur ebenfalls auf dem Server ausgeführt.

Mögliche Laufzeitfehler:

Kontakt

<u>ErrStringOverflow</u>	Prozedur- oder Funktionsname zu lang (alpha1)
<u>ErrFldType</u>	Rückgabewert hat nicht den korrekten Typ
<u>ErrNoProcInfo</u>	Prozedur ist nicht vorhanden oder wurde nicht übersetzt
<u>ErrNoSub</u>	Prozedurfunktion ist nicht vorhanden
<u>ErrArgumentsDiff</u>	Anzahl der Prozedurfunktionsargumente abweichend

CallOld(alpha1[,var2[,...var9]])



Aufruf einer A- Prozedur mit Rückgabewert

alpha1 Name der A-
Prozedur

... Funktionsargumente

Siehe Verwandte Befehle,
Call(), RmtCall()



Die mit diesem Befehl aufgerufene A- Prozedur kann nur vom CONZEPT 16-Standard-Client verarbeitet werden. Aufrufe einer A- Prozedur innerhalb der Verarbeitung vom CONZEPT 16-Server, Programmierschnittstelle etc. führen zu dem Laufzeitfehler ErrCallOld. Soll in diesen Umgebungen kein Laufzeitfehler generiert werden, kann er mit ErrIgnore() übersprungen werden.

Mit dieser Systemprozedur wird aus der aktuellen Prozedur heraus die in (alpha1) angegebene A- Prozedur aufgerufen. Nach Beendigung der Prozedur kehrt die Verarbeitung in die aktuelle Prozedur zurück.

Beispiel:

```
CallOld('Sum1') // Ruft die Prozedur SUM1 auf
```

An die aufgerufene Prozedur können bis zu 8 Werte beliebigen Typs übergeben werden. Die Typprüfung findet bei der Ausführung der Prozedur statt. In der aufgerufenen Prozedur müssen die übergebenen Werte ("Argumente") entsprechend deklariert sein.

Beispiel:

```
// Sum2 mit dem Wert 45 und dem Inhalt des Felds IntRes aufrufenCallOld('Sum2', 45, IntRes);
```

Bei übergebenen Feldern oder Variablen kann die Bezeichnung var vorangestellt werden. Dies bedeutet, dass zunächst die in den Feldern oder Variablen enthaltenen Werte in die Variablen der aufgerufenen Prozedur übertragen werden. In der aufgerufenen Prozedur können diese Werte dann verändert werden. Nach der Rückkehr in die ursprüngliche Prozedur werden diese Werte dann wieder in die entsprechenden Felder oder Variablen übertragen (call-by-reference). Dies ist sinnvoll, wenn eine Prozedur ein Ergebnis an die aufrufende Prozedur zurückliefern soll.

Beispiel:

```
CallOld('VMS', 45, var Ct.Number);
```

Alle acht Argumente bei CallOld() können mit var gekennzeichnet werden. Das Attribut var kann nicht bei konstanten Werten, Systemfunktionen oder Ausdrücken benutzt werden.



do...while

Schleife mit Austrittsbedingung

Verwandte Befehle,
for...loop...while/until,

Siehe while,

Vergleichsoperatoren,
break, cycle

Syntax:

```
do <Anweisung> [[ ; <Anweisung> ]] while (<Ausdruck>);
```

Im Gegensatz zur Schleife mit Eintrittsbedingung wird bei dieser Schleifenform die Anweisung mindestens einmal durchgeführt und solange wiederholt, wie das Resultat des Ausdrucks true ist.

Um innerhalb einer Schleife eine extreme Schachtelung von if...else-Anweisungen zu vermeiden, können mit der Funktion cycle die restlichen Anweisungen in der Schleife übersprungen und die Schleife am Anfang fortgesetzt werden. Mit der Funktion break wird die Schleife beendet.



Bei Verwendung von cycle wird zum Anfang der Schleife gesprungen. Die Bedingung im while wird hierbei nicht geprüft.

Beispiel:

```
do CheckAuftrag();while (RecRead(3, 2, _RecNext) = _r0k);
```

In diesem Beispiel werden die durch do...while eingeschlossenen Anweisungen so oft wiederholt, bis RecRead() ein anderes Ergebnis als _rOk liefert.



Bei der Verwendung des PASCAL-Styles muss eine andere Syntax beachtet werden. Weitere Informationen befinden sich im Abschnitt Style-Unterschiede.



exit

Direktes Beenden aller Prozeduren

Verwandte

Siehe Befehle,

return

Mittels exit wird die komplette Verarbeitung von Prozeduren abgebrochen. Während return nur die aktuelle Prozedurfunktion verläßt, beendet exit alle laufenden Prozeduren. Dies ist bei verschachtelten Prozeduren sinnvoll, um Fehlerbehandlungen zu vereinfachen.

for...loop...while/until



Schleife mit Initialisierung

Verwandte Befehle,

Siehe do...while, while,

Vergleichsoperatoren,

break, cycle

Syntax:

```
for <Anweisung> loop <Anweisung> while <Ausdruck> <Anweisung>
```

oder

```
for <Anweisung> loop <Anweisung> until <Ausdruck> <Anweisung>
```

Dieser Schleifentyp verfügt über drei verschiedene Elemente:

- Initialisierung for ...
- Wiederholungsanweisung loop ...
- Bedingung while ... oder until ...

Zuerst wird die Initialisierungsanweisung durchgeführt. Danach wird zunächst die Schleifenbedingung geprüft, bevor die Schleife das erste Mal durchlaufen wird. Nach dem Durchlauf wird die Wiederholungsanweisung durchgeführt und anschließend die Schleifenbedingung erneut überprüft.

Als Schleifenbedingung kann sowohl until (Schleife läuft bis die Bedingung erfüllt ist) als auch while (Schleife läuft solange die Bedingung erfüllt ist) verwendet werden. In beiden Fällen handelt es sich um eine Schleifen-Eintrittsbedingung.

Die for-Schleife eignet sich besonders für komplexe Schleifen, da als Initialisierungs- bzw. Wiederholungsanweisung auch Anweisungsblöcke möglich sind.

Um innerhalb einer Schleife eine extreme Schachtelung von if...else-Anweisungen zu vermeiden, können mit dem Befehl cycle die restlichen Anweisungen in der Schleife übersprungen werden. Mit dem Befehl break wird die Schleife beendet.

Beispiele:

```
for tCount # 0 loop Inc(tCount)while (tCount < 10000) {
```

```
... }
```



Bei der Verwendung des PASCAL-Styles muss eine andere Syntax beachtet werden. Weitere Informationen befinden sich im Abschnitt Style-Unterschiede.

if...else



Verzweigung

Verwandte Befehle,logic,Siehe switch...case...default,Vergleichsoperatoren,Logische Operatoren**Syntax:**

```
if (<Ausdruck>) <Anweisung> [ else <Anweisung> ]
```

Bei der Verzweigung wird eine Anweisung nur durchgeführt, wenn der angegebene Ausdruck ein logisches Ja-Resultat liefert.

Beispiel:

```
if (fSumme > 1000) return('Wert zu groß');...
```

In diesem Fall wird die return-Anweisung nur dann ausgeführt, wenn der Ausdruck "(Summe > 1000)" das Resultat Ja (true) liefert. Im negativen Fall (false) wird die Anweisung ignoriert. Sofern im negativen Fall der optionale else-Teil der Verzweigung vorhanden ist, wird die Anweisung hinter else durchgeführt.

Beispiel:

```
if (fSumme = 3) return('Summe gleich 3');else return('Summe ungleich 3');...
```

Bei einer Verzweigung wird also in Abhängigkeit vom Resultat des Ausdrucks entweder die bedingte Anweisung oder (falls vorhanden) die else-Anweisung durchgeführt.

Mehrere logische Ausdrücke können mit den entsprechenden logischen Operatoren verknüpft werden. Die Ausdrücke werden dabei nur so weit ausgewertet, bis das Ergebnis feststeht.

Beispiel:

```
if ((tHdl > 0) and (tHdl->wpCaption = ''))
```

Ist in der Variablen tHdl ein Wert größer 0 gespeichert, wird die Caption verglichen. Ein Deskriptor von 0 oder kleiner führt in diesem Fall nicht zu einem Fehler _ErrHdlInvalid, da der zweite Ausdruck nicht ausgewertet wird.



Bei der Verwendung des PASCAL-Styles muss eine andere Syntax beachtet werden. Weitere Informationen befinden sich im Abschnitt Style-Unterschiede.



main

Deklaration der Prozedur-Hauptfunktion

Verwandte

Siehe Befehle,

sub

Der Unterschied zwischen main und sub besteht darin, dass nur eine main-Funktion pro Prozedur deklariert werden kann und main keinen Funktionsnamen besitzt.

Bei einem Aufruf der Prozedur ohne zusätzlichen Funktionsnamen wird main aufgerufen.

In der main-Funktion können genauso wie in den sub-Funktionen Übergabe- und Rückgabewerte definiert werden.

Beispiele:

```
// Hauptfunktion ohne Übergabe- und Rückgabewert  
main{}  
// Hauptfunktion mit Übergabe- und Rückgabe
```

return



Prozedurfunktion beenden

Verwandte

Siehe Befehle,

main, sub

Das Schlüsselwort return beendet die aktuelle Prozedurfunktion, ohne dass noch nachfolgende Anweisungen verarbeitet werden. Sofern die aktuelle Prozedurfunktion aus einer anderen Funktion heraus aufgerufen wurde, wird die Verarbeitung in der ursprünglichen Funktion fortgesetzt.

Sofern für die Prozedurfunktion, in der return benutzt wird, ein Rückgabebetyp deklariert ist, muss nach return ein Ausdruck des gleichen Typs folgen. Dadurch wird der Rückgabewert definiert.



Der Rückgabewert sollte in Klammern nach dem return angegeben werden, speziell wenn der Rückgabewert aus dem Resultat einer Funktion besteht. Das Finden von Programmierfehlern wird damit erleichtert.

Folgende Funktionen lassen sich beide fehlerfrei übersetzen:

Beispiele:

```
sub myRecRead( aFile : int; aKey : int; aFlags : int;) : int;{ return RecRead(aFile, aKey,
```

Der Fehler in der zweiten Funktion besteht darin, dass kein Rückgabewert definiert wurde. Die Zeile wird als zwei Anweisungen interpretiert. Da die RecRead()-Anweisung hinter der return-Anweisung steht, wird nie ein Datensatz gelesen.

Die Funktion sollte wie folgt definiert werden:

```
sub myRecRead( aFile : int; aKey : int; aFlags : int;) : int;{ return(RecRead(aFile, aKey,
```

Sollte in diesem Fall kein Rückgabewert definiert sein, wird das bereits bei der Übersetzung der Funktion festgestellt und ein entsprechender Fehler wird ausgegeben.



RmtCall(alpha1[,...]) : int

Prozedurfunktion auf dem Server aufrufen

alpha1 Name der Prozedurfunktion

... Funktionsargumente

ID des neuen Benutzers oder Fehlercode

ErrLimitExceeded Das Limit für RmtCall-Aufrufe wurde erreicht

ErrOutOfMemory Nicht genügend Arbeitsspeicher zur Durchführung des Befehls vorhanden

Resultat int ErrDbidUserLimit Benutzerlimit des Servers erreicht oder Cachegröße der Zieldatenbank für Benutzermenge unzureichend



ErrDbaDataLocked Die Datenbank ist in exklusiver Benutzung


Siehe Verwandte Befehle, Call(), CallOld(),
Verwendung von Remote-Prozeduren (Blog)

Mit dieser Funktion kann eine variable Prozedurfunktion aufgerufen werden, die dann vom CONZEPT 16-Server verarbeitet wird. (alpha1) bezeichnet den Namen der Funktion. Daneben können bis zu 23 weitere Argumente für die Funktion übergeben werden. Die Richtigkeit der Argumenttypen wird erst zur Laufzeit überprüft. Die Verarbeitung der Funktion erfolgt asynchron, dies bedeutet das Programm wird nach dem Aufruf der Funktion durch RmtCall() sofort fortgeführt ohne auf das Beenden der aufgerufenen Funktion zu warten.

Eine Übergabe von Referenz-Variablen durch Angabe von var ist nicht möglich.

Für die Ausführung der Prozedur auf dem Server wird ein neuer Benutzer erzeugt. Dieser Benutzer verfügt über eigene Feldpuffer. Die Prozedur läuft folglich völlig unabhängig vom Client. Kommt es während der Verarbeitung der Prozedur zu einem Laufzeitfehler, wird dieser in der Log-Datei der Datenbank abgelegt (siehe Log-Einträge).

 Es können nur A+ Prozeduren vom Server verarbeitet werden. Innerhalb der aufgerufenen Prozedur wird der Befehl CallOld() ignoriert. Es können nur Befehle verwendet werden, die mit dem Symbol  gekennzeichnet sind.

 Wird vor dem Aufruf von RmtCall die Anzahl der bereits benutzen RmtCall-Aufrufe mit Dbainfo(_DbarMtprocCount) mit dem Prozedurlimit Dbainfo(_DbarMtprocLimit) verglichen, kann trotzdem der Fehlercode ErrLimitExceeded zurückgegeben werden, auch wenn das Limit zuvor nicht erreicht war. In diesem Fall wurde in der Zwischenzeit durch einen anderen Client ein RmtCall gestartet.

Wichtiger Hinweis



Per RmtCall() gestartete Prozeduren werden innerhalb des Datenbankprozesses durchgeführt. Dabei wird keine TCP/IP-Verbindung mehr benötigt, der Datenzugriff ist daher schneller als bei einem Clientprozess. Allerdings kann die Prozedurausführung die Leistung und im ungünstigen Fall auch die Stabilität des Datenbankprozesses negativ beeinflussen. Ein übermäßiger Ressourcenverbrauch oder auch Programmierfehler können zu Performance-Problemen führen oder das Schließen der Datenbank verhindern. Zudem hat der Client nach dem Start der Prozedur keine direkten Kontrollmöglichkeiten mehr. Ein weiterer Nachteil von RmtCall() liegt im reduzierten Befehlsumfang, etliche Prozedurbefehle sind innerhalb des Datenbankservers nicht verfügbar.

Ein socketbasierter SOA-Service kann auf derselben Maschine wie der Datenbankserver laufen und vermeidet die Nachteile von RmtCall(). Datensatzoperationen sind zwar etwas langsamer, dafür sind aber mehr Befehle ausführbar (beispielsweise für Druckfunktionen). Insbesondere durch die Verwendung der Job-Funktionen innerhalb der SOA-Anwendung kann ein Leistungsspektrum erreicht werden, das innerhalb des Datenbankprozesses nicht realisierbar ist.

Da Selektionen mit der Option SelServer direkt vom Server durchgeführt werden, sollte RmtCall() am besten nur dann verwendet werden, wenn es bei einer umfangreichen Datensatzverarbeitung auf maximale Performance ankommt, sonst ist der socketbasierte SOA-Service immer vorzuziehen!



sub

Deklaration einer Prozedur-Unterfunktion

Verwandte

Befehle,

Siehe main,

declare,

return

Innerhalb einer Prozedur können bis zu 65.500 einzelne Funktionen definiert werden, die sowohl innerhalb der Prozedur als auch von anderen Prozeduren aufgerufen werden können.

Funktionsname

Nach dem Schlüsselwort sub wird der Name der Funktion angegeben. Der Name darf maximal 40 Zeichen lang sein und muss mit einem Buchstaben oder "_" beginnen.

Argumente

Bei einer Funktion können bis zu 64 Argumente deklariert werden. Im Gegensatz zur main-Funktion muss bei der Deklaration der Name der Funktion angegeben werden.

```
sub <name>( <name> : <typ> [[ ; <name> : <typ> ]])
```

Die Deklaration der einzelnen Argumente erfolgt analog zu der Deklaration von Variablen (global).

Dabei werden die Argumente per Wert übergeben, das heißt im jeweiligen Argument steht eine Kopie des übergebenen Wertes (call-by-value). Dabei können keine kompletten Arrays als Wert übergeben werden.

Beispiel:

```
sub Validate( aFieldVal      : int; aDifference  : int;)
```

Alternativ können die Argumente auch als Referenz deklariert werden (call-by-reference). Dabei wird dem Argumentnamen das Schlüsselwort var vorangestellt. In diesem Fall können auch Arrays übergeben werden. Die Übergabe als Referenz bedeutet, dass das Argument als Bezeichnung der übergebenen Variablen benutzt wird. Eine Veränderung des Argumentwertes hat somit eine gleichzeitige Veränderung der übergebenen Variablen zur Folge, da das Argument lediglich eine Referenz darstellt.

Bei Referenzargumenten können demnach keine konstanten Werte übergeben werden. Zulässig sind ausschließlich Variablen, Elemente von Arrays, Arrays und Felder der Datenstruktur. Dabei muss beachtet werden, dass aus dem Argument die ursprüngliche Art (zum Beispiel Feld oder Variable) der Referenz nicht mehr hervorgeht.

Bei Referenzen dürfen bei einem Übergabewert vom Typ alpha keine Längenangaben stehen und bei einem Array keine Elementanzahl, da diese Werte sich implizit aus den übergebenen Variablen ergeben.

Argumente können auch optional deklariert werden. Dies geschieht durch das vorangestellte Schlüsselwort `opt`.

Beispiel:

```
sub Validate( aFieldVal      : int; opt aDifference : int;      )
```

Nach dem ersten `opt`-Argument müssen alle nachfolgenden Argumente ebenfalls mit `opt` deklariert werden. `opt` und `var` können nicht kombiniert werden. Nicht übergebene Argumente werden in der Funktion mit NULL initialisiert. Es ist ferner zulässig, alle Argumente mit `opt` zu deklarieren.

Der Aufruf einer Funktion mit optionalen Argumente ist auch mit Call() möglich.

Beispiel:

```
sub Validate( var aFieldVal      : int; var aValTab      : byte[]; var aError      : alph
```

Rückgabewert

Eine Funktion kann mit einem Rückgabetyt deklariert werden. Durch Definition eines Rückgabetyps kann die Funktion bei Zuweisungen, innerhalb eines Ausdrucks oder als Funktionsargument benutzt werden. Dabei wird der Typ des Rückgabewerts mit `'<typ>'` nach dem Namen bzw. der Argumentliste der Funktion angegeben.

Beispiel:

```
sub Difference ( aValOrg, aValNew : int; ) : int;
```

Der Rückgabewert muss mit der return-Anweisung explizit angegeben werden. Dabei sind beliebig viele return-Anweisungen innerhalb der Funktion möglich. Sofern kein return ausgeführt wird, wird ein leerer Wert (0) zurückgegeben. Es ist zu beachten, dass mit return die Funktion an der entsprechenden Stelle beendet wird.

Beispiel:

```
sub Difference ( aValOrg, aValNew : int; ) : int;{ if (aValOrg > aValNew) return(aValOrg - aValNew)
```

Aufruf von sub-Funktionen

Der Aufruf einer Funktion geschieht durch Angabe des Funktionsnamens, gefolgt von der in Klammern eingeschlossenen Liste der Argumente. Falls das erste Argument einer sub-Funktion vom Typ handle ist, kann das Argument dem Funktionsaufruf mit dem Pfeiloperator `->` vorangestellt werden. Alternativ zu `Function(aHandle, a, b)` kann also auch `aHandle->Function(a,b)` verwendet werden.

Besitzt die Funktion keine Argumente, wird eine leere Liste angegeben. Beim Aufruf einer sub-Funktion in einer anderen Prozedur muss der Prozedurname dem Funktionsnamen vorangestellt werden.

Beispiele:

```
Diff(SaveVal, NewVal);    // ruft die Funktion Diff in derselben
```

```
// Pr
```

Hieraus ist ersichtlich, dass die main-Funktion einer anderen Prozedur nur dann aufgerufen werden kann, wenn keine sub-Funktion innerhalb der aufrufenden Prozedur den gleichen Namen hat, wie die aufgerufene Prozedur. Da die übergebenen Argumente an eine Funktion bei der Übersetzung überprüft werden, müssen die benutzten Funktionen aus anderen Prozeduren bereits in kompilierter Form vorliegen.

Sofern die Funktion einen Rückgabewert besitzt, kann dieser in einer Zuweisung, einem Ausdruck oder als Funktionsargument benutzt werden. Funktionen mit Rückgabewert können auch alleinstehend aufgerufen werden, in diesem Fall wird der Rückgabewert ignoriert.

Beispiele:

```
LocDiff # Diff(SaveVal, NewVal);if (Diff(SaveVal, NewVal) > 100) ...Kd.Check(Diff(SaveVal, NewVal)
```

Sind bei der aufgerufenen Funktion Argumente als Referenz deklariert, so müssen die übergebenen Variablen ebenfalls mit `var` deklariert werden.

Beispiel:

```
sub Validate ( var aFieldVal : int; )...Validate(var Af.Sum)Validate(var ValueTab[10])
```

sub-Funktionen können ebenfalls aus 4.0-kompatiblen Prozeduren aufgerufen werden. Unabhängig vom verwendeten Verfahren für die Parameterübergabe (call-by-reference oder call-by-value) werden die Parameter ohne `var` deklariert.

Zum Aufruf der Funktion wird beim Befehl Call() der Name der Prozedur und der Name der Funktion durch einen ':' getrennt angegeben. Bei der Angabe der Parameter werden die Parameter mit einem var gekennzeichnet, die als Referenz übergeben werden sollen.

Beispiel:

```
Call('Kd.Check:TestNumber'); // ruft die Funktion TestNumber in der
```

switch...case  default
Fallabfrage

Verwandte

Befehle,

if...else,

Siehe Beispiel -

switch /

EvtMouseItem /

EvtLstDataInit

Syntax:

```
switch (<Ausdruck>){ case <Ausdruck> [[,<Ausdruck>]] : <Anweisung> [[; case <Ausdruck> [[,<Ausdruck>]] : <Anweisung> [[; case <Ausdruck> [[,<Ausdruck>]] : <Anweisung> [[; case <Ausdruck> [[,<Ausdruck>]] : <Anweisung> [[; default : <Anweisung> ; }
```

Die Fallabfrage weist mehrere Verzweigungen auf. In Abhängigkeit eines Wertes (Ausdruck) wird eine von mehreren Anweisungen durchgeführt. Vor jeder dieser Anweisungen steht ein Ausdruck oder mehrere Ausdrücke. Ist dieser Ausdruck bzw. einer der Ausdrücke mit dem bestimmenden Wert identisch, so wird die nachfolgende Anweisung durchgeführt. Der bestimmende Wert kann von jedem Typ sein. Alle Ausdrücke vor den Anweisungen müssen vom gleichen Typ sein.

Beispiel:

```
switch (iCmdTyp){ case 1, 2 : iResult # 0; case 3 : iResult # 1; case 4 : iResult # 2; default : iResult # 0; }
```

In diesem Beispiel wird je nach Wert der Variablen 'iCmdTyp' der Wert der Variablen 'iResult' entsprechend definiert. Stimmt der Wert mit keinem der angegebenen Ausdrücke überein, wird die Anweisung nach default durchgeführt.

default muss in der switch-Anweisung nicht vorhanden sein. Ist default nicht vorhanden und stimmt keiner der Ausdrücke mit dem Bestimmungswert überein, so wird auch keine Anweisung durchgeführt. Der Effekt ist der gleiche, wenn default ohne eine nachfolgende Anweisung vorhanden ist.

Eine Besonderheit der switch-Anweisung besteht in der Verwendung von Ausdrücken bei den einzelnen Anweisungen, die auch Felder oder Systemvariablen enthalten können. Damit kann allerdings auch der gleiche Wert mehrfach in der switch-Anweisung enthalten sein. In diesem Fall wird nur die Anweisung beim ersten gleichen Wert durchgeführt.



Bei der Verwendung des PASCAL-Styles muss eine andere Syntax beachtet werden. Weitere Informationen befinden sich im Abschnitt Style-Unterschiede.



try

Beginn der Ausnahmebehandlung

VerwandteBefehle, trysub,ErrGet(), ErrPos(),ErrSet(),Siehe ErrTryIgnore(),ErrTryCatch(),ErrIgnore(),ErrCall(),Fehlerbehandlung(Blog)

Mit der Anweisung für Ausnahmebehandlung kann bei einer Anweisungsfolge die Behandlung von Fehlern von den eigentlichen Anweisungen getrennt werden. Bei vielen Anweisungen wird nach der Anweisung geprüft, ob ein zulässiges Resultat vorhanden ist. In Abhängigkeit davon werden dann entweder weitere Anweisungen durchgeführt oder das unzulässige Resultat wird in irgendeiner Form behandelt. Diese Vorgehensweise kann Anweisungsfolgen stark verkomplizieren und auch unübersichtlicher machen.

Durch die Verwendung der Ausnahmebehandlung in Form des sogenannten try-Blocks kann dies vermieden werden. Dabei führen Verarbeitungsfehler zum Verlassen des try-Blocks. Der jeweilige Verarbeitungsfehler kann dann nach dem try-Block behandelt werden.

Zur Erkennung solcher Fehler existiert eine globale Fehlervariable, die entweder durch den Entwickler oder durch eine Systemfunktion gesetzt wird. Es existieren somit drei verschiedene Arten von Fehlern:

Applikationsfehler

Diese Fehler werden explizit durch den Programmierer definiert und mit dem Befehl ErrSet() gesetzt.

Funktionsfehler

Diese Fehler sind das Resultat einer Systemfunktion. Systemfunktionen, die den globalen Fehlerwert beeinflussen, sind am Ende dieses Abschnitts aufgelistet.

Laufzeitfehler

Dies sind Fehler, die während der Verarbeitung der Prozedur auftreten und normalerweise zum Abbruch der Prozedur führen (zum Beispiel "Division durch Null" oder "Deskriptor ungültig").

Der Fehlerwert kann mit dem Befehl ErrGet() abgefragt werden. Dabei sind positive Werte für die Resultate von Datensatzoperationen reserviert; der Wert 0 bezeichnet keinen Fehler; negative Werte von -1 bis -10000 sind für Systemfehler reserviert; negative Werte ab -10000 können durch den Entwickler für eigene Fehlerwerte benutzt werden.

Wird innerhalb eines try-Blocks eine Funktion aufgerufen, wirkt sich das Setzen des globalen Fehlerwertes auf die Durchführung der aufrufenden Funktion aus. Werden keine weiteren Maßnahmen ergriffen, wird die aufgerufene Funktion komplett durchgeführt. Hat am Ende der Funktion der globale Fehlerwert einen Wert ungleich 0 (ErrOk), wird unmittelbar nach der Rückkehr der Funktion (noch vor der Zuweisung evtl. vorhandener Rückgabewerte) der try-Block verlassen.

Soll in der aufgerufenen Funktion ebenfalls eine Fehlerverarbeitung stattfinden, muss hier auch ein try-Block vorhanden sein. Soll sich der globale Fehlerwert nicht auf die aufrufende Funktion auswirken, muss der Fehlerwert mit ErrSet() auf den Wert ErrOk gesetzt werden. Auch am Ende der Fehlerbehandlung sollte im Regelfall der Fehlerwert geleert werden.

Beispiel:

```
try{ RecRead(1, 1, _RecLock);    // Kunde lesen und sperren  RecLink(1, 2, 1, _RecLock) // Aufruf
```

In diesem Beispiel kann RecRead() bzw. RecLink() anstatt rOk andere Resultate zurückliefern. Anstatt diese jetzt in der Anweisungsfolge zu überprüfen, wird bei einem solchen Resultat der try-Block verlassen. Danach wird mit ErrGet() geprüft, ob ein Fehler aufgetreten ist. Die globale Fehlervariable wird am Anfang des try-Blocks automatisch auf 0 (kein Fehler) gesetzt.

Innerhalb des try-Blocks reicht ein ErrSet() mit einem Wert ungleich 0 aus, um den Block zu verlassen. Dies gilt insbesondere für den Aufruf von eigenen Funktionen innerhalb des try-Blocks. Sofern innerhalb einer aufgerufenen Funktion ein ErrSet() erfolgt, wird der try-Block nach der Rückkehr der Funktion verlassen.

try-Blöcke können nicht geschachtelt werden. Eine Schachtelung ist nur insofern möglich, dass eine aufgerufene Funktion ebenfalls einen try-Block beinhalten kann, dann mit einer eigenen Fehlerbehandlung.

Wird innerhalb eines try-Blocks eine Funktion aufgerufen und in dieser Funktion kommt es zu einem Verarbeitungsfehler, wird die aufgerufene Funktion bis zum Ende durchgeführt. Ist zum Zeitpunkt der Rückkehr zur aufrufenden Funktion der globale Fehler noch gesetzt, wird nach der Rückkehr der Funktion die Verarbeitung nach dem try-Block fortgesetzt.

Soll die Verarbeitung sofort unterbrochen werden, wenn der globale Fehlerwert in einer aufgerufenen Funktion gesetzt wird, muss der trysub-Block verwendet werden.

Direkte Fehlerbehandlung

In manchen Fällen muss allerdings ein Fehler direkt bei der Anweisung behandelt werden, da der Fehler Bestandteil der normalen Verarbeitung ist und nach der Behandlung die Verarbeitung der Anweisungsfolge fortgesetzt wird. Zu diesem Zweck darf bei bestimmten Fehlerwerten der try-Block nicht verlassen werden (dies gilt vor allem bei Resultaten von Satzoperationen).

Mit dem Befehl ErrTryIgnore() kann veranlasst werden, dass der try-Block bestimmte Fehlerwerte ignoriert. Der Anweisung kann entweder ein einzelner Fehlerwert oder

ein Bereich von Fehlerwerten übergeben werden. Dabei wird bei jedem Aufruf von ErrTryIgnore() der zu ignorierende Fehlerwert neu gesetzt. Bei mehreren Aufrufen summieren sich die Fehlerwerte demnach nicht.

Beispiel:

```
ErrTryIgnore(_rLocked)          // Resultat _rLocked wird ignoriertErrTryIgnore(_rLocked, _rNoRec)
```

Die Anweisung ErrTryIgnore() muss immer innerhalb eines try-Blocks stehen.

Beispiel:

```
try{ ErrTryIgnore(_rLocked, _rNoRec); if (RecRead(1, 3, 0) != _rOk)    // Resultat wird hier ve
```

Fehlerposition

Bei größeren try-Blöcken reicht der Fehlerwert vielfach zur anschließenden Behandlung nicht aus, da derselbe Fehlerwert an unterschiedlichen Stellen innerhalb des try-Blocks auftreten kann. Zur Feststellung der Fehlerposition können innerhalb des try-Blocks entsprechende Markierungen, sogenannte "Labels", verwendet werden. Ein Label beginnt mit einem Doppelpunkt, gefolgt von einem Namen und bezeichnet einen bestimmten Abschnitt im try-Block, nämlich von der Anweisung nach dem Label bis zum nächsten Label.

In der Fehlerbehandlung kann dann mit der Funktion ErrPos() die Position des aufgetretenen Fehlers bestimmt werden.

Beispiel:

```
try{ :Kunde RecRead(1, 1, _RecLock); ... :Auftrag RecRead(2, 1, _RecLock); ...}if (ErrGet()
```

ErrPos() liefert ein ganzzahliges Resultat, welches einem der definierten Labels entspricht. Ist ErrPos() gleich 0, trat der Fehler vor dem ersten Label auf.

Laufzeitfehler

Laufzeitfehler führen normalerweise zu einem Abbruch der Prozedur. Innerhalb von try-Blöcken können Laufzeitfehler abgefangen werden und wie andere Fehlerwerte in der Fehlerbehandlung verarbeitet werden. Dazu muss allerdings für jeden einzelnen Typ von Laufzeitfehler das Abfangen in try-Blöcken explizit ein- und ausgeschaltet werden.

Dies geschieht durch den Befehl ErrTryCatch(). Soll beispielsweise eine Division durch Null abgefangen werden, so geschieht dies durch ErrTryCatch(_ErrDivisionByZero, true). Die Einstellungen zum Abfangen von Laufzeitfehlern sind global. Damit sind Veränderungen für jeden try-Block relevant.

Laufzeitfehler, die außerhalb eines try-Blocks auftreten, führen allerdings immer zum Prozedurabbruch. Soll vor diesem Prozedurabbruch der Laufzeitfehler protokolliert werden, kann mit dem Befehl ErrCall() eine Funktion registriert werden, die vor dem Prozedurabbruch aufgerufen wird.


Die Anweisung ErrTryCatch() kann außerhalb eines try-Blocks stehen.

Beispiel:

```
ErrTryCatch(_ErrDivisionByZero, true);try{ fResult # (10.0 / 0.0);}switch (ErrGet()){ case _Er
```



Über die Funktion ErrIgnore() kann die Generierung der Laufzeitfehler _ErrStringOverflow, _ErrCnv und _ErrDecimal abgeschaltet werden.

Folgende Anweisung setzen den globalen Fehlerwert und führen zum Verlassen des try-Blocks, wenn der entsprechende Fehler nicht mit der Anweisung ErrTryIgnore() übergangen wird. Die Anweisungen sind in der Hilfe mit  gekennzeichnet.

- BinCopy
- BinDelete
- BinDirDelete
- BinDirMove
- BinDirOpen
- BinExport
- BinImport
- BinMove
- BinOpen
- BinRename
- BinUpdate
- DateMake
- DbaConnect
- DllLoad
- FsiAttributes
- FsiDelete
- FsiFileInfo
- FsiLock
- FsiOpen
- FsiPathChange
- FsiPathDelete
- FsiRead
- FsiRename
- FsiSize
- FsiWrite
- MailClose
- OEMLoad
- OEMSave
- RecBufCreate
- RecDelete
- RecInsert
- RecLink
- RecRead
- RecReplace
- SckClose
- SckConnect
- SckListen
- SckRead
- SckStartTls

- SckWrite
- SelAddLink
- SelAddSortFld
- SelClear
- SelCopy
- SelDefQuery
- SelDelete
- SelRead
- SelRecDelete
- SelRecInsert
- SelRun
- SelStore
- StoDirOpen
- StoOpen
- SysExecute
- TextCopy
- TextCreate
- TextDelete
- TextRead
- TextRename
- TextWrite
- TimeMake
- UrmDelete
- UrmOpen
- UrmPermGet
- UrmPermGetRaw
- UrmPropType
- UrmRead



trysub

Beginn der Ausnahmebehandlung

Verwandte

Befehle, try,

ErrThrowProc(),

ErrGet(), ErrPos(),

ErrSet(),

Siehe ErrTryIgnore(),

ErrTryCatch(),

ErrIgnore(),

ErrCall(),

Fehlerbehandlung

(Blog)

Das try-Konstrukt wirkt sich nicht auf aufgerufene Funktionen aus. Wird innerhalb einer aufgerufenen Funktion der globale Fehlerwert gesetzt (siehe ErrSet()), wirkt sich das erst bei der Rückkehr in die aufrufende Funktion aus. Wird zwischenzeitlich der globale Fehlerwert zurück gesetzt, wird die Verarbeitung fortgesetzt. Soll auch innerhalb der aufgerufenen Funktion eine Ausnahmebehandlung vorgenommen werden, muss auch hier ein try-Block definiert werden.

Soll die Verarbeitung auch dann unterbrochen werden, wenn in einer aufgerufenen Funktion der globale Fehlerwert gesetzt wird, muss das trysub-Konstrukt verwendet werden. Die aufgerufene Funktion wird dann im Fehlerfall unterbrochen. Der Funktionsaufruf kehrt in die aufrufende Funktion zurück und verlässt den trysub-Block. Die Funktion, in der der Fehler aufgetreten ist, kann mit der Anweisung ErrThrowProc() ermittelt werden. Die Fehlerverarbeitung kann vollständig in einer Funktion erfolgen.

Beispiel:

```
sub function1{ RecRead(..., _RecLock); ... function2(); RecReplace(..., _RecUnlock);}main{ t
```

Kann der Datensatz nicht gesperrt oder nicht zurückgeschrieben werden, wird der trysub-Block verlassen und die aufrufende Funktion bei der Fehlerbehandlung fortgesetzt. Auch bei tieferer Verschachtelung der Funktionsaufrufe, bleibt der trysub-Block aktiv.

Befindet sich in einer aufgerufenen Funktion ein eigener try-Block, wird die dort implementierte Fehlerbehandlung durchgeführt.

Für das Verlassen des trysub-Blocks, sowie die Verarbeitung von Fehlern, gelten die gleichen Funktionen und Regeln, wie beim try-Block.



while

Schleife mit Eintrittsbedingung

Verwandte Befehle,

do...while,

Siehe for...loop...while/until,

Vergleichsoperatoren,

break, cycle

Syntax:

```
while (<Ausdruck>) <Anweisung>
```

Diese Form der Anweisung wiederholt solange eine weitere Anweisung, wie das Resultat eines Ausdrucks wahr ist. Der Ausdruck muss ein logisches Ergebnis liefern.

Beispiel:

```
while (tNumber > 100){ Dec(tNumber); RecDelete(3);}
```

Ist in diesem Beispiel am Anfang der Schleife bereits die $tNumber \leq 100$, so wird die Schleifenanweisung überhaupt nicht durchgeführt. Ansonsten wird die Schleife solange wiederholt, bis die $tNumber \leq 100$ ist.

Um innerhalb einer Schleife eine extreme Schachtelung von if...else-Anweisungen zu vermeiden, können mit der Funktion cycle die restlichen Anweisungen in der Schleife übersprungen werden. Mit der Funktion break wird die Schleife beendet.



Bei der Verwendung des PASCAL-Styles muss eine andere Syntax beachtet werden. Weitere Informationen befinden sich im Abschnitt Style-Unterschiede.

Konvertierungsbefehle

Befehle zum Konvertieren von Werten in andere Datentypen

Datentypen,

Siehe Befehlsgruppen,

Befehlsliste

Mit folgenden Befehlen kann eine Konvertierung der Datentypen vorgenommen werden:

	Ziel	<u>alpha</u>	<u>bigint</u>	<u>caltime</u>	<u>date</u>	<u>decimal</u>	<u>float</u>	<u>int</u>	<u>logic</u>	<u>time</u>
Quelle										
<u>alpha</u>	-		<u>CnvBA()</u>	<u>CnvCA()</u>	<u>CnvDA()</u>	<u>CnvMA()</u>	<u>CnvFA()</u>	<u>CnvIA()</u>	-	<u>CnvTA()</u>
<u>bigint</u>	<u>CnvAB()</u>	-		<u>CnvCB()</u>	<u>CnvDB()</u>	<u>CnvMB()</u>	<u>CnvFB()</u>	<u>CnvIB()</u>	-	<u>CnvTB()</u>
<u>caltime</u>	<u>CnvAC()</u>	<u>CnvBC()</u>	-						-	
<u>date</u>	<u>CnvAD()</u>	<u>CnvBD()</u>	-					<u>CnvID()</u>	-	
<u>decimal</u>	<u>CnvAM()</u>	<u>CnvBM()</u>	-				<u>CnvFM()</u>	<u>CnvIM()</u>	-	
<u>float</u>	<u>CnvAF()</u>	<u>CnvBF()</u>	-			<u>CnvMF()</u>		<u>CnvIF()</u>	-	
<u>int</u>	<u>CnvAI()</u>	<u>CnvBI()</u>	-		<u>CnvDI()</u>	<u>CnvMI()</u>	<u>CnvFI()</u>		<u>CnvLI()</u>	<u>CnvTI()</u>
<u>logic</u>	-	-	-					<u>CnvIL()</u>	-	
<u>time</u>	<u>CnvAT()</u>	<u>CnvBT()</u>	-					<u>CnvIT()</u>	-	
Quelle										

Die Datentypen byte und word werden als Datentyp int behandelt. Bei der Zuweisung eines umgewandelten Wertes zu byte oder word kann es dadurch zu einem Überlauf (_ErrValueOverflow) kommen.

Befehle

- CnvAB
- CnvAC
- CnvAD
- CnvAF
- CnvAI
- CnvAM
- CnvAT
- CnvBA
- CnvBC
- CnvBD
- CnvBF
- CnvBI
- CnvBM
- CnvBT
- CnvCA
- CnvCB
- CnvDA
- CnvDB
- CnvDI
- CnvFA
- CnvFB

- CnvFI
- CnvFM
- CnvIA
- CnvIB
- CnvID
- CnvIF
- CnvIL
- CnvIM
- CnvIT
- CnvLI
- CnvMA
- CnvMB
- CnvMF
- CnvMI
- CnvTA
- CnvTB
- CnvTI

Konstanten

- FmtCaltimDate
- FmtCaltimDateBlank
- FmtCaltimDHMS
- FmtCaltimISO
- FmtCaltimRFC
- FmtCaltimTimeFull
- FmtCaltimTimeHM
- FmtCaltimTimeHMS
- FmtCaltimUTC
- FmtDateDMY
- FmtDateLong
- FmtDateLongYear
- FmtDateMDY
- FmtDateYMD
- FmtInternal
- FmtNone
- FmtNumCurrency
- FmtNumCurrencyIntl
- FmtNumHex
- FmtNumLeadZero
- FmtNumNoGroup
- FmtNumNoZero
- FmtNumPoint
- FmtTime24Hours
- FmtTimeHSeconds
- FmtTimeLFormat
- FmtTimeNoHours
- FmtTimeNoMarker
- FmtTimeNoMinutes
- FmtTimeNoSep
- FmtTimeSeconds

- FmtTimeSFormat



CnvAB(bigint1[, int2[, handle3[, int4]]) : alpha
 Umwandlung eines ganzzahligen Wertes (64 Bit) in einen
 alphanumerischen Wert

bigint1 Umzuwandelnder Wert

Formatoptionen (optional)

<u>__FmtNumNoGroup</u>	Keine Tausendertrennung
<u>__FmtNumNoZero</u>	Nullunterdrückung
<u>__FmtNumLeadZero</u>	Umwandlung mit führenden Nullen
<u>__FmtNumHex</u>	Darstellung in Hexadezimalform
<u>__FmtInternal</u>	Standardformat
<u>__FmtCaltimRFC</u>	Umwandlung nach <u>RFC 2822</u>
<u>__FmtCaltimISO</u>	Umwandlung nach <u>ISO 8601</u>
<u>__FmtCaltimDHMS</u>	Umwandlung in Tage, Stunden, Minuten, Sekunden und Millisekunden
<u>__FmtCaltimUTC</u>	Umwandlung in UTC-Zeit
<u>__FmtCaltimDate</u>	Konvertierung des Datums mit 'T' als Trenner zwischen Datum und Zeit
<u>__FmtCaltimDateBlank</u>	Konvertierung des Datums mit Leerzeichen als Trenner zwischen Datum und Zeit
<u>__FmtCaltimTimeHMS</u>	Konvertierung der Zeit im Format "HH:mm:ss"
<u>__FmtCaltimTimeHM</u>	Konvertierung der Zeit im Format "HH:mm"
<u>__FmtCaltimTimeFull</u>	Konvertierung der Zeit im Format "HH:mm:ss.nnnnnnn"
<u>__FmtNone</u>	Keine Optionen

handle3 Regionaldeskriptor (optional)

int4 Anzahl der Stellen (optional)

Resultat alpha Umgewandelter Wert

Siehe Verwandte Befehle, CnvBA(), ErrIgnore(),
LocaleLoad()

Diese Funktion wandelt einen Wert vom Typ bigint in einen Wert vom Typ alpha. Die angegebenen Formatoptionen (int2) entscheiden über das genaue Format.

Die Länge des Resultats ist abhängig vom umzuwandelnden Wert.

Als Optionen (int2) können folgende Konstanten angegeben werden:

- Numerische Darstellungen:
 - ◆ _FmtNumNoGroup Keine Tausendertrennung
 - ◆ _FmtNumNoZero Nullunterdrückung
 - ◆ _FmtNumLeadZero Umwandlung mit führenden Nullen
 - ◆ _FmtNumHex Darstellung in Hexadezimalform
 - ◆ _FmtInternal Standardformat
 - ◆ _FmtNone Keine Optionen
- Datums- und Zeitdarstellungen:
 - ◆ _FmtCaltimRFC Darstellung nach RFC 2822
 - ◆ _FmtCaltimISO Darstellung nach ISO 8601
 - ◆ _FmtCaltimDHMS Darstellung als Tage, Stunden, Minuten, Sekunden und Millisekunden als Zeitdifferenz zum 01.01.1601 00:00:00.000
 - ◆ _FmtCaltimUTC Umwandlung nach UTC
 - ◆ _FmtCaltimDate Konvertierung des Datums mit 'T' als Trenner zwischen Datum und Zeit
 - ◆ _FmtCaltimDateBlank Konvertierung des Datums mit Leerzeichen als Trenner zwischen Datum und Zeit
 - ◆ _FmtCaltimTimeHMS Konvertierung der Zeit im Format "HH:mm:ss"
 - ◆ _FmtCaltimTimeHM Konvertierung der Zeit im Format "HH:mm"
 - ◆ _FmtCaltimTimeFull Konvertierung der Zeit im Format "HH:mm:ss.nnnnnnn"

Optionen der Numerischen Darstellung können nicht mit Optionen der Datums- und Zeitdarstellung kombiniert werden.

Bei der Option _FmtCaltimISO werden die Bestandteile des formatierten Wertes durch die Kombination mit den Optionen _FmtCaltimDate(...) und _FmtCaltimTime... definiert. Aus jeder dieser zwei Gruppen kann nur eine Option mit _FmtCaltimISO kombiniert werden.

Die Option _FmtCaltimUTC kann nur in der Kombination mit einer der Konstanten _FmtCaltimRFC bzw. _FmtCaltimISO angegeben werden. Ohne die Option _FmtCaltimUTC wird die lokale Zeitzone verwendet.

Wird in (handle3) ein Deskriptor eines Locale-Objekt angegeben, bestimmen die dort eingetragenen Eigenschaften das Format des Rückgabewertes.

Wird in (int4) die Anzahl der Stellen vorgegeben, wird eine Zeichenkette der angegebenen Länge erzeugt. Die Zeichenkette wird mit Leerzeichen auf die entsprechende Länge aufgefüllt. Bei der Verwendung der Option _FmtNumLeadZero werden statt Leerzeichen Nullen verwendet. Zu der Länge zählen ebenfalls Vorzeichen und Tausendertrennzeichen. Übersteigt der Wert die angegebene Anzahl der Stellen, wird der Laufzeitfehler _ErrCnv erzeugt. Wird dieser Laufzeitfehler mit ErrIgnore(_ErrCnv, true) unterdrückt, wird eine leere Zeichenkette zurückgegeben.

Der Laufzeitfehler ErrCnv wird ebenfalls ausgelöst, wenn ein Wert < 0 oder > 8.962.174.367.999.999.999 (0x7C600A1AA3C03FFF\b) mit einer der _FmtCalttime...-Optionen umgewandelt werden soll.

Wird der Wert 0x8000000000000000\b ohne die Formatoption _FmtNumHex konvertiert, wird eine leere Zeichenkette zurückgegeben.

Beispiele:

```
// Numerische DarstellungCnvAB(3000000000\b, _FmtInternal) // '3000000000'CnvAB(2134\b, _FmtNone)
CnvAB(tBigint, _FmtCalttimeISO | _FmtCalttimeDate) // '2009-05-08'CnvAB(tBigint, _FmtCalttimeISO |
```

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt



CnvAC(caltime1, int2) : alpha

Umwandlung eines Zeitstempels in einen alphanumerischen Wert

caltime1 Umzuwandelnder Wert

Formatoptionen

	<u>FmtCaltimerFC</u>	Umwandlung nach <u>RFC 2822</u>
	<u>FmtCaltimerISO</u>	Umwandlung nach <u>ISO 8601</u>
	<u>FmtCaltimerDHMS</u>	Umwandlung in Tage, Stunden, Minuten, Sekunden und Millisekunden
	<u>FmtCaltimerUTC</u>	Umwandlung in UTC-Zeit
	<u>FmtCaltimerDate</u>	Konvertierung des Datums mit 'T' als Trenner zwischen Datum und Zeit
int2	<u>FmtCaltimerDateBlank</u>	Konvertierung des Datums mit Leerzeichen als Trenner zwischen Datum und Zeit
	<u>FmtCaltimerTimeHMS</u>	Konvertierung der Zeit im Format "HH:mm:ss"
	<u>FmtCaltimerTimeHM</u>	Konvertierung der Zeit im Format "HH:mm"
	<u>FmtCaltimerTimeFull</u>	Konvertierung der Zeit im Format "HH:mm:ss.nnnnnnn"
	<u>FmtCaltimerTZ</u>	Umwandlung mit Anzeige der Zeitzone

Resultat alpha Umgewandelter Wert

Siehe Verwandte Befehle, CnvCA()

Diese Funktion wandelt einen Wert vom Typ caltimer in einen Wert vom Typ alpha. Die angegebenen Formatoptionen (int2) entscheiden über das genaue Format.

Als Optionen (int2) können folgende Konstanten angegeben werden:

- FmtCaltimerFC Darstellung nach RFC 2822
- FmtCaltimerISO Darstellung nach ISO 8601
- FmtCaltimerDHMS Darstellung als Tage, Stunden, Minuten, Sekunden und Millisekunden als Zeitdifferenz zum 01.01.1601 00:00:00.000
- FmtCaltimerUTC Umwandlung nach UTC
- FmtCaltimerDate Konvertierung des Datums mit 'T' als Trenner zwischen Datum und Zeit

Kontakt

- _FmtCaltimеDateBlank Konvertierung des Datums mit Leerzeichen als Trenner zwischen Datum und Zeit
- _FmtCaltimеTimeHMS Konvertierung der Zeit im Format "HH:mm:ss"
- _FmtCaltimеTimeHM Konvertierung der Zeit im Format "HH:mm"
- _FmtCaltimеTimeFull Konvertierung der Zeit im Format "HH:mm:ss.nnnnnnn"
- _FmtCaltimеTZ Umwandlung mit Anzeige der Zeitzone

Bei der Option _FmtCaltimеISO werden die Bestandteile des formatierten Wertes durch die Kombination mit den Optionen _FmtCaltimеDate(...) und _FmtCaltimеTime... definiert. Aus jeder dieser zwei Gruppen kann nur eine Option mit _FmtCaltimеISO kombiniert werden. Zusätzlich kann mit _FmtCaltimеTZ die Zeitzone hinzugefügt werden.

Die Option _FmtCaltimеUTC kann nur in der Kombination mit einer der Konstanten _FmtCaltimеRFC bzw. _FmtCaltimеISO angegeben werden. Ohne die Option _FmtCaltimеUTC wird die angegebene Zeitonenabweichung (siehe vpBiasMinutes) beachtet.

Beispiele:

```
tCaltimе->vpBiasMinutes # 120;tCaltimе->vpDate # 08.05.2009;tCaltimе->vpTime # 16:52:23;CnvAC(tCa
```

Mögliche Laufzeitfehler:

_ErrValueInvalid Es wurde eine falsche Formatoption in (int2) angegeben.



CnvAD(date1[, int2[, handle3]]) : alpha

Umwandlung eines Datumswerts in einen alphanumerischen Wert

date1 Umzuwandelnder Wert

Formatoptionen (optional)

FmtDateLong Langes
Datumsformat

int2 FmtDateLongYear Immer
4-stelliges Jahr

FmtInternal Standardformat

FmtNone Keine Optionen

handle3 Regionaldeskriptor (optional)

Resultat alpha Umgewandelter Wert

Siehe Verwandte Befehle, CnvDA(),
ErrIgnore(), LocaleLoad()

Diese Funktion wandelt einen Wert vom Typ date in einen Wert vom Typ alpha. Die Umwandlung erfolgt in das aktuelle Datumsformat.

Wird in (handle3) der Deskriptor eines Locale-Objekts übergeben, bestimmen die dort eingetragenen Eigenschaften das Format des Rückgabewertes. Bei der Angabe von FmtInternal wird auch bei der Angabe eines Locale-Objekts das interne Format verwendet.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird ein leerer alpha-Wert zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt



CnvAF(float1[, int2[, handle3[, int4[, int5]]]]) : alpha

Umwandlung eines Gleitkommawertes in einen alphanumerischen Wert

float1 Umzuwandelnder Wert

Formatoptionen (optional)

<u>FmtNumNoGroup</u>	Keine Tausendertrennung
<u>FmtNumNoZero</u>	Nullunterdrückung
<u>FmtNumPoint</u>	'.' als Dezimaltrennzeichen
<u>FmtNumHex</u>	Umwandlung in Hexadezimaldarstellung
<u>FmtNumCurrency</u>	Währungsformat
<u>FmtNumCurrencyIntl</u>	Internationales Währungssymbol
<u>FmtNumLeadZero</u>	Umwandlung mit führenden Nullen
<u>FmtInternal</u>	Standardformat
<u>FmtNone</u>	Keine Optionen

handle3 Regionaldeskriptor (optional)

int4 Anzahl der Nachkommastellen (optional)

int5 Länge der Zeichenkette (optional)

Result alpha Umgewandelter Wert

Siehe Verwandte Befehle, CnvFA(), ErrIgnore(), LocaleLoad()

Diese Funktion wandelt einen Wert vom Typ float in einen Wert vom Typ alpha. Die Länge des Resultats ist abhängig vom umzuwandelnden Wert.

Wird in (handle3) der Deskriptor eines Locale-Objektes übergeben, bestimmen die dort eingetragenen Eigenschaften das Format des Rückgabewertes.

In (int4) wird die Anzahl der Nachkommastellen angegeben. Bei der Angabe von -1 werden die Ländereinstellungen von Windows verwendet. Besitzt der Wert mehr signifikante Nachkommastellen als angegeben, findet eine kaufmännische Rundung statt. Bei -2 werden so viele Nachkommastellen dargestellt, wie der Wert benötigt.

Ohne Angabe von (int4) wird die Anzahl der Nachkommastellen in den landesspezifischen Einstellungen benutzt. Wird kein Regionaldeskriptor angegeben (handle3 = 0), wird die systemdefinierten Einstellungen verwendet.

Die maximale Länge der erzeugten Zeichenkette kann in (int5) angegeben werden. Wird eine Zeichenkette erzeugt, deren Länge größer als die angegebene Länge ist, wird der Laufzeitfehler ErrCnv erzeugt. Wird dieser Laufzeitfehler mit ErrIgnore(_ErrCnv, true) unterdrückt, wird eine leere Zeichenkette zurückgegeben.

Wird in (int2) der Wert FmtNumLeadZero angegeben, muss in (int4) und in (int5) die Anzahl der Nachkommastellen und die Länge der Zeichenkette angegeben werden. Die Zahl in (float1) wird in eine Zeichenkette mit führenden Nullen gewandelt.



Der Übergabeparameter FmtNumLeadZero kann nicht mit den Parametern FmtNumCurrency oder FmtNumCurrencyIntl kombiniert werden.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird ein leerer alpha-Wert zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Beispiele:

```
CnvAF(1000.0)                // '1.000,00' CnvAF(1000.0, _FmtNumNoGroup)    // '1000,00' Cr
```

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt



CnvAI(int1[, int2[, handle3[, int4]]) : alpha
 Umwandlung eines ganzzahligen Wertes (32 Bit) in einen
 alphanumerischen Wert

int1 Umzuwandelnder Wert
 Formatoptionen (optional)
 _FmtNumNoGroup Keine
 Tausendertrennung
 _FmtNumNoZero Nullunterdrückung
 _FmtNumLeadZero Umwandlung mit
 führenden Nullen
 int2 _FmtNumHex Darstellung in
 Hexadezimalform
 _FmtInternal Standardformat
 _FmtNone Keine Optionen

handle3 Regionaldeskriptor (optional)

int4 Anzahl der Stellen (optional)

Resultat alpha Umgewandelter Wert

Siehe Verwandte Befehle

Diese Funktion wandelt einen Wert vom Typ int in einen Wert vom Typ alpha. Die angegebenen Formatoptionen entscheiden über das genaue Format.

Die Länge des Resultats ist abhängig vom umzuwandelnden Wert.

Wird in (handle3) ein Deskriptor eines Locale-Objekt angegeben, bestimmen die dort eingetragenen Eigenschaften das Format des Rückgabewertes.

Wird in (int4) die Anzahl der Stellen vorgegeben, wird eine Zeichenkette der angegebenen Länge erzeugt. Die Zeichenkette wird mit Leerzeichen auf die entsprechende Länge aufgefüllt. Bei der Verwendung der Option _FmtNumLeadZero wird statt Leerzeichen Nullen verwendet. Zu der Länge zählen ebenfalls Vorzeichen und Tausendertrennzeichen. Übersteigt der Wert die angegebene Anzahl der Stellen, wird der Laufzeitfehler _ErrCnv erzeugt. Wird dieser Laufzeitfehler mit ErrIgnore(_ErrCnv, true) unterdrückt, wird eine leere Zeichenkette zurückgegeben.

Wird der Wert 0x80000000 ohne die Formatoption _FmtNumHex konvertiert, wird eine leere Zeichenkette zurückgegeben.

Beispiele:

```
CnvAI(2134, _FmtInternal)
```

```
// '2134' CnvAI(2134, _FmtNone, hdlLangGB)
```

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei Typkonvertierung

_ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt



CnvAM(decimal1[, int2[, handle3[, int4[, int5]]]]) : alpha

Umwandlung eines Dezimalwerts in einen alphanumerischen Wert

decimal1 Umzuwandelnder Wert

Formatoptionen (optional)

	<u>_FmtNumNoGroup</u>	Keine Tausendertrennung
	<u>_FmtNumNoZero</u>	Nullunterdrückung
	<u>_FmtNumPoint</u>	"," als Dezimaltrennzeichen
int2	<u>_FmtNumCurrency</u>	Währungsformat
	<u>_FmtNumCurrencyIntl</u>	Internationales Währungssymbol
	<u>_FmtNumLeadZero</u>	Umwandlung mit führenden Nullen
	<u>_FmtInternal</u>	Standardformat
	<u>_FmtNone</u>	Keine Optionen

handle3 Regionaldeskriptor (optional)

int4 Anzahl der Nachkommastellen (optional)

int5 Länge der Zeichenkette (optional)

Resultat alpha Umgewandelter Wert

Siehe [Verwandte Befehle](#)

Diese Funktion wandelt einen Wert vom Typ decimal in einen Wert vom Typ alpha. Die Länge des Resultats ist abhängig vom umzuwandelnden Wert.

Wird in (handle3) der Deskriptor eines Locale-Objektes übergeben, bestimmen die dort eingetragenen Eigenschaften das Format des Rückgabewertes.

In (int4) wird die Anzahl der Nachkommastellen angegeben. Bei der Angabe von -1 werden die Ländereinstellungen von Windows verwendet. Besitzt der Wert mehr signifikante Nachkommastellen als angegeben, findet eine kaufmännische Rundung statt. Bei -2 werden so viele Nachkommastellen dargestellt, wie der Wert benötigt.

Ohne Angabe von (int4) wird die Anzahl der Nachkommastellen in den landesspezifischen Einstellungen benutzt. Wird kein Regionaldeskriptor angegeben (handle3 = 0), wird die systemdefinierte Einstellung verwendet.

Die maximale Länge der erzeugten Zeichenkette kann in (int5) angegeben werden. Wird eine Zeichenkette erzeugt, deren Länge größer als die angegebene Länge ist, wird der Laufzeitfehler ErrCnv erzeugt. Wird dieser Laufzeitfehler mit ErrIgnore(_ErrCnv, true) unterdrückt, wird eine leere Zeichenkette zurückgegeben.

Wird in (int2) der Wert _FmtNumLeadZero angegeben, muss in (int4) und in (int5) die Anzahl der Nachkommastellen und die Länge der Zeichenkette angegeben werden. Die Zahl in (decimal1) wird in eine Zeichenkette mit führenden Nullen gewandelt.



Die Option _FmtNumLeadZero kann nicht mit den Optionen _FmtNumCurrency oder _FmtNumCurrencyIntl kombiniert werden.

Kontakt

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird ein leerer Alphawert zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Beispiele:

```
CnvAM(1000.0\m) // '1.000,00'CnvAM(1000.0\m, _FmtNumNoGroup) // '1000,0'
```

Mögliche Laufzeitfehler

ErrCnv Fehler bei Typkonvertierung

ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt



CnvAT(time1[, int2[, handle3]]) : alpha

Umwandlung eines Zeitwerts in einen alphanumerischen Wert

time1 Umzuwandelnder Wert

Formatoptionen (optional)

	<u>FmtTimeSeconds</u>	Sekunden ausgeben
	<u>FmtTimeHSeconds</u>	Hundertstelsekunden ausgeben
	<u>FmtTimeNoHours</u>	Stunden nicht ausgeben
	<u>FmtTimeNoMinutes</u>	Minuten nicht ausgeben
int2	<u>FmtTime24Hours</u>	Immer 24 Stunden-Format
	<u>FmtTimeNoSep</u>	Ohne Trennzeichen
	<u>FmtTimeNoMarker</u>	Ohne Symbol
	<u>FmtTimeLFormat</u>	Darstellung aus <u>LclTimeLFormat</u>
	<u>FmtTimeSFormat</u>	Darstellung aus <u>LclTimeSFormat</u>
	<u>FmtInternal</u>	Standardformat
	<u>FmtNone</u>	Keine Optionen

handle3 Regionaldeskriptor (optional)

Resultat alpha Umgewandelter Wert

Siehe Verwandte Befehle, CnvTA(), ErrIgnore(), LocaleLoad()

Diese Funktion wandelt einen Wert vom Typ time in einen Wert vom Typ alpha.

Werden in (int2) keine Formatoptionen angegeben, erfolgt die Darstellung im länderspezifischen Format.

Wird in (int2) FmtTimeLFormat oder FmtTimeSFormat angegeben, erfolgt die Darstellung im Format der Eigenschaft LclTimeLFormat bzw. LclTimeSFormat vom angegebenen Regionaldeskriptors (handle3) oder der aktuellen Ländereinstellungen. Ist eine dieser Optionen angegeben, werden alle weiteren Optionen ignoriert.

Wird in (handle3) der Deskriptor eines Locale-Objekts übergeben, bestimmen die dort eingetragenen Eigenschaften das Format des Rückgabewerts.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird ein leerer Alphawert zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Beispiele:

```
CnvAT(01:02:03.04, _FmtTimeSeconds | _FmtTimeNoHours)
```

```
// '02:03'CnvAT(01:02:03.04, _FmtTimeSeconds | _FmtTimeNoHours)
```

Mögliche Laufzeitfehler:

Kontakt

ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt
ErrCnv Fehler bei der Typkonvertierung



CnvBA(alpha1[, int2]) : bigint

Umwandlung eines alphanumerischen Werts in einen ganzzahligen Wert (64 Bit)

alpha1 Umzuwandelnder Wert

Formatoptionen (optional)

_FmtNumHex Umwandlung

int2 von
hexadezimaler
Darstellung

Resultat bigint Umgewandelter Wert

Siehe Verwandte Befehle, CnvAB(),
ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ alpha in einen Wert vom Typ bigint.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt.
Es wird 0\b zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Bei der Angabe der Option _FmtNumHex wird die übergebene Zeichenkette als Hexadezimalzahl interpretiert und in eine ganze Zahl umgerechnet.

Beispiele:

```
CnvBA('128') // 128\bCnvBA('FF', _FmtNumHex) // 255\bCnvBA('FFFFFF01', _Fm
```

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung



CnvBC(caltimel) : bigint

Umwandlung eines Kalenderzeitwerts in einen ganzzahligen Wert (64 Bit)

caltimel Umzuwandelnder Wert

Resultat bigint Umgewandelter Wert

Siehe Verwandte Befehle,
CnvCB()

Diese Funktion wandelt einen Wert vom Typ caltimel in einen Wert vom Typ bigint. Der Wert gibt die Zeit an, die seit dem 01.01.1601 vergangen ist.

Als Einheit werden 100 Nanosekunden verwendet. Ein Unterschied von 10.000.000 Einheiten entspricht somit einer Zeitdifferenz von einer Sekunde.

Der Wert entspricht immer der UTC-Zeit (Coordinated Universal Time), die aktive Zeitzone (Eigenschaft vpBiasMinutes) wird bei der Umwandlung berücksichtigt und muss zuvor gesetzt werden. Die Umrechnung eines caltime-Wertes in der Zeitzone GMT+1 und in der Zeitzone GMT+2 ergibt den gleichen Wert, wenn sich die Zeitwerte um eine Stunde (zum Beispiel 14:00 Uhr und 15:00 Uhr) unterscheiden.



CnvBD(date1) : bigint

Datum in einen Zeitstempel umwandeln

date1 Umzuwandelnder Wert

Resultat bigint Umgewandelter Wert

Siehe Verwandte Befehle,
 CnvBT()

Diese Funktion wandelt einen Wert vom Typ date in einen Wert vom Typ bigint. Das Resultat gibt die Zeit an, die seit dem 01.01.1601 bis zu diesem Datum (00:00 Uhr) vergangen ist. Als Einheit werden 100 Nanosekunden verwendet.

Diese Funktion wird verwendet, um mit Zeitstempeln (in bigint umgewandelte calttime-Werte, siehe CnvBC()), Berechnungen vorzunehmen.

Wird der undefinierte Datums-Wert 0.0.0 übergeben, gibt die Anweisung -9.223.372.036.854.775.807 (kleinster Wert) als Resultat zurück.

Beispiel:

```
tTimeStamp # CnvBD(31.12.2009);
```



CnvBF(float1) : bigint

Umwandlung eines Gleitkommawerts in einen ganzzahligen Wert (64 Bit)

float1 Umzuwandelnder Wert

Resultat bigint Umgewandelter Wert

Siehe Verwandte Befehle,
CnvFB(), ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ float in einen Wert vom Typ bigint.

Vorhandene Nachkommastellen werden vor der Umwandlung kaufmännisch gerundet.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler _ErrCnv erzeugt.

Es wird 0\b zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei Typkonvertierung

CnvBI(int1) : bigint

Umwandlung eines ganzzahligen Werts (32 Bit) in einen ganzzahligen Wert (64 Bit)

int1 Umzuwandelnder Wert

Resultat bigint Umgewandelter Wert

Siehe Verwandte Befehle, CnvIB()



Ab der Version 5.8.01 ist diese Funktion nicht mehr explizit notwendig, da eine implizite Typkonvertierung zwischen int und bigint durchgeführt wird.

Diese Funktion wandelt einen Wert vom Typ int in einen Wert vom Typ bigint.



CnvBM(decimal1) : bigint



Umwandlung eines Dezimalwerts in einen ganzzahligen Wert (64 Bit)

decimal1 Umzuwandelnder Wert

Resultat bigint Umgewandelter Wert

Siehe Verwandte Befehle,
CnvMB(), ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ decimal in einen Wert vom Typ bigint.
Vorhandene Nachkommastellen werden vor der Wandlung kaufmännisch gerundet.

Ist (decimal1) nicht im Wertebereich von bigint oder auf undefiniert gesetzt, wird der Laufzeitfehler _ErrCnv erzeugt. Es wird 0\b zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei Typkonvertierung



CnvBT(time1) : bigint

Zeit in einen Zeitstempel umwandeln

time1 Umzuwandelnder Wert

Resultat bigint Umgewandelter Wert

Siehe Verwandte Befehle,
CnvBD()

Diese Funktion wandelt einen Wert vom Typ time in einen Wert vom Typ bigint. Das Resultat gibt die Zeit an, die seit dem 01.01.1601 bis zu dieser Uhrzeit vergangen ist. Als Einheit werden 100 Nanosekunden verwendet.

Diese Funktion wird verwendet, um mit Zeitstempeln (in bigint umgewandelte caltime-Werte, siehe CnvBC()), Berechnungen vorzunehmen.

Wird der undefinierte Zeit-Wert 24:00:00 übergeben, gibt die Anweisung -9.223.372.036.854.775.807 (kleinster Wert) als Resultat zurück.

Beispiele:

```
tTimeDiff # CnvBT(01:35:30);tTimeStamp # tTimeStamp + tTimeDiff;
```



CnvCA(alpha1, int2) : caltime

Umwandlung eines alphanumerischen Werts in einen Zeitstempel

alpha1 Umzuwandelnder Wert

Formatoptionen

_FmtCaltimRFC Umwandlung
von RFC 2822

_FmtCaltimUTC Umwandlung
nach UTC

int2 _FmtCaltimDHMS Umwandlung
aus Tagen,
Stunden,
Minuten,
Sekunden und
Millisekunden

Resultat caltime Umgewandelter Wert

Siehe Verwandte Befehle, CnvAC()

Diese Funktion wandelt einen Wert vom Typ alpha in einen Wert vom Typ caltime. Die angegebenen Formatoptionen (int2) entscheiden über das vorliegende Format.

Als Optionenen (int2) können folgende Konstanten angegeben werden:

- _FmtCaltimRFC Umwandlung von RFC 2822
- _FmtCaltimUTC Umwandlung nach UTC
- _FmtCaltimDHMS Umwandlung aus Tage, Stunden, Minuten, Sekunden und Millisekunden. Die einzelnen Bestandteile sind optional.

Die Option _FmtCaltimRFC kann mit _FmtCaltimUTC kombiniert werden. Wird die Option _FmtCaltimUTC angegeben, liegt der caltime-Wert anschließend in UTC-Zeit vor (vpBiasMinutes = 0).

Beispiele:

CnvCA('Fri, 08 May 2009 16:52:23 +0200', _FmtCaltimRFC)CnvCA('Fri, 08 May 2009 14:52:23 GMT', _F

Mögliche Laufzeitfehler:

_ErrValueInvalid Es wurde eine falsche Formatoption in (int2) angegeben.



CnvCB(bigint1[, logic2]) : caltime

Umwandlung eines ganzzahligen Werts (64 Bit) in einen Kalenderzeitwert

bigint1 Umzuwandelnder Wert

Umwandlung in lokalen

logic2 Zeitstempel /

UTC-Zeitstempel (optional)

Resultat caltime Umgewandelter Wert

Siehe Verwandte Befehle, CnvBC(),
ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ bigint in einen Wert vom Typ caltime. In (bigint1) wird die Anzahl der vergangenen 100 Nanosekunden seit dem 01.01.1601 angegeben.

Als Einheit werden 100 Nanosekunden verwendet. Ein Unterschied von 10.000.000 Einheiten entspricht somit einer Zeitdifferenz von einer Sekunde.

Der Wert (bigint1) entspricht dabei der Anzahl von 100-Nanosekunden-Intervallen seit dem 1.1.1601 00:00:00 UTC.

Wird (logic2) nicht, oder mit true angegeben, wird die lokale Zeitzone bei der Umwandlung berücksichtigt und in die Eigenschaft vpBiasMinutes der caltime-Variablen eingetragen. Wird im zweiten Parameter false angegeben, bleibt der Zeitstempel im UTC-Format (Universal Time Coordinated).

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler _ErrCnv erzeugt. Dies ist der Fall, wenn der umzuwandelnde Wert (bigint1) < 0 oder > 8.962.174.367.999.999.999 (0x7C600A1AA3C03FFF) ist. Es wird eine leere caltime zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei Typkonvertierung



CnvDA(alpha1[, int2[, handle3]]) : date

Umwandlung eines alphanumerischen Werts in einen Datumswert

alpha1 Umzuwandelnder Wert

Formatoptionen (optional)

FmtDateDMY Tag-Monat-Jahr

FmtDateMDY Monat-Tag-Jahr

int2

FmtDateYMD Jahr-Monat-Tag

FmtInternal Standardformat

FmtNone Keine Optionen

handle3 Regionaldeskriptor (optional)

Resultat date Umgewandelter Wert

Siehe Verwandte Befehle, CnvAD(),

ErrIgnore(), LocaleLoad()

Diese Funktion wandelt einen Wert vom Typ alpha in einen Wert vom Typ date.

Die Interpretation erfolgt in der länderspezifischen Reihenfolge von Tag, Monat und Jahr, sofern in (int2) keine andere Option verwendet wird. Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird ein leeres Datum zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Wird in (handle3) der Deskriptor eines Locale-Objektes übergeben, bestimmen die dort eingetragenen Eigenschaften das Format des übergebenen Wertes.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt



CnvDB(bigint1, logic2) : date

Umwandlung eines 64-Bit Zeitstempels in einen Datumswert

bigint1 Umzuwandelnder
 Zeitstempel

logic2 Umwandlung in lokales
 Datum / UTC-Datum

Resultat date Umgewandelter Wert

Siehe Verwandte Befehle,
 CnvTB(), ErrIgnore()

Diese Funktion wandelt einen 64-Bit Zeitstempel (bigint1) in einer bigint-Variablen in einen Wert vom Typ date um. Die im Zeitstempel enthaltene Uhrzeit wird dabei ignoriert. Ein 64-Bit Zeitstempel kann beispielsweise mit den Methoden vmSystemTime() und vmServerTime() generiert werden.

In (logic2) wird true übergeben, um den Zeitstempel in das lokale Datum umzuwandeln. Wird im zweiten Parameter false angegeben, erfolgt die Umwandlung in das UTC-Datum (Universal Time Coordinated).

Wird ein negativer Wert übergeben, kann er nicht umgewandelt werden und es wird der Laufzeitfehler _ErrCnv erzeugt. Der gleiche Laufzeitfehler wird generiert, wenn das Datum des Zeitstempels außerhalb des Definitionsbereiches des Datentyps date liegt. Im Falle eines Laufzeitfehlers ist der Rückgabewert nicht definiert. Bei der Übergabe von 0 wird 0.0.0 zurückgeliefert.

Beispiele:

```
tCaltime->vmSystemTime();                    // Setzen der Systemzeit einschließlich BiasMinutestCaltime->vpT
```

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei der Typkonvertierung



CnvDI(int1) : date

Umwandlung eines ganzzahligen Werts (32 Bit) in einen Datumswert

int1 Umzuwandelnder Wert

Resultat date Umgewandelter Wert

Siehe Verwandte Befehle,

CnvID(), ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ int in einen Wert vom Typ date.

(int1) entspricht der Anzahl der Tage seit dem 1.1.1900. Damit lassen sich einfach zu einem Datum eine bestimmte Anzahl von Tagen hinzurechnen. Mit CnvDI(0) kann ein leeres Datum generiert werden.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird ein leeres Datum zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung



CnvFA(alpha1[, int2[, handle3]]) : float

Umwandlung eines alphanumerischen Werts in einen Gleitkommawert

alpha1 Umzuwandelnder Wert

Formatoptionen (optional)

_FmtNumPoint '.' als

Dezimaltrennzeichen

int2

_FmtNumHex

Umwandlung aus

Hexadezimaldarstellung

_FmtNone

Keine Optionen

handle3 Regionaldeskriptor (optional)

Resultat float Umgewandelter Wert

Siehe Verwandte Befehle, CnvAF(),

ErrIgnore(), LocaleLoad()

Diese Funktion wandelt einen Wert vom Typ alpha in einen Wert vom Typ float.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler _ErrCnv erzeugt. Es wird 0.0\f zurückgeliefert, wenn der Laufzeitfehler übergangen wird.



Ohne die Option _FmtNumPoint wird als Dezimaltrennzeichen das im Betriebssystem hinterlegte Dezimaltrennzeichen verwendet.

Wird in (handle3) der Deskriptor eines Locale-Objektes übergeben, bestimmen die dort eingetragenen Eigenschaften das Format des übergebenen Wertes.

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei Angabe von _FmtNumHex, wenn der umzuwandelnde Wert nicht genau 16 hexadezimal kodierte Zeichen enthält oder außerhalb des Wertebereiches von float liegt.

_ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt



CnvFB(bigint1) : float

Umwandlung eines ganzzahligen Werts (64 Bit) in einen Gleitkommawert

bigint1 Umzuwandelnder Wert

Resultat float Umgewandelter Wert

Siehe Verwandte Befehle,
CnvBF()

Diese Funktion wandelt einen Wert vom Typ bigint in einen Wert vom Typ float.

Bei Werten in (bigint1) mit mehr als 15 Stellen ist das Resultat aufgrund der maximalen Genauigkeit des float-Typs ungenau.



CnvFI(int1) : float

Umwandlung eines ganzzahligen Werts (32 Bit) in einen
Gleitkommawert

int1 Umzuwandelnder Wert

Resultat float umgewandelter Wert

Siehe Verwandte Befehle,
 CnvIF()

Diese Funktion wandelt einen Wert vom Typ int in einen Wert vom Typ float.

CnvFM(decimal1) : float



Umwandlung eines Dezimalwerts in einen Gleitkommawert

decimal1 Umzuwandelnder Wert

Resultat float Umgewandelter Wert

Siehe Verwandte Befehle,
CnvMF(), ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ decimal in einen Wert vom Typ float. Der Wert wird dabei auf 15 Stellen gerundet.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird 0.0\f zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung



CnvIA(alpha1[, int2]) : int

Umwandlung eines alphanumerischen Werts in einen ganzzahligen Wert
(32 Bit)

alpha1 Umzuwandelnder Wert
Formatoptionen (optional)
 _FmtNumHex Umwandlung
int2 von
hexadezimaler
Darstellung

Resultat int Umgewandelter Wert

Siehe Verwandte Befehle, CnvAI(),
ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ alpha in einen Wert vom Typ int. Dabei werden alle Ziffern innerhalb der Zeichenkette berücksichtigt. Alle anderen Zeichen, mit Ausnahme des Vorzeichens vor der ersten Ziffer, werden ignoriert.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler _ErrCnv erzeugt. Es wird 0 zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Bei der Angabe der Option _FmtNumHex wird die übergebene Zeichenkette als Hexadezimalzahl interpretiert und in einen ganzzahligen Wert umgerechnet.

Beispiele:

CnvIA('128')

// 128CnvIA('FF', _FmtNumHex)

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei Typkonvertierung



CnvIB(bigint1) : int

Umwandlung eines ganzzahligen Werts (64 Bit) in einen ganzzahligen Wert (32 Bit)

bigint1 Umzuwandelnder Wert

Resultat int Umgewandelter Wert

Siehe Verwandte Befehle,
CnvBI(), ErrIgnore()



Ab der Version 5.8.01 ist diese Funktion nicht mehr explizit notwendig, da eine implizite Typkonvertierung zwischen int und bigint durchgeführt wird.

Diese Funktion wandelt einen Wert vom Typ bigint in einen Wert vom Typ int.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird 0 zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei der Typkonvertierung

CnvID(date1) : int



Umwandlung eines Datumswerts in einen ganzzahligen Wert (32 Bit)

date1 Umzuwandelnder Wert

Resultat int Umgewandelter Wert

Siehe Verwandte Befehle,

CnvDI()

Diese Funktion wandelt einen Wert vom Typ date in einen Wert vom Typ int.

Das Resultat entspricht der Anzahl der Tage seit dem 1.1.1900.

Beispiele:

```
CnvID(1.1.1900) // 1CnvID(12.6.1984) // 30845
```



CnvIF(float1) : int

Umwandlung eines Gleitkommawertes in einen ganzzahligen Wert (32 Bit)

float1 Umzuwandelnder Wert

Resultat int Umgewandelter Wert

Siehe Verwandte Befehle,
CnvFI(), ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ float in einen Wert vom Typ int. Der Wert in (float1) wird kaufmännisch gerundet.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler _ErrCnv erzeugt. Es wird 0 zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei Typkonvertierung

CnvIL(logic1) : int



Umwandlung eines logischen Werts in einen ganzzahligen Wert (32 Bit)

logic1 Umzuwandelnder Wert

Resultat int Umgewandelter Wert

Siehe Verwandte Befehle,

CnvLI()

Diese Funktion wandelt einen Wert vom Typ logic in einen Wert vom Typ int.

Falls (logic1) gleich true ist wird 1 zurückgeliefert, bei false wird 0 zurückgeliefert.



CnvIM(decimal1) : int

Umwandlung eines Dezimal-Werts in einen ganzzahligen Wert (32 Bit)

decimal1 Umzuwandelnder Wert

Resultat int Umgewandelter Wert

Siehe [Verwandte Befehle](#),
[CnvMI\(\)](#), [ErrIgnore\(\)](#)

Diese Funktion wandelt einen Wert vom Typ decimal in einen Wert vom Typ int. Der Wert in (decimal1) wird kaufmännisch gerundet.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird 0 zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

CnvIT(time1) : int



Umwandlung eines Zeitwerts in einen ganzzahligen Wert (32 Bit)

time1 Umzuwandelnder Wert

Resultat int Umgewandelter Wert

Siehe Verwandte Befehle,

CnvTI()

Diese Funktion wandelt einen Wert vom Typ time in einen Wert vom Typ int.

Das Resultat entspricht der Anzahl der Millisekunden seit 00:00 Uhr.

CnvLI(int1) : logic



Umwandlung eines ganzzahligen Werts (32 Bit) in einen logischen Wert

int1 Umzuwandelnder Wert

Resultat logic Umgewandelter Wert

Siehe Verwandte Befehle,

CnvIL()

Diese Funktion wandelt einen Wert vom Typ int in einen Wert vom Typ logic.

Falls (int1) 0 ist wird false zurückgeliefert, bei einem Wert ungleich 0 wird true zurückgeliefert.



CnvMA(alpha1[, int2[, handle3]]) : decimal

Umwandlung eines alphanumerischen Werts in einen Dezimalwert

alpha1 Umzuwandelnder Wert

Formatoptionen (optional)

int2 FmtNumPoint '.' als
Dezimaltrennzeichen

FmtNone Keine Optionen

handle3 Regionaldeskriptor (optional)

Resultat decimal Umgewandelter Wert

Siehe Verwandte Befehle, CnvAM(),
ErrIgnore(), LocaleLoad()

Diese Funktion wandelt einen Wert vom Typ alpha in einen Wert vom Typ decimal.

Wird in (handle3) der Deskriptor eines Locale-Objektes übergeben, bestimmen die dort eingetragenen Eigenschaften das Format des übergebenen Wertes.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird 0\m zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

ErrHdlInvalid Der in (handle3) übergebene Regionaldeskriptor ist nicht korrekt

CnvMB(bigint1) : decimal



Umwandlung eines ganzzahligen Werts (64 Bit) in einen Dezimalwert

bigint1 Umzuwandelnder Wert

Resultat decimal Umgewandelter Wert

Siehe Verwandte Befehle, CnvBM()

Diese Funktion wandelt einen Wert vom Typ bigint in einen Wert vom Typ decimal.

CnvMF(float1) : decimal



Umwandlung eines Gleitkommawerts in einen Dezimalwert

float1 Umzuwandelnder Wert

Resultat decimal Umgewandelter Wert

Siehe Verwandte Befehle, CnvFM(),
ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ float in einen Wert vom Typ decimal.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt.
Es wird 0\m zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

CnvMI(int1) : decimal



Umwandlung eines ganzzahligen Werts (32 Bit) in einen Dezimalwert

int1 Umzuwandelnder Wert

Resultat decimal Umgewandelter Wert

Siehe Verwandte Befehle, CnvIM()

Diese Funktion wandelt einen Wert vom Typ int in einen Wert vom Typ decimal.



CnvTA(alpha1[, int2]) : time

Umwandlung eines alphanumerischen Werts in einen Zeitwert

alpha1 Umzuwandelnder Wert

Formatoptionen (optional)

_FmtTimeNoHours Keine
Stunden
angegeben

int2 _FmtTimeNoMinutes Keine
Minuten
angegeben

_FmtNone Keine
Optionen

Resultat time Umgewandelter Wert

Siehe Verwandte Befehle, CnvAT(),
ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ alpha in einen Wert vom Typ time.

Werden keine Formatoptionen angegeben, erfolgt die Interpretation im Format Stunde-Minute-Sekunde-Hundertstelsekunden.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt. Es wird eine leere Uhrzeit zurückgeliefert und der globale Fehlerwert auf ErrValueInvalid gesetzt, wenn der Laufzeitfehler übergangen wird.

Beispiele:

```
CnvTA('02:03', _FmtTimeNoHours)
```

```
// 00:02:03,00CnvTA('03', _FmtTimeNoHours | _Fmt
```

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung



CnvTB(bigint1, logic2) : time

Umwandlung eines 64-Bit Zeitstempels in einen Zeitwert

bigint1 Umzuwandelnder
 Zeitstempel

logic2 Umwandlung in lokale Zeit
 / UTC-Zeit

Resultat time Umgewandelter Wert

Siehe Verwandte Befehle,
 CnvDB(), ErrIgnore()

Diese Funktion wandelt einen 64-Bit Zeitstempel (bigint1) in einer bigint-Variablen in einen Wert vom Typ time um. Das im Zeitstempel enthaltene Datum wird dabei ignoriert. Ein 64-Bit Zeitstempel kann beispielsweise mit den Methoden vmSystemTime() und vmServerTime() generiert werden.

In (logic2) wird true übergeben, um den Zeitstempel in die lokale Zeit umzuwandeln. Wird im zweiten Parameter false angegeben, erfolgt die Umwandlung in die UTC-Zeit (Universal Time Coordinated).

Wird ein negativer Wert übergeben, kann er nicht umgewandelt werden und es wird der Laufzeitfehler _ErrCnv erzeugt. In diesem Fall ist der Rückgabewert nicht definiert. Bei der Übergabe von 0 wird 24:00:00.0 zurückgegeben.

Mögliche Laufzeitfehler:

_ErrCnv Fehler bei der Typkonvertierung

CnvTI(int1) : time



Umwandlung eines ganzzahligen Werts (32 Bit) in einen Zeitwert

int1 Umzuwandelnder Wert

Resultat time Umgewandelter Wert

Siehe Verwandte Befehle,

CnvIT(), ErrIgnore()

Diese Funktion wandelt einen Wert vom Typ int in einen Wert vom Typ time.

Der Wert (int1) entspricht der Anzahl von Millisekunden seit 00:00 Uhr.

Kann der Wert nicht umgewandelt werden, wird der Laufzeitfehler ErrCnv erzeugt.
Es wird eine leere Uhrzeit zurückgeliefert, wenn der Laufzeitfehler übergangen wird.

Mögliche Laufzeitfehler:

ErrCnv Fehler bei Typkonvertierung

Konstanten für Formatierungen

Liste aller Formatoptionen für Konvertierungen

Siehe Konvertierungsbefehle

<u>FmtCaltimeDate</u>	Konvertierung des Datums mit 'T' als Trenner zwischen Datum und Zeit
<u>FmtCaltimeDateBlank</u>	Konvertierung des Datums mit Leerzeichen als Trenner zwischen Datum und Zeit
<u>FmtCaltimeDHMS</u>	Umwandlung in Tage, Stunden, Minuten, Sekunden und Millisekunden
<u>FmtCaltimeISO</u>	Umwandlung nach <u>ISO 8601</u>
<u>FmtCaltimeRFC</u>	Datumsumwandlung nach <u>RFC 2822</u>
<u>FmtCaltimeTimeFull</u>	Konvertierung der Zeit im Format "HH:mm:ss.nnnnnnnn"
<u>FmtCaltimeTimeHM</u>	Konvertierung der Zeit im Format "HH:mm"
<u>FmtCaltimeTimeHMS</u>	Konvertierung der Zeit im Format "HH:mm:ss"
<u>FmtCaltimeTZ</u>	Umwandlung mit Anzeige der Zeitzone
<u>FmtCaltimeUTC</u>	Datumsumwandlung in UTC
<u>FmtDateDMY</u>	Datumsformat Tag-Monat-Jahr
<u>FmtDateMDY</u>	Datumsformat Monat-Tag-Jahr
<u>FmtDateYMD</u>	Datumsformat Jahr-Monat-Tag
<u>FmtDateLong</u>	Langes Datumsformat
<u>FmtDateLongYear</u>	Lange Datumsanzeige
<u>FmtInternal</u>	Standardformatierung
<u>FmtNone</u>	Keine Formatierung
<u>FmtNumCurrency</u>	Umwandlung in Währungsformat (Symbol)
<u>FmtNumCurrencyIntl</u>	Umwandlung in Währungsformat (Code)
<u>FmtNumHex</u>	Umwandlung in oder von hexadezimaler Darstellung
<u>FmtNumLeadZero</u>	Umwandlung mit führenden Nullen
<u>FmtNumNoGroup</u>	Keine Tausendertrennung
<u>FmtNumNoZero</u>	Keine Nullen
<u>FmtNumPoint</u>	Punkt zur Dezimaltrennung
<u>FmtTime24Hours</u>	24 Stunden-Format
<u>FmtTimeHSeconds</u>	Formatierung mit Hundertstelsekunden
<u>FmtTimeLFormat</u>	Formatierung unter Berücksichtigung von <u>LclTimeLFormat</u>
<u>FmtTimeNoHours</u>	Formatierung ohne Stunden
<u>FmtTimeNoMarker</u>	Formatierung ohne Zeitsymbol
<u>FmtTimeNoMinutes</u>	Formatierung ohne Minuten
<u>FmtTimeNoSep</u>	Formatierung ohne Trennzeichen
<u>FmtTimeSeconds</u>	Formatierung mit Sekunden
<u>FmtTimeSFormat</u>	Formatierung unter Berücksichtigung von <u>LclTimeSFormat</u>

_FmtCaltimеDate

Konvertierung des Datums mit 'T' als Trenner zwischen Datum und Zeit

Wert 2.097.152 / 0x00200000

Verwandte Befehle,

Siehe _FmtCaltimеISO,

_FmtCaltimеDateBlank

Die Verwendung der Option _FmtCaltimеDate setzt die Option _FmtCaltimеISO voraus. Mit dieser Option ist das Datum Bestandteil des konvertierten Wertes. Als Trennzeichen zwischen Datum und Zeit wird 'T' verwendet. Die Option kann zusätzlich mit _FmtCaltimеUTC sowie einer der Konstanten _FmtCaltimеTimeHMS, _FmtCaltimеTimeHM bzw. _FmtCaltimеTimeFull kombiniert werden. Ist keine dieser drei Optionen angegeben, wird kein Trennzeichen verwendet.

Beispiele:

```
CnvAC(tCaltimе, _FmtCaltimеISO | _FmtCaltimеDate | _FmtCaltimеTimeHMS); // 2013-11-28T16:52:23C
```


_FmtCaltimеDateBlank

Konvertierung des Datums mit Leerzeichen als Trenner zwischen Datum und Zeit

Wert 4.194.304 /
0x00400000

Verwandte

Siehe Befehle,
_FmtCaltimеISO,
_FmtCaltimеDate

Die Verwendung der Option _FmtCaltimеDateBlank setzt die Option _FmtCaltimеISO voraus. Mit dieser Option ist das Datum Bestandteil des konvertierten Wertes. Als Trennzeichen zwischen Datum und Zeit wird ein Leerzeichen verwendet. Die Option kann zusätzlich mit _FmtCaltimеUTC sowie einer der Konstanten _FmtCaltimеTimeHMS, _FmtCaltimеTimeHM bzw. _FmtCaltimеTimeFull kombiniert werden. Ist keine dieser drei Optionen angegeben, wird kein Trennzeichen verwendet.

Beispiele:

```
CnvAC(tCaltimе, _FmtCaltimеISO | _FmtCaltimеDateBlank | _FmtCaltimеTimeHMS); // 2013-11-28 16:52
```

_FmtCaltimeDHMS
 Umwandlung nach Tage, Stunden, Minuten, Sekunden und Millisekunden seit
 01.01.1601 00:00:00.000

Wert 1.073.741.828 /
 0x40000004

Verwandte

Siehe Befehle,
_FmtCaltimeRFC,
_FmtCaltimeISO

Durch die Verwendung von _FmtCaltimeDHMS werden Datums- / Zeitwertes vom Datentyp bigint oder caltime in den Typ alpha in das Format "5d 7h 20m 3s 510ms" konvertiert. Nicht vorhandene Bestandteile werden weggelassen. Die angegebenen Werte entsprechen der Differenz zum 01.01.1601 00:00:00.000. Daher ist diese Option für die Ausgabe von berechneten Zeitdifferenzen ausgelegt.

Beispiele:

```
tCaltime->vpDate # 03.01.1601;tCaltime->vpTime # 11:20:07;CnvAC(tCaltime, _FmtCaltimeDHMS); // 2
```

_FmtCaltimeISO
 Umwandlung nach ISO 8601

Wert 1.073.741.825 /
 0x40000001

Verwandte Befehle,

Siehe _FmtCaltimeRFC,
_FmtCaltimeDHMS

Durch die Verwendung von _FmtCaltimeISO wird bei der Umwandlung eines Datums- / Zeitwertes vom Datentyp bigint oder caltime in den Typ alpha der ISO 8601 Standard verwendet. Diese Option kann mit _FmtCaltimeUTC kombiniert werden, um den Datums- / Zeitwert in UTC umzurechnen.

Zusätzlich kann die Option mit einer der Konstanten _FmtCaltimeDate bzw. _FmtCaltimeDateBlank sowie einer der Konstanten _FmtCaltimeTimeHMS, _FmtCaltimeTimeHM bzw. _FmtCaltimeTimeFull kombiniert werden um die anzuzeigenden Bestandteile zu definieren.

Beispiele:

```
CnvAC(tCaltime, _FmtCaltimeISO | _FmtCaltimeDateBlank | _FmtCaltimeTimeFull); // '2009-05-08 16:5
```

_FmtCaltimerFC
Umwandlung nach RFC 2822
Wert 1.073.741.826 /
0x40000002

Verwandte Befehle,
Siehe _FmtCaltimerISO,
_FmtCaltimerDHMS

Durch die Verwendung von _FmtCaltimerFC wird bei der Umwandlung eines Datums- / Zeitwertes vom Datentyp bigint oder caltimer in den Typ alpha der RFC 2822 Standard verwendet. Diese Option kann mit _FmtCaltimerUTC kombiniert werden, um den Datums- / Zeitwert in UTC umzurechnen.

_FmtCaltimeTimeFull
Konvertierung der Zeit im Format "HH:mm:ss.nnnnnnnn"
Wert 581.632 / 0x0008E000

Verwandte Befehle,

Siehe _FmtCaltimeISO,
_FmtCaltimeTimeHMS,
_FmtCaltimeTimeHM

Die Verwendung der Option _FmtCaltimeTimeFull setzt die Option _FmtCaltimeISO voraus. Mit dieser Option ist die Zeit im Format "HH:mm:ss.nnnnnnnn" Bestandteil des konvertierten Wertes. Die Option kann zusätzlich mit _FmtCaltimeUTC sowie einer der Konstanten _FmtCaltimeDate bzw. _FmtCaltimeDateBlank kombiniert werden.

Beispiele:

```
CnvAC(tCaltime, _FmtCaltimeISO | _FmtCaltimeDate | _FmtCaltimeTimeFull); // 2013-11-28T05:06:04.
```

_FmtCaltimeTypeHM
Konvertierung der Zeit im Format "HH:mm"
Wert 24.576 / 0x00006000

Verwandte Befehle,

Siehe _FmtCaltimeTypeISO,
_FmtCaltimeTypeHMS,
_FmtCaltimeTypeFull

Die Verwendung der Option _FmtCaltimeTypeHM setzt die Option _FmtCaltimeTypeISO voraus. Mit dieser Option ist die Zeit im Format "HH:mm" Bestandteil des konvertierten Wertes. Die Option kann zusätzlich mit _FmtCaltimeTypeUTC sowie einer der Konstanten _FmtCaltimeTypeDate bzw. _FmtCaltimeTypeDateBlank kombiniert werden.

Beispiele:

```
CnvAC(tCaltimeType, _FmtCaltimeTypeISO | _FmtCaltimeTypeDate | _FmtCaltimeTypeHM); // 2013-11-28T05:06CnvAC
```

_FmtCaltimeTypeHMS
Konvertierung der Zeit im Format "HH:mm:ss"
Wert 57.344 / 0x0000E000

Verwandte Befehle,

Siehe _FmtCaltimeTypeISO,
_FmtCaltimeTypeHM,
_FmtCaltimeTypeFull

Die Verwendung der Option _FmtCaltimeTypeHMS setzt die Option _FmtCaltimeTypeISO voraus. Mit dieser Option ist die Zeit im Format "HH:mm:ss" Bestandteil des konvertierten Wertes. Die Option kann zusätzlich mit _FmtCaltimeTypeUTC sowie einer der Konstanten _FmtCaltimeTypeDate bzw. _FmtCaltimeTypeDateBlank kombiniert werden.

Beispiele:

```
CnvAC(tCaltimeType, _FmtCaltimeTypeISO | _FmtCaltimeTypeDate | _FmtCaltimeTypeFull); // 2013-11-28T05:06:04
```

_FmtCaltimeUTC
Umwandlung in UTC
Wert 64 / 0x00000040

Verwandte Befehle,

Siehe _FmtCaltimeRFC,
_FmtCaltimeISO,
_FmtCaltimeDHMS

Die Konstante _FmtCaltimeUTC kann zusätzlich zu einer der Konstanten _FmtCaltimeRFC bzw. _FmtCaltimeISO angegeben werden. Die Umwandlung findet dann in UTC statt.

_FmtDateDMY
Datumsformatierung Tag-Monat-Jahr
Wert 2 / 0x0002

Verwandte

Siehe Befehle,
CnvDA()

Durch die Verwendung dieser Option bei CnvDA() wird das Datum in der Reihenfolge Tag-Monat-Jahr interpretiert.

_FmtDateLong
Langes Datumsformat
Wert 1 / 0x0001

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von _FmtDateLong wird bei der Umwandlung eines
Datumswerts in einen alphanumerischen Wert das lange Datumsformat verwendet.

_FmtDateLongYear
4-stellige Jahreszahl
Wert 2 / 0x0002

Siehe Verwandte
Befehle

Durch die Verwendung von _FmtDateLongYear wird bei der Umwandlung eines Datums werts in einen alphanumerischen Wert eine 4-stellige Jahreszahl verwendet.

_FmtDateMDY
Datumsformatierung Monat-Tag-Jahr
Wert 1 / 0x0001

Verwandte

Siehe Befehle,
CnvDA()

Durch die Verwendung dieser Option bei CnvDA() wird das Datum in der Reihenfolge Monat-Tag-Jahr interpretiert.

_FmtDateYMD
Datumsformatierung Jahr-Monat-Tag
Wert 3 / 0x0003

Verwandte

Siehe Befehle,
CnvDA()

Durch die Verwendung dieser Option bei CnvDA() wird das Datum in der Reihenfolge Jahr-Monat-Tag interpretiert.

_FmtInternal
Standard-Formatoption

Wert 256 /
0x00000100

Siehe Verwandte
Befehle

Die Verwendung der Standard-Formatoption führt zu einem immer gleichen Ausgabeformat, unabhängig von den länderspezifischen Einstellungen.

Beispiele:

int : 12345 -12345

float : 781.899 -781.899 (variable Anzahl Nachkommastellen)

date : 02.07.2001

time : 22.18.51

Die Option kann nicht mit anderen Optionen kombiniert werden. Nachkommastellen und Regionaldeskriptor werden ignoriert.

_FmtNone

Leere Formatoption

Wert 0 /
0x00000000

Siehe [Verwandte](#)
[Befehle](#)

Die leere Formatoption entspricht der Angabe keiner bestimmten Formatoption.

_FmtNone kann nicht mit anderen Optionen kombiniert werden.

`_FmtNumCurrency`
Währungsformat (Symbol)
Wert 16 / 0x00000010

Siehe [Verwandte Befehle](#),
[_FmtNumCurrencyIntl](#)

Mit `_FmtNumCurrency` wird bei der Umwandlung von numerisch nach alphanumerisch ein länderspezifisches Währungsformat zugrundegelegt. Dadurch werden die Trennzeichen, die Tausendertrennung, die Nachkommastellen usw. entsprechend der jeweiligen Norm definiert. Es ist zu beachten, dass verschiedene Währungssymbole nur in Verbindung mit dem landesspezifischen Zeichensatz korrekt dargestellt werden können.

Bei mehr als zwei Nachkommastellen wird der Wert kaufmännisch gerundet.

Beispiel:

\$78,342.55
1.234,56 €

_FmtNumCurrencyIntl
Währungsformat (Code)
Wert 32 / 0x00000020

Siehe Verwandte Befehle,
_FmtNumCurrency

Mit _FmtNumCurrencyIntl wird bei der Umwandlung von numerisch nach alphanumerisch ein länderspezifisches Währungsformat nach internationaler Norm zugrundegelegt. Dadurch werden die Nachkommastellen und der Währungscode definiert. Im Gegensatz zu _FmtNumCurrency wird der internationale Währungscode (ISO 4217) anstelle eines Währungssymbols benutzt. Die Trennzeichen entsprechen dabei immer den Trennzeichen der lokalen Währung.

Bei mehr als zwei Nachkommastellen wird der Wert kaufmännisch gerundet.

Beispiel:

EUR 1.234,56

_FmtNumHex

Umwandlung in oder von hexadezimaler Darstellung

Wert 512 /
0x00000200

Verwandte

Siehe Befehle,

_FmtIntFlags

Mit diesem Übergabeparameter kann mit den Funktionen CnvAI(), CnvAB() und CnvAF() ein numerischer Wert in die hexadezimale Darstellung konvertiert werden.

Umgekehrt kann mit den Funktionen CnvIA(), CnvBA() und CnvFA() aus der hexadezimalen Darstellung in einen numerischen Wert konvertiert werden.

Es können positive und negative Zahlen gewandelt werden.

Beispiele:

```
CnvAI(255, _FmtNumHex)           // ffCnvAI(-255, _FmtNumHex)           // ffffffff01CnvAI(-255, _FmtNumHex)
```

`_FmtNumLeadZero`

Führende Nullen

Wert `8 /`
`0x00000008`

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von `_FmtNumLeadZero` wird bei der Umwandlung von numerisch nach alphanumerisch die Zeichenkette mit Nullen auf die angegebene Länge aufgefüllt.

Kann die resultierende Zeichenkette nicht in den angegebenen Stellen angezeigt werden, oder wird das Ergebnis in einen zu kurzen String zurückgegeben, wird er mit `'*'` aufgefüllt.

Beispiele:

```
CnvAI(1234, _FmtNumLeadZero, 0, 5)
```

```
// '01234' CnvAI(1234, _FmtNumLeadZero,
```

_FmtNumNoGroup
Keine Tausendertrennung

Wert 1 /
0x00000001

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von _FmtNumNoGroup wird bei der Umwandlung von numerisch nach alphanumerisch eine Darstellung ohne Tausendertrennung erreicht.

Beispiel:

1277,500

`_FmtNumNoZero`

Keine Nullen

Wert 2 /
0x00000002

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von `_FmtNumNoZero` wird bei der Umwandlung von numerisch nach alphanumerisch eine leere Zeichenkette zurückgeliefert, wenn der umzuwandelnde Wert gleich 0 ist.

`_FmtNumPoint`

Punkt als Dezimaltrennzeichen

Wert `5 /`
`0x00000005`

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von `_FmtNumPoint` wird bei der Umwandlung zwischen numerisch und alphanumerisch ein Punkt (.) als Dezimaltrennzeichen benutzt (unabhängig von der länderspezifischen Einstellung).

Beispiel:

```
CnvMA('123.12') // 12.312,00CnvMA('123.12', _FmtNumPoint) // 123,12
```

`_FmtTime24Hours`
Formatierung im 24 Stunden-Format
Wert 64 /
0x0040

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von `_FmtTime24Hours` wird bei der Umwandlung von Zeit nach alphanumerisch (siehe [CnvAT\(\)](#)) immer das 24 Stunden-Format benutzt (unabhängig von der länderspezifischen Einstellung).

_FmtTimeHSeconds

Formatierung mit Hundertstelsekunden

Wert 2 / 0x0002

Siehe Verwandte
Befehle

Durch die Verwendung von _FmtTimeHSeconds werden bei der Umwandlung von Zeit nach alphanumerisch (siehe CnvAT()) die Sekunden und Hundertstelsekunden mit berücksichtigt.

_FmtTimeLFormat
Formatierung unter Berücksichtigung von LclTimeLFormat
Wert 8.192 / 0x2000

Verwandte
Siehe Befehle,
FmtTimeFlags,
CnvAT()

Durch die Verwendung von _FmtTimeLFormat werden bei der Umwandlung von Zeit nach alphanumerisch (siehe CnvAT()) die Formatierung der Eigenschaft LclTimeLFormat berücksichtigt.



Ist diese Option bei CnvAT() angegeben, werden alle weiteren Optionen ignoriert.

`_FmtTimeNoHours`
Formatierung ohne Stunden
Wert 8 / 0x0008

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von `_FmtTimeNoHours` werden bei der Umwandlung zwischen Zeit und alphanumerisch (siehe [CnvAT\(\)](#) und [CnvTA\(\)](#)) die Stunden nicht berücksichtigt.

`_FmtTimeNoMarker`
Formatierung ohne Zeitsymbol

Wert 32 /
0x0020

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von `_FmtTimeNoMarker` werden bei der Umwandlung von Zeit nach alphanumerisch (siehe [CnvAT\(\)](#)) keine Symbole beim 12-Stunden-Format (zum Beispiel AM/PM) benutzt (unabhängig von der länderspezifischen Einstellung).

_FmtTimeNoMinutes
Formatierung ohne Minuten
Wert 4 / 0x0004

Siehe Verwandte
Befehle

Durch die Verwendung von _FmtTimeNoMinutes werden bei der Umwandlung zwischen Zeit und alphanumerisch (siehe CnvAT() und CnvTA()) die Minuten nicht berücksichtigt.

Diese Option kann nur in Verbindung mit _FmtTimeNoHours verwendet werden.

`_FmtTimeNoSep`
Formatierung ohne Trennzeichen
Wert 16 /
0x0010

Siehe [Verwandte](#)
[Befehle](#)

Durch die Verwendung von `_FmtTimeNoSep` wird bei der Umwandlung von Zeit nach alphanumerisch (siehe [CnvAT\(\)](#)) kein Trennzeichen benutzt (unabhängig von der länderspezifischen Einstellung).

_FmtTimeSeconds
Formatierung mit Sekunden
Wert 1 / 0x0001

Siehe Verwandte
Befehle

Durch die Verwendung von _FmtTimeSeconds werden bei der Umwandlung von Zeit nach alphanumerisch (siehe CnvAT()) die Sekunden mit berücksichtigt.

_FmtTimeSFormat
Formatierung unter Berücksichtigung von LclTimeSFormat
Wert 4.096 / 0x1000

Verwandte

Siehe Befehle,
FmtTimeFlags,
CnvAT()

Durch die Verwendung von _FmtTimeSFormat werden bei der Umwandlung von Zeit nach alphanumerisch (siehe CnvAT()) die Formatierung der Eigenschaft LclTimeSFormat berücksichtigt.



Ist diese Option bei CnvAT() angegeben, werden alle weiteren Optionen ignoriert.

Implizite Typkonvertierung

Konvertierung von int und bigint

int,
Siehe bigint,
CnvBI(),
CnvIB()

Bei Zuweisungen, Argumenten und in Ausdrücken werden Variablen, Konstanten oder Felder vom Typ int und bigint nun implizit konvertiert, wenn der Zieltyp (int bzw. bigint) nicht dem Quelltyp entspricht.

Beispiel:

Der folgende Code wird unter der Version 5.7.11 übersetzt.

```
main() local {   tVal1   : int;   tVal2   : bigint;   tResult : int; }{ tVal1 # 1000; tVal2
```

Die Typkonvertierung muss explizit durch Angabe der entsprechenden Konvertierungsfunktion (z. B. CnvIB() für tVal2) durchgeführt werden. Mit der Version 5.8.01 ist dies zwar weiterhin möglich, jedoch nicht mehr notwendig. Die Typkonvertierung zwischen int und bigint wird durch das P-Code-System während der Laufzeit implizit durchgeführt:

```
main() local {   tVal1   : int;   tVal2   : bigint;   tResult : int; }{ tVal1 # 1000; tVal2
```

Wie zu sehen ist, hat sich die Zuweisung des Wertes -500 an die Variable tVal2 vereinfacht, da die Angabe von \b zur Kennzeichnung eines bigint-Wertes nicht mehr notwendig ist. Weiterhin entfallen die Konvertierungsfunktionen CnvBI() und CnvIB(), da auch hier tVal1 implizit nach bigint, sowie das Ergebnis des Ausdrucks (tVal1 + tVal2) implizit nach int konvertiert wird.

Die Konvertierung greift auch bei der Übergabe von Argumenten an Funktionen. Im obigen Beispiel könnte auch CnvAB()(tResult) verwendet werden, obwohl tResult vom Typ int ist. Dies funktioniert deshalb, weil eine implizite Konvertierung für tResult nach bigint durchgeführt wird, was dem formalen Argument (= Zieltyp) von CnvAB() entspricht.



Bei der Zuweisung oder Berechnung eines Wertes, der außerhalb des Wertebereiches des Zieldatentyps liegt, wird der Laufzeitfehler __ErrValueOverflow ausgelöst.

Beispiel:

```
main() local {   tVal1   : int; }{ tVal1 # 10000000000\b; // 10 Milliarden}
```

Der zugewiesene Wert liegt außerhalb des Wertebereiches des Datentyps int. Der Code lässt sich zwar fehlerfrei übersetzen. Bei der Ausführung wird jedoch der Laufzeitfehler __ErrValueOverflow ausgelöst.

Debugger-Befehle

Befehle zur Steuerung des Debuggers

Liste sortiert

nach

Siehe Gruppen,
Alphabetische

Liste aller

Befehle

Befehle

- DbgConnect
- DbgControl
- DbgDisconnect
- DbgDump
- DbgStop
- DbgTrace

Konstanten

- _DbgCallStackOff
- _DbgCallStackOn
- _DbgDumpCallStack
- _DbgEnter
- _DbgEnterOff
- _DbgEnterSub
- _DbgFrameOpenOff
- _DbgFrameOpenOn
- _DbgLeave
- _DbgOff
- _DbgStop
- _DbgTime
- _DbgTimeStart
- _DbgTraceClear

Die Funktion des Debuggers ist im Abschnitt Externer Debugger beschrieben.

DbgConnect(alpha1, logic2, logic3)



Verbindung mit Debugger aufnehmen

alpha1 Computername / IP-Adresse

logic2 Verbindungsversuchsschleife

logic3 Verbindungswiederherstellung

Verwandte Befehle,

Siehe DbgDisconnect(),

DbgControl(), DbgTrace(),

Debugger

Mit diesem Befehl wird eine Verbindung zum CONZEPT 16-Debugger aufgebaut. Der Debugger muss nicht auf dem selben Computer laufen, auf dem der CONZEPT 16-Client läuft.

Im Parameter (alpha1) wird der Name des Computers oder die IP-Adresse angegeben, auf dem der Debugger läuft. Die Kommunikation erfolgt über das Protokoll TCP/IP. Laufen Debugger und Client auf dem selben Rechner, kann die IP-Adresse auch mit '*' angegeben werden.

Die Parameter (logic2) und (logic3) steuern das Kommunikationsverhalten bei einem Verbindungsabbruch:

(logic2) (logic3) Verhalten

<u>false</u>	<u>false</u> od. <u>true</u>	Kann die Verbindung nicht aufgenommen werden, wird kein weiterer Versuch mehr unternommen.
<u>true</u>	<u>false</u>	Es wird solange versucht eine Verbindung aufzubauen, bis sie zustande kommt. Bei einem Verbindungsverlust wird die Verbindung nicht neu aufgebaut.
<u>true</u>	<u>true</u>	Es wird solange versucht eine Verbindung aufzubauen, bis sie zustande kommt. Bei einem Verbindungsverlust wird versucht diese wieder herzustellen.


Nach dem Befehl DbgDisconnect() wird keine Verbindung mehr zum Debugger aufgenommen.

Ein Client kann sich nur mit einem Debugger verbinden. Weitere DbgConnect()-Anweisungen bei bestehender Verbindung werden ignoriert.

Beispiel:

Wird DbgConnect() in einer Terminal-Server-Umgebung verwendet, sollte der externe Debugger auf der lokalen Maschine (nicht in der Terminal-Session) gestartet werden. Zum Verbinden mit der lokalen Maschine wird dann folgende Programmzeile verwendet:

```
DbgConnect(NetInfo(_NtiAddressTSC), false, false);
```

DbgControl(int1) 

Debugger steuern

int1 Optionen

Verwandte

Befehle,

Siehe DbgConnect(),

Debugger

Mit diesem Befehl können verschiedene Ausgaben des Debuggers und das Ablaufverhalten von Prozeduren und Funktionen gesteuert werden. Die Wirkung der übergebenen Konstanten sind dort beschrieben. Folgende Optionen können übergeben werden:

- **_DbgEnter**

Aufrufe durch das System (Ereignisse) werden im Debugger protokolliert. Die Protokollierung kann mit _DbgEnterOff wieder aufgehoben werden.

- **_DbgEnterSub**

Aufrufe von Funktionen werden im Debugger protokolliert. Die Option muss mit _DbgEnter und kann mit _DbgLeave kombiniert werden. Die Protokollierung kann mit _DbgEnterOff wieder aufgehoben werden.

- **_DbgEnterOff**

Es werden keine weiteren Funktionsaufrufe protokolliert.

- **_DbgLeave**

Das Beenden von Prozeduren und Funktionen im Debugger wird protokolliert. Die Option muss mit _DbgEnter und kann mit _DbgEnterSub kombiniert werden.

- **_DbgStop**

Der Debugger stoppt die Verarbeitung der Prozedur an dieser Stelle. Dies kann ebenfalls mit dem Befehl DbgStop() oder dem Setzen eines Breakpoints erreicht werden.

Die Verarbeitung der Prozedur wird angehalten und es wird eine entsprechende Benachrichtigung eingeblendet.

- **_DbgOff**

Der Debug-Modus wird aufgehoben. Weitere Aufrufe von DbgConnect(), DbgDisconnect(), DbgStop() und DbgTrace() werden ignoriert. Zusätzlich ignoriert die Funktion DbgControl() alle Optionen, außer _DbgCallStackOn und _DbgCallStackOff. Darüberhinaus wird die Geschwindigkeit der Applikation erhöht, da bei Funktionsaufrufen die interne Statusprotokollierung minimiert wird.

Die Option wird erst mit dem Beenden des Clients, oder dem Neustart des Designers aufgehoben.

Trotz deaktivieren der interen Aufrufprotokollierung von Prozeduren und Funktionen können aktuelle Prozedurinformationen mit den Compiler-Makros

ausgelesen werden.

- **_DbgTimeStart**

Die interne Stoppuhr wird gestartet bzw. zurückgesetzt. Die vergangene Zeit kann über die Option `_DbgTime` abgefragt werden.

- **_DbgTime**

Mit der Option wird die vergangene Zeit in Millisekunden seit dem letzten Aufruf mit der Option `_DbgTimeStart` in das Protokollfenster des Debuggers ausgegeben.

- **_DbgCallStackOn**

Die Protokollierung der Funktionsaufrufe wird gestartet. Die Aufrufreihenfolge kann mit der Anweisung `DbgDump()` in eine externe Datei geschrieben werden.



Die Protokollierung der Aufrufreihenfolge hat zwar nur einen geringen Einfluss auf die Ausführungsgeschwindigkeit, sollte aber nur zu Debugging-Zwecken aktiviert werden. So wird systemseitig eine maximale Performance für die Prozedurausführung erreicht.

- **_DbgCallStackOff**

Die Protokollierung der Funktionsaufrufe wird angehalten.

- **_DbgFrameOpenOn**

Aktiviert die Protokollierung für Dialog-Befehle. Protokolliert Aufrufe von `WinOpen()`, `WinDialog()`, `WinDialogRun()` und `WinAddByName()`. Sofern sich die Aufrufe auf Dialoge beziehen, werden auch die Befehle `WinAdd()` und `WinCreate()` protokolliert. In der Spalte "Ausgabe" des Protokolls wird zuerst in eckigen Klammern der Name des aufgerufenen Befehls ausgegeben. Dann folgt der Name des Dialogs. Die Spalten "Prozedur" und "Zeile" enthalten Prozedur- und Funktionsname sowie die Zeilennummer des Aufrufs.

- **_DbgFrameOpenOff**

Deaktiviert die Protokollierung für Dialog-Befehle.

- **_DbgTraceClear**

Protokoll leeren.

Vor der Ausführung von `DbgControl()` muss eine Verbindung zum Debugger vorhanden sein. Die Verbindung kann mit `DbgConnect()` oder aus dem `Editor` hergestellt werden. Für die Ausführung mit einem `_DbgCallStack...`-Parameter muss keine Verbindung zum externen Debugger vorhanden sein.

Beispiele:

```
// Alle externen Aufrufe (Ereignisse) im Debugger protokollierenDbgControl(_DbgEnter);// Alle ext
```

Kontakt

DbgDisconnect()



Verbindung zum Debugger trennen

Verwandte

Siehe Befehle,

DbgConnect(),

Debugger

Mit diesem Befehl wird die Verbindung zum Debugger unterbrochen. Es finden keine weiteren Versuche eines Verbindungsaufbaus statt. Alle weiteren Ausgaben über DbgTrace() werden ignoriert.

DbgDump(alpha1, int2)



Aufrufreihenfolge in externe Datei schreiben

alpha1 Pfad und Dateiname der externen Datei

int2 Typ der Information

_DbgDumpCallStack Aufrufreihenfolge

Siehe [Verwandte Befehle](#), [DbgControl\(\)](#),
[Debugger](#), [Laufzeitfehler finden \(Blog\)](#)

Tritt in einer häufig aufgerufenen oder tief verschachtelten Prozedur ein Laufzeitfehler auf, kann die Reihenfolge der aufgerufenen Funktionen (inklusive Prozedur und Zeilennummer der aufrufenden Prozedur) in der in (alpha1) angegebenen externen Datei gespeichert werden. In (int2) kann der Typ der Information angegeben werden. Zur Zeit kann hier nur _DbgDumpCallStack angegeben werden.

Der CallStack muss zuvor mit der Anweisung [DbgControl\(_DbgCallStackOn\)](#) gestartet werden.

Beispiel

Die folgende Prozedur wurde in der Datenbank mit dem Namen "dump" gespeichert.

```
sub sub2{ DbgDump(_Sys->spPathTemp + '\dump.txt', _DbgDumpCallStack);}sub sub1{ sub2();}main{
```

Die Datei "dump.txt" enthält nach der Ausführung folgenden Inhalt:

```
### CALL STACK DUMP BEGIN[1] dump:sub2
```

DbgStop()



Verarbeitung anhalten

Verwandte

Siehe Befehle,

DbgControl(),

Debugger

Die Verarbeitung der Prozedur wird an diesem Befehl angehalten. Die Wirkungsweise entspricht dem Befehl DbgControl(_DbgStop) bzw. einem Breakpoint an dieser Stelle. Auf dem Client-Rechner wird ein Fenster angezeigt, um die Tätigkeit des Debuggers anzuzeigen.

Kontakt

DbgTrace(alpha1)



Text im Debugger-Fenster ausgeben

alpha1 Auszugebender
Text

Verwandte

Siehe Befehle,
DbgConnect(),
Debugger

Der Text in (alpha1) wird im Protokollbereich des Debuggers ausgegeben. Zuvor muss eine Verbindung mit dem Debugger aufgebaut worden sein (z. B. über DbgConnect()).

Konstanten für Debugger-Befehle

Konstanten für Debugger-Befehle

Siehe Debugger-Befehle,
Debugger

- _DbgCallStackOff
- _DbgCallStackOn
- _DbgDumpCallStack
- _DbgEnter
- _DbgEnterOff
- _DbgEnterSub
- _DbgFrameOpenOff
- _DbgFrameOpenOn
- _DbgLeave
- _DbgOff
- _DbgStop
- _DbgTime
- _DbgTimeStart
- _DbgTraceClear

Funktionen und Operatoren

Funktionen und Operatoren

Siehe Befehlsgruppen,
Befehlsliste

- Datumsfunktionen
- Mathematische Funktionen
- Operatoren
- Zeichenkettenfunktionen
- Zeitfunktionen

Mathematische Funktionen

Mathematische Funktionen und Konstanten

Befehlsgruppen,

Siehe Befehlsliste,

Mathematische

Operatoren

Befehle

- Abs
- Acos
- Asin
- Atan
- Ceil
- Cos
- Dec
- Exp
- Floor
- Fract
- Inc
- Log10
- Log2
- LogN
- Max
- Min
- Pow
- Random
- Rescale
- Rnd
- Sgn
- Sin
- Sqrt
- Tan
- Trn

Konstanten

- MaxInt
- MinInt

Abs(float1) :

float

Abs(decimal1) :

decimal



Abs(int1) : int

Abs(bigint1) :

bigint

Betragsfunktion

float1 /

decimal1 / Ausgangswert

int1 /

bigint1


Resultat float /
decimal / Betrag
int /
bigint

Siehe Verwandte
Befehle, Sgn()

Vom Ausgangswert wird der Betrag errechnet.

Beispiele:

Abs(-15.71) // 15.71 Abs(15.71) // 15.71 Abs(0.0) // 0.0

Acos(float1) : float 

Arkuskosinusfunktion

float1 Ausgangswert (Kosinus
des Winkels)


Resultat float Arkuskosinus
(Bogenmaß)

Siehe Verwandte Befehle, Cos(),
Asin(), Atan()

Vom Ausgangswert (float1) wird der Arkuskosinus (Bogenmaß) errechnet. Gültige Ergebnisse sind für alle Ausgangswerte (float1) definiert die im Bereich [-1.0,1.0] liegen. Für andere Werte wird 0.0 zurückgegeben.

Beispiele:

```
Acos(1.0) // 0.0 Acos(0.5) // Pi / 3 Acos(0.0) // Pi / 2
```

Asin(float1) : float 

Arkussinusfunktion

float1 Ausgangswert (Sinus
des Winkels)


Resultat float Arkussinus
(Bogenmaß)

Siehe Verwandte Befehle,
Sin(), Acos(), Atan()

Vom Ausgangswert (float1) wird der Arkussinus (Bogenmaß) errechnet. Gültige Ergebnisse sind für alle Ausgangswerte (float1) definiert die im Bereich [-1.0,1.0] liegen. Für andere Werte wird 0.0 zurückgegeben.

Beispiele:

```
Asin(1.0) // Pi / 2Asin(0.5) // Pi / 6Asin(0.0) // 0.0
```

Atan(float1) : float 

Arkustangensfunktion

float1 Ausgangswert (Tangens
des Winkels)

Resultat float Arkustangens
(Bogenmaß)

Siehe Verwandte Befehle, Tan(),
Asin(), Acos()

Vom Ausgangswert (float1) wird der Arkustangens (Bogenmaß) errechnet.

Beispiele:

```
Atan(Sqrt(3.0)) // Pi / 3  Atan(1.0) // Pi / 4  Atan(0.0) // 0.0
```

Ceil(float1) : float

Ceil(decimal1) :



decimal

Aufrundungsfunktion

float1 /

decimal1 Ausgangswert

Resultat float / Nächst
decimal größere
 ganze Zahl



Siehe Verwandte Befehle,
Floor()

Diese Funktion liefert die nächste ganze Zahl, die größer oder gleich dem Parameter ist.

Beispiele:

Ceil(-2.1) // -2.0 Ceil(-2.0) // -2.0 Ceil(0.0) // 0.0 Ceil(2.0) // 2.0 Ceil(2.1) // 3.0

Cos(float1) : 
 float
 Cosinusfunktion

float1 Ausgangswert
 (Bogenmaß)

Resultat float Cosinus

Verwandte

Siehe Befehle,
Acos(), Sin(),
Tan()

Vom Ausgangswert (float1 = Bogenmaß) wird der Cosinus errechnet.

Beispiele:

```
Pi # 3.141592653590;Cos(Pi)           // -1.0Cos(2.0 * Pi)    // 1.0Cos(0.0)           // 1.0
```

Dec(var int1[, int2])
Dekrementierungsfunktion



Ganzzahlige
int1 Variable oder
Feld
Zu
int2 subtrahierender
Wert (optional)

Siehe [Verwandte](#)
[Befehle](#), [Inc\(\)](#)

Mit dieser Anweisung kann der Wert eines Feldes oder einer Variablen um 1 (int2 nicht vorhanden) oder um (int2) vermindert werden.

Beispiele:

```
tInt1 # 8; // ganzzahlige Variable
tInt2 # -1; // ganzzahlige Variable
iFeld # 10; // ganzzahliges Feld
```

Exp(float1) : float

Exp(decimal1) :



decimal

Exponentialfunktion

float1 /

decimal1 Exponent

Resultat float / Potenz zur
decimal Basis e

Siehe Verwandte Befehle,
LogN(), Pow()

Das Resultat entspricht $e^{(\text{float1} / \text{decimal1})}$.

Beispiele:

Exp(1.0) // 2.7183Exp(2.5) // 12.1825Exp(0) // 1.0

Floor(float1) : float

Floor(decimal1) :




decimal

Abrundungsfunktion

float1 /

decimal1 Ausgangswert

Resultat float / Nächst
 decimal kleinere 
 ganze Zahl

Siehe Verwandte Befehle,
 Ceil()

Diese Funktion liefert die nächste ganze Zahl, die kleiner oder gleich dem Parameter ist.

Beispiele:

Floor(-3.14) // -4.0Floor(-3.0) // -3.0Floor(0.0) // 0.0Floor(3.0) // 3.0Floor(3.14) /

Fract(float1[,int2]) : float

Fract(decimal1[,int2]) :



decimal

Nachkommastellenfunktion

float1 /

decimal1 Ausgangswert

int2 Nachkommastelle

Resultat float /
decimal Nachkommastellen


Siehe Verwandte Befehle

Mit diesem Befehl kann der Nachkomma-Teil eines Wertes komplett oder teilweise ermittelt werden.

Ab der in (int2) angegebenen Nachkommastelle wird der Wert ermittelt.

Beispiele:

Fract(17.56781, 0) // 0.56781 Fract(17.56781, 1) // 0.06781 Fract(17.56781, 2) // 0.00781 Fract(17.56781, 3) // 0.000781

Inc(var int1[, int2]) 

Inkrement-Funktion

Ganzzahlige

int1 Variable
oder Feld

Zu

int2 addierender
Wert

(optional)

Verwandte

Siehe Befehle,

Dec()

Mit dieser Anweisung kann der Wert eines Feldes oder einer Variablen um 1 (int2 nicht vorhanden) oder um (int2) erhöht werden.

Beispiele:

```
tInt1 # 8; // ganzzahlige Variable
tInt2 # -1; // ganzzahlige Variable
iFeld # 10; // ganzzahliges Feld
```

Log2(float1) : float

Log2(decimal1) : decimal



Logarithmusfunktion (Basis 2)

float1 /
decimal1 Ausgangswert

Resultat float / Logarithmus
 decimal (Basis 2)

Siehe Verwandte Befehle, Log10(),
 LogN(), Pow()

Vom Ausgangswert (float1 / decimal1) wird der Logarithmus zur Basis 2 errechnet.

Beispiele:

Log2(32.0) // 5.0Log2(256.0) // 8.0Log2(1.0) // 0.0

Mögliche Laufzeitfehler:

ErrMathArgument (float1 / decimal1) ist kleiner oder gleich 0

Log10(float1) : float

Log10(decimal1) : decimal



Logarithmusfunktion (Basis 10)

float1 /
decimal1 Ausgangswert

Resultat float / Logarithmus
 decimal (Basis 10)

Siehe Verwandte Befehle, Log2(),
 LogN(), Pow()

Vom Ausgangswert (float1 / decimal1) wird der Logarithmus zur Basis 10 errechnet.

Beispiele:

Log10(100.0) // 2.0Log10(10000.0) // 4.0Log10(1.0) // 0.0

Mögliche Laufzeitfehler:

ErrMathArgument (float1 / decimal1) ist kleiner oder gleich 0

LogN(float1) : float

LogN(decimal1) : decimal



Logarithmusfunktion (Basis e)

float1 /
decimal1 Ausgangswert

Resultat float / Logarithmus
 decimal (Basis e)

Siehe Verwandte Befehle, Log10(),
 Log2(), Pow(), Exp()


Vom Ausgangswert (float1 / decimal1) wird der Logarithmus zur Basis e errechnet.

Beispiele:

LogN(2.5) // 0.9163 LogN(10.0) // 2.3026 LogN(1.0) // 0.0

Mögliche Laufzeitfehler:

ErrMathArgument (float1 / decimal1) ist kleiner oder gleich 0

Max(var1, var2) : 

var

Maximum-Funktion

var1 1. Wert

var2 2. Wert (Typ wie (var1))

Resultat var Maximum (Typ wie
(var1))


Siehe Verwandte Befehle,
Min()

Diese Funktion liefert den größeren der beiden übergebenen Werte zurück. Beide Werte müssen vom gleichen Typ sein. Der Rückgabewert hat den gleichen Typ, wie die übergebenen Werte.

Beim Vergleich von Zeichenketten wird der ASCII-Wert der Zeichen verwendet.

Beispiele:

Max(12, 25) // 25 Max(-0.6, -10.0) // -0.6 Max('D', 'H') // 'H' Max('a', 'Z') // 'a'


Min(var1, var2) : 
 var
 Minimum-Funktion
 var1 1. Wert
 var2 2. Wert (Typ wie (var1))
 Resultat var Minimum (Typ wie
 (var1))

Siehe Verwandte Befehle,
Max()

Diese Funktion liefert den kleineren der beiden übergebenen Werte zurück.

Beispiele:

Min(12, 25) // 12Min(-0.6, -10.0) // -10.0Min('D', 'H') // 'D'

Pow(float1,
float2) : float
Pow(decimal1, ,
decimal2) :
decimal

Potenzfunktion

float1 /
decimal1 Basis

float2 /
decimal2 Exponent

Resultat float /
 decimal Potenz

Siehe Verwandte
 Befehle, Log10(),
 Log2(), LogN(),
 Sqrt()

Das Resultat entspricht (float1 / decimal1)^(float2 / decimal 2).

Beispiele:

Pow(3.0, 4.0) // 81.0Pow(17.3, 1.0) // 17.3Pow(1.0, 34.67) // 1.0Pow(2.0, 0.0) // 1.0

Random() : 

float

Zufallsfunktion


Resultat float Zufallszahl

Siehe Verwandte
Befehle, Rnd()

Diese Funktion liefert einen Zufallswert im Bereich zwischen Null und Eins (exklusive der Grenzen) mit 12 Nachkommastellen Genauigkeit.

Beispiele:

```
// Werte zwischen 0 und 1 (exklusive der Grenzen)Random() // 0.786247683765Random() // 0.18237787
```

Rescale(decimal1, )
int2) : decimal

Komma verschieben

decimal1 Ausgangswert

int2 Anzahl der Stellen

Wert mit

Resultat decimal verschobenem

Komma

Siehe Verwandte Befehle

Mit diesem Befehl wird der Wert (decimal1) mit 10er-Potenzen multipliziert (int2 > 0) oder dividiert (int2 < 0). Wird der maximale Wertebereich des decimal-Typs überschritten, ist das Resultat DecimalError (gegebenenfalls erfolgt der Laufzeitfehler ErrValueInvalid). Der Befehl führt intern eine Kommaverschiebung ohne weitere Berechnungen durch.

Beispiele:

Rescale(17.43\m, 1) // 174.3 Rescale(17.43\m, 3) // 17430 Rescale(17.43\m, -2) // 0.1743 Rescale(1

Mögliche Laufzeitfehler:

ErrValueInvalid Resultat überschreitet Dezimal-Wertebereich

Rnd(float1[,int2])

: float

Rnd(decimal1[,int2])

: decimal

Rundungsfunktion

float1 /

decimal1 Ausgangswert

int2 Anzahl der
 verbleibende
 Nachkommastellen
 (optional)

Resultat float / Gerundeter
 decimal Wert

Siehe Verwandte Befehle,
 Trn()

Diese Funktion rundet den übergebenen Wert kaufmännisch.

In (int2) kann eine Anzahl von Nachkommastellen angegeben werden, auf die gerundet werden soll. Wird (int2) nicht angegeben oder ist (int2) gleich 0, wird auf einen ganzzahligen Wert gerundet. Je nach Typ kann maximal 7 (bei float) bzw. 100 (bei decimal) angegeben werden.

Beispiel:

```
Rnd(145.6673)        // 146 Rnd(145.6673, 2) // 145.67
```

Mögliche Laufzeitfehler:

ErrValueRange (int2) ist kleiner als 0 oder größer 7 (bei float) bzw. 100 (bei decimal)

Sgn(float1) : int

Sgn(decimal1) :

int



Sgn(int1) : int

Sgn(bigint1) :

int

Signumfunktion

float1 /

decimal1 / int1 Ausgangswert

/ bigint1

Resultat int Vorzeichen

Siehe Verwandte

Befehle, Abs()

Die Signumfunktion liefert das Vorzeichen des in (float1 / decimal1 / int1 / bigint1) übergebenen Wertes.

Beispiele:

Sgn(-87.827) // -1 Sgn(39.002) // 1 Sgn(0) // 0

Sin(float1) : 

float

Sinusfunktion

float1 Ausgangswert
(Bogenmaß)

Resultat float Sinus

Verwandte

Siehe Befehle,
Asin(), Cos(),
Tan()

Vom Ausgangswert (float1 = Bogenmaß) wird der Sinus errechnet.

Beispiele:

Pi # 3.141592653590;

Sin(Pi) // 0.0

Sin(Pi / 2.0) // 1.0

Sin(0.0) // 0.0

Sqrt(float1) : float

Sqrt(decimal1) :



decimal

Quadratwurzelfunktion

float1 /

decimal1 Ausgangswert

Resultat float /
decimal Quadratwurzel

Siehe Verwandte Befehle,
Pow()

Vom Ausgangswert (float1 / decimal1) wird die Quadratwurzel errechnet.

Beispiele:

Sqrt(100.0) // 10.0Sqrt(64.0) // 8.0Sqrt(2.0) // 1.4142

Mögliche Laufzeitfehler:

ErrMathArgument (Float1) ist kleiner 0

Tan(float1) :



float

Tangensfunktion

float1 Ausgangswert
(Bogenmaß)

Resultat float Tangens

Verwandte

Siehe Befehle,
Atan(), Cos(),
Sin()

Vom Ausgangswert (float1 = Bogenmaß) wird der Tangens errechnet.

Beispiele:

Pi # 3.141592653590;

Tan(Pi) // 0.0

Tan(0.75 * Pi) // -1.0

Tan(0.0) // 0.0

Trn(float1[,int2]) : float

Trn(decimal1[,int2]) : decimal



Nachkommastellen abschneiden

float1 /
decimal1 Ausgangswert

int2 Nachkommastellen (optional)

Resultat float / Abgeschnittener
 decimal Wert

Siehe Verwandte Befehle, Rnd()

Diese Funktion schneidet die Nachkommastellen des Arguments ab. Optional kann eine Nachkommastelle in (int2) angegeben werden, nach der abgeschnitten wird.

Beispiele:

Trn(17.56781, 0) // 17.000000 Trn(17.56781, 1) // 17.500000 Trn(17.56781, 2) // 17.560000 Trn(17.56781, 3) // 17.567000

Konstanten des Integer-Wertebereichs
Konstanten des Integer-Wertebereichs

Siehe Befehlsgruppen,
Befehlsliste

- MaxInt
- MinInt

MaxInt
Größter Integer-Wert

Verwandte

Siehe Befehle,
int,

MinInt

Wert : 2.147.483.647 / 0x7FFFFFFF

MinInt
Kleinster Integer-Wert

Verwandte
Siehe Befehle,
int,
MaxInt

Wert : -2.147.483.647 / 0x80000001

Zeichenkettenfunktionen

Zeichenkettenfunktionen

Siehe Befehlsgruppen,

Befehlsliste

Befehle

- Ähnlichkeitsoperatoren (=* / =*^)
- StrAdj
- StrChar
- StrCnv
- StrCut
- StrDecrypt
- StrDel
- StrEncrypt
- StrFind
- StrFindRegEx
- StrFmt
- StrIns
- StrLen
- StrToChar

Konstanten

- _StrAll
- _StrBegin
- _StrCaseIgnore
- _StrEnd
- _StrFindReverse
- _StrFindToken
- _StrFromANSI
- _StrFromBase64
- _StrFromHTML
- _StrFromOEM
- _StrFromOEM_852
- _StrFromURI
- _StrFromUTF8
- _StrLetter
- _StrLetterExt
- _StrLower
- _StrLower1252
- _StrNameNoDouble
- _StrNameNoFirst
- _StrNameReorder
- _StrSoundex1
- _StrSoundex2
- _StrToANSI
- _StrToBase64
- _StrToHTML
- _StrToOEM
- _StrToOEM_852

- StrToURI
- StrToUTF8
- StrUmlaut
- StrUpper
- StrUpper1252

Ähnlichkeitsoperatoren (=* / =*^)

Vergleicht Zeichenketten auf Ähnlichkeit

Siehe Verwandte Befehle,
Vergleichsoperatoren

Mit den Ähnlichkeitsoperatoren können zwei alphanumerische Werte miteinander verglichen werden. Einer der Werte kann die Wildcards '?' und / oder '*' enthalten. Das '?' steht für genau ein beliebiges Zeichen. Das '*' steht für eine unbestimmte Anzahl beliebiger Zeichen.

Beispiele:

Ausdruck	Ergebnis
('Auftrag' =* 'Auftrag')	<u>true</u>
('Auftrag' =* '?uft*')	<u>true</u>
('Auftrag' =* '*tra*')	<u>true</u>
('Auftrag' =* 'Auftra??')	<u>false</u>
('Kleopatra' =* '*tra*')	<u>true</u>
('Kleopatra' =* '*le*pat*')	<u>true</u>
('89089089' =* '89*89')	<u>true</u>

Soll bei dem Vergleich die Groß-/Kleinschreibung nicht berücksichtigt werden, muss der Operator =*^ verwendet werden.

Beispiele:

Ausdruck	Ergebnis
('Kleopatra' =*^ '*tRa*')	<u>true</u>
('Kleopatra' =*^ '*Le*Pat*')	<u>true</u>
('AaAa' =*^ 'aA*')	<u>true</u>



StrAdj(alpha1, int2) : alpha

Alphawert justieren (Leerzeichen entfernen)

alpha1 Ausgangswert

Optionen

_StrBegin Entfernt die
Leerzeichen am
Anfang

int2 _StrEnd Entfernt die
Leerzeichen am
Ende

_StrAll Entfernt alle
Leerzeichen

Resultat alpha Justierte Zeichenkette

Siehe Verwandte Befehle,
StrDel(), StrFmt()

Mit dieser Funktion können Leerzeichen aus einem Alphawert entfernt werden.

Beispiele:

```
StrAdj(' Ein Test ', _StrBegin)
```

```
// 'Ein Test 'StrAdj(' Ein Test ', _StrEnd)
```



StrChar(int1,[int2]) : alpha
 Alphawert aus ASCII-Code bilden

int1 ASCII-Code

int2 Anzahl der Zeichen
 (optional)

Resultat alpha Zeichen(-folge)

Siehe Verwandte Befehle,
StrToChar()

Mit StrChar() wird aus dem ASCII-Code (int1) ein Alphawert gebildet. Mit (int2) kann angegeben werden, mit wievielen Zeichen der Alphawert gefüllt werden soll.


In (int1) können Werte von 0 bis 255 angegeben werden.

Beispiele:

```
StrChar(65)      // 'A' StrChar(98, 5) // 'bbbbb'
```

Mögliche Laufzeitfehler:

__ErrValueRange (int1) ist kleiner als 0 oder größer als 255
 (int2) ist kleiner als 0 oder größer als 65520

StrCnv(alpha1, int2) : 

alpha

Alphawert umwandeln

alpha1 Ausgangswert

int2 Optionen (siehe Text)

Resultat alpha Umgewandelte
Zeichenkette

Siehe Verwandte Befehle

Mit StrCnv() kann der Alphawert (alpha1) umgewandelt werden. Die Zeichensatzumwandlungen sind vor allem bei Datenimport und -export über die Web-Schnittstelle und die Socket-Befehle sinnvoll.

In (int2) können folgende Optionen angegeben werden:

<u>StrToOEM</u>	Nach OEM-Zeichensatz wandeln
<u>StrToANSI</u>	Nach ANSI-Zeichensatz wandeln
<u>StrToHTML</u>	Nach Unicode und HTML-Kodierung wandeln
<u>StrToURI</u>	Nach UTF-8 und URI-Kodierung wandeln
<u>StrToUTF8</u>	Nach UTF-8-Zeichensatz wandeln
<u>StrToBase64</u>	Nach Base64 kodieren
<u>StrFromOEM</u>	Von OEM-Zeichensatz wandeln
<u>StrFromANSI</u>	Von ANSI-Zeichensatz wandeln
<u>StrFromHTML</u>	Von HTML-kodiertem Unicode wandeln
<u>StrFromURI</u>	Von URI-kodiertem UTF-8 wandeln
<u>StrFromUTF8</u>	Von UTF-8-Zeichensatz wandeln
<u>StrFromBase64</u>	Aus Base64 dekodieren
<u>StrUpper</u>	a-z, ä, ö, ü in Großbuchstaben wandeln
<u>StrUpper1252</u>	Alle Kleinbuchstaben der Windows-Codepage 1252 in Großbuchstaben wandeln
<u>StrLower</u>	In Kleinbuchstaben wandeln
<u>StrLower1252</u>	Alle Großbuchstaben der Windows-Codepage 1252 in Kleinbuchstaben wandeln
<u>StrUmlaut</u>	Umlaute wandeln
<u>StrLetter</u>	Sonderzeichen entfernen
<u>StrLetterExt</u>	Sonderzeichen entfernen (erweitert)
<u>StrSoundex1</u>	Soundex Stufe 1
<u>StrSoundex2</u>	Soundex Stufe 2
<u>StrNameReorder</u>	Namensumstellung
<u>StrNameNoFirst</u>	Kein Vorname
<u>StrNameNoDouble</u>	Kein doppelter Nachname

Die Optionen können untereinander kombiniert werden.



Eine Kombination von Konstanten aus dem Bereichen StrFrom... und StrTo... ist nicht möglich. Für die Optionen StrFromBase64 und StrToBase64 gilt diese Einschränkung nicht. Somit kann beispielsweise die Option StrFromBase64 mit

der Option _StrToANSI kombiniert werden. Eine Kombination der Optionen _StrFromUTF8 und _StrToANSI ist jedoch nicht möglich.

Beim Schreiben von Daten in eine externe Datei über die Dateibefehle findet automatisch eine Wandlung vom CONZEPT 16-Zeichensatz in den OEM-Zeichensatz statt.

Beispiele:

```
StrCnv('KLEIN', _StrLower) // 'klein' StrCnv('größer', _StrUmlaut)
```

Mögliche Laufzeitfehler:

<u>_ErrValueInvalid</u>	Bei der Option (int2) <u>_StrFromBase64</u> ist die Ausgangszeichenkette nicht valide Base64-kodiert.
	Bei Option (int2) <u>_StrFrom...</u> und <u>_StrTo...</u> konnte die Zeichenkette nicht gewandelt werden.
<u>_ErrStringOverflow</u>	Bei der Option (int2) <u>_StrToBase64</u> , wenn die Zielzeichenkette größer als 65.520 Zeichen ist.



StrCut(alpha1, int2, int3) : alpha
 Zeichenfolge aus einem Alphawert kopieren
 alpha1 Ausgangswert
 int2 Erstes zu kopierendes
 Zeichen
 int3 Anzahl zu kopierender
 Zeichen

Resultat alpha Teil-Zeichenkette

Siehe [Verwandte Befehle](#)

Mit dieser Funktion kann aus einem Wert (alpha1) ein Teil kopiert werden. Aus dem Wert (alpha1) wird ab der Stelle (int2) kopiert, und zwar eine Anzahl von (int3) Zeichen. Das Resultat besteht aus dem kopierten Teilstring.

Ist (int2) kleiner 1, wird ab dem ersten Zeichen kopiert. Ist (int2) größer als die Länge von (alpha1) oder ist (int3) kleiner 1, ist das Resultat leer. Ist (int3) größer als die Anzahl der verfügbaren Zeichen, so werden alle verfügbaren Zeichen kopiert.

Beispiele:

```
StrCut('Papierkorb', 3, 4)    // 'pier' StrCut('Papierkorb', -10, 6) // 'Papier' StrCut('Papierkorb', 10, 10) // 'Papierkorb'
```



StrDecrypt(alpha1[, alpha2]) : alpha
 Verschlüsselte Zeichenkette entschlüsseln
 alpha1 Verschlüsselte Zeichenkette
 alpha2 Schlüssel (optional)

Resultat alpha Entschlüsselte Zeichenkette

Siehe Verwandte Befehle, StrEncrypt()

Mit diesem Befehl wird die in (alpha1) übergebene Zeichenkette entschlüsselt. Wurde bei der Verschlüsselung mit StrEncrypt() ein Schlüssel angegeben, muss der gleiche Schlüssel in (alpha2) angegeben werden.

Konnte die Zeichenkette nicht entschlüsselt werden, wird eine leere Zeichenkette zurückgegeben.

Beispiele:

```
// Der interne Schlüssel wird verwendet  
Decoded # StrDecrypt(aCoded); // Der Schlüssel hgi/opm52nh
```

Mögliche Laufzeitfehler:

ErrValueRange (alpha2) ist länger als 64 Zeichen



StrDel(alpha1, int2, int3) : alpha

Zeichenfolge aus einem Alphawert löschen

alpha1 Ausgangswert

int2 Erstes zu löschendes Zeichen

int3 Anzahl zu löschender Zeichen

Resultat alpha Verkürzte Zeichenkette


Siehe [Verwandte Befehle](#), [StrIns\(\)](#)

Mit dieser Funktion kann aus einem Alphawert eine bestimmte Anzahl von Zeichen gelöscht werden. Beim Wert (alpha1) werden ab der Stelle (int2) die Anzahl (int3) Zeichen gelöscht. Es können mehr Zeichen in (int3) angegeben werden, als vorhanden sind.

Ist (int2) kleiner 1, so wird ab dem ersten Zeichen gelöscht. Ist (int2) größer als die Länge von (alpha1) oder (int3) kleiner 1, so wird nichts gelöscht.

Beispiele:

```
StrDel('Papierkorb', 3, 4)    // 'Pakorb'
StrDel('Papierkorb', -10, 6) // 'korb'
StrDel('Papierkorb', 0, 0)   // 'Papierkorb'
```

StrEncrypt(alpha1, int2[, 
alpha3]) : alpha

Zeichenkette verschlüsseln

alpha1 Zu verschlüsselnde Zeichenkette

int2 Länge der zu verschlüsselnden
Zeichenkette

alpha3 Schlüssel (optional)


Resultat alpha Verschlüsselte
Zeichenkette

Siehe Verwandte Befehle, StrDecrypt()

Mit diesem Befehl wird die in (alpha1) übergebene Zeichenkette verschlüsselt. Es können Zeichenketten mit einer Länge bis zu 3000 Zeichen übergeben werden. In (int2) wird die Länge der zu verschlüsselnden Zeichenkette angegeben. Wird hier 0 oder ein Wert kleiner als die Länge der übergebenen Zeichenkette übergeben, wird die tatsächliche Länge von (alpha1) verwendet. Bei größeren Werten wird die Zeichenkette vor der Verschlüsselung auf die angegebene Länge verlängert.

Eine leere Zeichenkette kann nicht verschlüsselt werden; auch dann nicht, wenn in (int2) eine Länge vorgegeben wird.

In (alpha3) kann ein bis zu 64 Zeichen langer Schlüssel übergeben werden. In diesem Schlüssel können alle Zeichen (ASCII-Code 1 bis 255) verwendet werden. Der Schlüssel sollte möglichst lang sein und nicht im Klartext in der Prozedur angegeben werden. Ein Schlüssel kann ebenfalls mit StrEncrypt() verschlüsselt werden. Wird der Schlüssel nicht angegeben, verwendet das System einen internen Schlüssel.

 Bei der Verschlüsselung werden zufällige Komponenten mit einbezogen. Wird die gleiche Zeichenkette zweimal verschlüsselt, entstehen somit zwei unterschiedliche verschlüsselte Zeichenketten.


Die verschlüsselten Zeichenketten können mit StrDecrypt() wieder entschlüsselt werden.

Beispiele:

```
// Interner Schlüssel wird verwendet # StrEncrypt('Geheime Information', 0); // Der Schlüssel
```

Mögliche Laufzeitfehler:

ErrValueRange Die angegebene Zeichenkette ist leer oder länger als 3000 Zeichen,
der Schlüssel in (alpha3) ist länger als 64 Zeichen

StrFind(alpha1, alpha2, int3[,int4]) : int 

Alphawert in einer Zeichenkette suchen

alpha1 Zu durchsuchende Zeichenkette

alpha2 Suchwert

int3 Startposition

Optionen (optional)

StrCaseIgnore Keine Unterscheidung
zwischen
Groß-/Kleinschreibung

int4 StrFindReverse Suche vom Ende der
Zeichenkette bis zur
Startposition

StrFindToken Begriffsorientierte
Suche

Resultat int Position des Suchwerts

Siehe Verwandte Befehle, StrFindRegEx()

Diese Funktion durchsucht (alpha1) nach dem Wert (alpha2). Sofern (alpha2) in (alpha1) vorhanden ist, wird als Resultat die Position von (alpha2) in (alpha1) zurückgeliefert. Wurde (alpha2) nicht gefunden, ist das Resultat 0.

Mit (int3) wird das Intervall für die Suche definiert. Die Suche beginnt ab dieser Position. Wird die Zeichenkette von hinten nach vorne durchsucht, endet die Suche bei der angegebenen Position.

Folgende Optionen (int4) sind zulässig:

- StrCaseIgnore

Bei der Suche wird die Groß/-Kleinschreibung nicht beachtet.

- StrFindReverse

Die Suche beginnt am Ende der Zeichenkette und endet an der Position (int3).

- StrFindToken

Die Suchergebnisse beschränken sich auf ganze Wörter.



Wird als Suchstring eine leere Zeichenkette angegeben, wird diese in jedem Fall gefunden.

Beispiele:

```
// Ermitteln der Zeichenkette zwischen zwei ZeichentPosStart # StrFind(tString, '%', 0);tPosEnd
```

StrFindRegEx(alpha1, alpha2, int3[, int4[, var
int5]]) : int



Regulären Ausdruck in einer Zeichenkette suchen

alpha1 Zu durchsuchende Zeichenkette

alpha2 Regulärer Ausdruck

int3 Startposition

Optionen (optional)

int4 _StrCaseIgnore Keine Unterscheidung
zwischen
Groß-/Kleinschreibung

var int5 Länge der gefundenen Zeichenkette
(optional)

Resultat int Position des Suchwerts



Siehe Verwandte Befehle, StrFind()

Diese Funktion durchsucht (alpha1) mit Hilfe des regulären Ausdrucks (alpha2).

Sofern eine Entsprechung von (alpha2) in (alpha1) vorhanden ist, wird als Resultat die Position der Entsprechung in (alpha1) zurückgeliefert. Wurde (alpha2) nicht gefunden, ist das Resultat 0.

Mit (int3) wird das Intervall für die Suche definiert. Die Suche beginnt ab dieser Position.

Folgende Optionen (int4) sind zulässig:

- _StrCaseIgnore

Bei der Suche wird die Groß-/Kleinschreibung nicht beachtet.

In (int5) wird, sofern eine Variable angegeben wurde, die Länge der gefundenen Zeichenkette zurück gegeben.



Wird als Suchstring eine leere Zeichenkette angegeben, wird diese in jedem Fall gefunden.

Beispiele:

```
// Ermitteln, ob das Wort RecInsert oder RecReplace in der Zeichenkette ohne Beachtung der Groß-/
```

Fehlerwerte:

Folgende Fehlerwerte können von dem Befehl zurückgegeben werden:

Fehlerwert

ErrRegExRuleSyntax

ErrRegExBadEscapeSequence

ErrRegExPropertySyntax

ErrRegExNotSupported

ErrRegExMismatchedParentheses

Bedeutung

Syntaxfehler im regulären Ausdruck

Nicht aufgelöste Escape-Sequenz im Ausdruck


Ungültige Unicode-Eigenschaft

Verwendung einer Funktion, die nicht unterstützt wird

Falsch verschachtelte Klammern im regulären Ausdruck

Kontakt

<u>ErrRegExNumberTooBig</u>	Dezimalzahl zu groß
<u>ErrRegExBadInterval</u>	Fehler im {min,max} Intervall
<u>ErrRegExMaxLtMin</u>	Im Intervall {min,max} ist max kleiner als min
<u>ErrRegExInvalidBackRef</u>	Rückbezug auf eine nicht vorhandene Referenz
<u>ErrRegExInvalidFlag</u>	Ungültiger Modus
<u>ErrRegExLookBehindLimit</u>	Rückschau Ausdrücke müssen eine beschränkte maximale Länge haben
<u>ErrRegExSetContainsString</u>	Reguläre Ausdrücke können keine UnicodeSets mit Zeichenketten beinhalten
<u>ErrRegExMissingCloseBracket</u>	Fehlende schließende Klammer in einem Klammersausdruck
<u>ErrRegExInvalidRange</u>	In einer Zeichenmenge [x-y] ist x größer als y
<u>ErrRegExStackOverflow</u>	Stapelüberlauf in der Ablaufverfolgung des regulären Ausdrucks

StrFmt(alpha1, int2, 
int3) : alpha

Alphawert formatieren

alpha1 Ausgangswert

int2 Gewünschte Länge

Optionen

_StrBegin Zeichenkette
nach vorne

int3 auffüllen

_StrEnd Zeichenkette
nach hinten
auffüllen

Resultat alpha Formatierte
Zeichenkette

Siehe Verwandte Befehle,
StrAdj(), StrDel(), StrIns()

Diese Funktion bringt (alpha1) auf die neue Länge (int2). Durch (int3) wird bestimmt, wie dies geschieht. Wird in (int3) _StrBegin angegeben, so werden entweder Leerzeichen am Anfang eingefügt oder Zeichen am Anfang entfernt. Bei _StrEnd werden entweder Leerzeichen an (alpha1) angehängen oder Zeichen am Ende entfernt.

Ist (int2) kleiner 1, so ist das Resultat leer. Ist (int2) größer als 65520, hat das Resultat 65520 Stellen.

Beispiele:

```
StrFmt('Laufwerk', 10, _StrBegin) // ' Laufwerk' StrFmt('Laufwerk', 10, _StrEnd) // 'Laufwerk'
```

StrIns(alpha1, alpha2, int3) : alpha
Zeichenfolge in einen Alphawert einfügen



alpha1 Ausgangswert

alpha2 Einzufügende Zeichenfolge

int3 Einfügeposition

Resultat alpha Erweiterte Zeichenkette

Siehe Verwandte Befehle, StrCut(),
StrDel()

Mit dieser Funktion kann ein Alphawert in einen anderen Alphawert eingefügt werden. In den Wert (alpha1) wird der Wert (alpha2) vor der Stelle (int3) eingefügt.

Ist (int3) kleiner 1, so wird vor dem ersten Zeichen eingefügt. Ist (int3) größer als die Länge von (alpha1), so wird (alpha2) an (alpha1) angehängen.

Beispiel:

```
StrIns('Testdruck', 'aus', 5) // 'Testausdruck'
```

Mögliche Laufzeitfehler:

ErrValueRange Die Gesamtlänge von (alpha1) und (alpha2) überschreitet die Definitionsgrenze von Zeichenketten (65.520 Zeichen).

Kontakt



StrLen(alpha1) : int

Länge einer Zeichenkette ermitteln

alpha1 Ausgangswert

Resultat int Anzahl der Zeichen

Siehe [Verwandte Befehle](#)

Mit dieser Funktion kann die Länge einer Zeichenkette (alpha1) ermittelt werden.

Beispiele:

```
StrLen('Zeichenkette') // 12StrLen('1 2 3')           // 5StrLen('')           // 0
```


StrToChar(alpha1, int2) : int



ASCII-Code aus einem Alphawert ermitteln

alpha1 Ausgangswert

int2 Zeichenposition

Resultat int ASCII-Code

Verwandte

Siehe Befehle,

StrChar()

Diese Funktion ermittelt den ASCII-Code des Zeichens (int2) in (alpha1). Ist (int2) kleiner 1 oder größer als die Länge von (alpha1), wird 0 zurückgeliefert.

Beispiele:

```
StrToChar('ABCDE', 1) // 65StrToChar('ABCDE', 3) // 67StrToChar('ABCDE', 10) // 0
```

Konstanten für Zeichenkettenfunktionen
 Konstanten für Zeichenkettenfunktionen
 Siehe Zeichenkettenfunktionen

<u>StrAll</u>	Komplette Zeichenkette
<u>StrBegin</u>	Anfang einer Zeichenkette
<u>StrCaseIgnore</u>	Suche ohne Beachtung von Groß-/Kleinschreibung
<u>StrEnd</u>	Ende einer Zeichenkette
<u>StrFindReverse</u>	Zeichenkette von hinten nach vorne durchsuchen
<u>StrFindToken</u>	Begriffsorientierte Suche
<u>StrFromOEM</u>	Von OEM-Zeichensatz wandeln
<u>StrFromOEM_852</u>	Von OEM-Zeichensatz (osteuropäisch) wandeln
<u>StrFromANSI</u>	Von ANSI-Zeichensatz wandeln
<u>StrFromURI</u>	Von URI-kodiertem UTF-8-Zeichensatz wandeln
<u>StrFromUTF8</u>	Von UTF-8-Zeichensatz wandeln
<u>StrToOEM</u>	Nach OEM-Zeichensatz wandeln
<u>StrToOEM_852</u>	Nach OEM-Zeichensatz (osteuropäisch) wandeln
<u>StrToANSI</u>	Nach ANSI-Zeichensatz wandeln
<u>StrToURI</u>	Nach URI-kodiertem UTF-8-Zeichensatz wandeln
<u>StrToUTF8</u>	Nach UTF-8-Zeichensatz wandeln
<u>StrUpper</u>	In Großbuchstaben wandeln
<u>StrUpper1252</u>	In Großbuchstaben wandeln (Windows-Codepage 1252)
<u>StrLower</u>	In Kleinbuchstaben wandeln
<u>StrLower1252</u>	In Kleinbuchstaben wandeln (Windows-Codepage 1252)
<u>StrUmlaut</u>	Umlaute wandeln
<u>StrLetter</u>	Sonderzeichen entfernen
<u>StrLetterExt</u>	Sonderzeichen entfernen (erweitert)
<u>StrSoundex1</u>	Soundex Stufe 1
<u>StrSoundex2</u>	Soundex Stufe 2
<u>StrNameReorder</u>	Namensumstellung
<u>StrNameNoFirst</u>	Kein Vorname
<u>StrNameNoDouble</u>	Kein doppelter Nachname

StrAll
Entfernt alle Leerzeichen aus einem alpha-Wert
Wert 4 / 0x04

Verwandte

Siehe Befehle,

StrAdj()

Option bei StrAdj() wodurch alle Leerzeichen in einer Zeichenkette entfernt werden.

_StrBegin

Bezeichner für den Anfang einer Zeichenkette

Wert 2 / 0x02

Verwandte

Siehe Befehle,

StrAdj(),

StrFmt()

Option bei StrAdj() und StrFmt() die den Anfang einer Zeichekette bezeichnet.

StrCaseIgnore
Zeichenkette ohne Groß-/Kleinunterscheidung durchsuchen
Wert 1 / 0x01

Verwandte

Siehe Befehle,

StrFind()

Option bei StrFind() und StrFindRegEx(), um eine Zeichenkette ohne Groß-/Kleinunterscheidung zu durchsuchen.

_StrEnd

Bezeichner für das Ende einer Zeichenkette

Wert 1 / 0x01

Verwandte

Siehe Befehle,

StrAdj(),

StrFmt()

Option bei StrAdj() und StrFmt() die das Ende einer Zeichekette bezeichnet.

_StrFindReverse

Start der Begriffssuche am Ende der Zeichenkette

Wert 4 / 0x04

Verwandte

Siehe Befehle,

StrFind()

Option bei StrFind(), um eine Zeichenkette von hinten nach vorne zu durchsuchen.

Mit der Startposition wird bei StrFind() das Intervall für die Suche definiert. Wird der Parameter _StrFindReverse verwendet, beginnt die Suche am Ende der Zeichenkette und endet an der Startposition.

_StrFindToken

Begriffsorientierte Suche in einer Zeichenkette

Wert 2 / 0x02

Verwandte

Siehe Befehle,

StrFind()

Option bei StrFind(), um eine Zeichenkette begriffsorientiert zu durchsuchen. Bei der begriffsorientierten Suche wird nur nach ganzen Wörtern gesucht. Der Suchbegriff wird nur dann gefunden, wenn nach ihm ein Worttrennzeichen steht.

Worttrennzeichen sind alle Zeichen mit Ausnahme von Buchstaben oder Zahlen. Das Zeilenende trennt ebenfalls Wörter voneinander.

_StrFromANSI
Zeichenkette vom ANSI-Zeichensatz umwandeln
Wert 4 / 0x00000004

Verwandte
Befehle,
Siehe StrCnv(),
_StrFromOEM,
_StrFromHTML,
_StrFromUTF8

Option bei StrCnv() wodurch eine Zeichenkette aus dem ANSI-Zeichensatz (Windows Zeichensatz) in den von CONZEPT 16 verwendeten Zeichensatz umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den ANSI-Zeichensatz zu wandeln, wird die Option _StrToANSI verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrFromANSI);
```

_StrFromBase64
Zeichenkette aus Base64 dekodieren

Wert 524.288 /
0x00080000

Verwandte

Siehe Befehle,
StrCnv(),
_StrToBase64

Option bei StrCnv() wodurch eine Zeichenkette aus Base64 dekodiert wird. Die Datenmenge verkleinert sich dabei um 25%. Diese Option kann mit einer anderen _StrFrom...- oder _StrTo...Konstanten kombiniert werden.



Binärdaten, die 0-Byte-Zeichen enthalten sind nach der Konvertierung nur bis zum ersten 0-Byte-Zeichen verwertbar. Für diesen Fall muss MemCnv() verwendet werden.

Beispiel:

```
tSubject # StrCnv(tSubject, _StrFromBase64 | _StrFromUTF8);
```

_StrFromOEM
Zeichenkette vom OEM-Zeichensatz umwandeln
Wert 3 / 0x00000003

Verwandte
Befehle,
Siehe StrCnv(),
_StrFromANSI,
_StrFromHTML,
_StrFromUTF8

Option bei StrCnv() wodurch eine Zeichenkette aus dem OEM-Zeichensatz (PC Zeichensatz) in den von CONZEPT 16 verwendeten Zeichensatz umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den OEM-Zeichensatz zu wandeln, wird die Option _StrToOEM verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrFromOEM);
```

_StrFromOEM_852
Zeichenkette vom OEM-Zeichensatz (osteuropäisch) umwandeln

Wert 12 /
0x0000000C

Verwandte

Befehle,

StrCnv(),

Siehe _StrFromANSI,

_StrFromHTML,

_StrFromOEM,

_StrFromUTF8

Option bei StrCnv() wodurch eine Zeichenkette aus dem OEM-Zeichensatz (PC Zeichensatz) in den von CONZEPT 16 verwendeten Zeichensatz umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den OEM-Zeichensatz (osteuropäisch) zu wandeln, wird die Option _StrToOEM verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrFromOEM_852);
```

_StrFromHTML
Zeichenkette vom HTML-kodiertem Unicode umwandeln
Wert 8 / 0x00000008

Verwandte
Befehle,
Siehe StrCnv(),
_StrFromOEM,
_StrFromANSI,
_StrFromUTF8

Option bei StrCnv() wodurch eine Zeichenkette aus dem HTML-kodierten Unicode in den von CONZEPT 16 verwendeten Zeichensatz umgewandelt wird. Hierbei werden alle Sonderzeichen (&#...;) entsprechend umgesetzt.

Um die Zeichenkette wieder in den HTML-kodierten Unicode zu wandeln, wird die Option _StrToHTML verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrFromHTML);
```

_StrFromURI
Zeichenkette von URI-kodiertem UTF-8-Zeichensatz umwandeln
Wert 10 / 0x0000000A

Verwandte
Siehe Befehle,
StrCnv(),
_StrFromHTML

Option bei StrCnv() wodurch eine Zeichenkette von URI-kodiertem UTF-8-Zeichensatz in eine Zeichenkette im CONZEPT 16-Zeichensatz umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Werden bei der Internet-Kommunikation (Domain-Namen und URIs) Bezeichner mit Sonderzeichen verwendet, müssen diese in eine Zeichenkette gewandelt werden, in der nur zugelassene Zeichen enthalten sind. Dies wird durch eine entsprechende Kodierung erreicht, wie sie in den RFC 3490, 3491 und 3492 beschrieben wird.

Um die Zeichenkette wieder in den URI-kodierten UTF-8-Zeichensatz zu wandeln, wird die Option _StrToURI verwendet.

Beispiel:

```
tIdentifier # StrCnv(tURI, _StrFromURI);
```

_StrFromUTF8
Zeichenkette vom UTF-8-Zeichensatz umwandeln
Wert 6 / 0x00000006

Verwandte
Befehle,
Siehe StrCnv(),
_StrFromOEM,
_StrFromANSI,
_StrFromHTML

Option bei StrCnv() wodurch eine Zeichenkette aus dem UTF-8-Zeichensatz in den von CONZEPT 16 verwendeten Zeichensatz umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den UTF-8-Zeichensatz zu wandeln, wird die Option _StrToUTF8 verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrFromUTF8);
```

_StrLetter
Sonderzeichen umwandeln
Wert 1.024 /
0x00000400

Verwandte
Siehe Befehle,
StrCnv(),
_StrLetterExt

Option bei StrCnv() wodurch alle Umlaute gemäß der Option _StrUmlaut gewandelt und alle Zeichen, die keine Großbuchstaben (A, B, C, ... Z) oder Zahlen (0, 1, 2, ...9) sind, entfernt werden.

_StrLetterExt
Sonderzeichen umwandeln (erweitert)
Wert 64 /
0x00000040

Verwandte
Siehe Befehle,
StrCnv(),
_StrLetter

Option bei StrCnv() wodurch alle alphabetischen Sonderzeichen und Umlaute auf ihr Basiszeichen gewandelt werden. Andere Sonderzeichen (Leerzeichen, Bindestrich usw.) werden entfernt.

_StrLower
Zeichenkette in Kleinbuchstaben wandeln

Wert 32 /
0x00000020

Verwandte

Befehle,

Siehe StrCnv(),
_StrLower1252,
_StrUpper

Option bei StrCnv() wodurch die Zeichenkette in Kleinbuchstaben gewandelt wird.

StrLower1252
Zeichenkette in Kleinbuchstaben wandeln
Wert 128 /
0x00000080

Verwandte
Befehle,
Siehe StrCnv(),
StrLower,
StrUpper1252

Option bei StrCnv() wodurch die Zeichenkette in Kleinbuchstaben gewandelt wird. Es werden alle Großbuchstaben der Windows-Codepage 1252 in Kleinbuchstaben gewandelt.

_StrNameNoDouble
Namenskonvertierung ohne Doppelnamen

Wert 32.768 /
0x00008000

Verwandte

Siehe Befehle, StrCnv(),
_StrNameReorder,
_StrNameNoFirst

Option bei StrCnv() wodurch eine Namenskonvertierung ohne Doppelnamen durchgeführt wird.

Diese Option kann nur in Verbindung mit _StrNameReorder verwendet werden. Der zweite Namen in Doppelnamen entfällt bei der Konvertierung.

Beispiele:

```
StrCnv('Maier, Peter', _StrNameReorder | _StrNameNoDouble);           // => 'Peter Maier'
```

_StrNameNoFirst
Namenskonvertierung ohne Vornamen
Wert 16.384 / 0x00004000

Verwandte Befehle,

Siehe StrCnv(),
_StrNameReorder,
_StrNameNoDouble

Option bei StrCnv() wodurch eine Namenskonvertierung ohne Vornamen durchgeführt wird.

Diese Option kann nur in Verbindung mit _StrNameReorder verwendet werden. Der Vorname entfällt bei der Konvertierung.

Beispiele:

```
StrCnv('Maier, Peter', _StrNameReorder | _StrNameNoFirst);           // => 'Maier'StrCnv
```

_StrNameReorder
Namensumstellung
Wert 8.192 / 0x00002000

Verwandte Befehle,
Siehe StrCnv(),
_StrNameNoFirst,
_StrNameNoDouble

Option bei StrCnv() wodurch eine Namensumstellung durchgeführt wird.

Beispiele:

```
StrCnv('Maier, Peter', _StrNameReorder);
```

```
// => 'Peter Maier'
```



Es ist zu beachten, dass die Namensbestandteile durch ein Komma getrennt sein müssen.

StrSoundex1
Zeichenkette in SOUNDSEX I umwandeln

Wert 2.048 /
0x00000800

Verwandte

Siehe Befehle,
StrCnv(),
StrSoundex2

Option bei StrCnv() wodurch die Zeichenkette in SOUNDSEX I gewandelt wird.

Folgende Umsetzungen werden vorgenommen:

ai = ei
ay = ei
ch = h
chs = x
ck = k
ey = ei
ie = i
j = i
ph = f
th = t
y = i

Zusätzlich beinhaltet SOUNDSEX I eine Umwandlung in Großbuchstaben und die Wandlung von Umlauten, sowie die Entfernung aller sonstigen Sonderzeichen (siehe StrLetterExt). Gleiche Buchstaben, die mehrfach hintereinander auftreten, werden auf einen Buchstaben reduziert (aus "Zimmer" wird "ZIMER").

StrSoundex2
Zeichenkette in SOUNDINDEX II umwandeln
Wert 4.096 /
0x00001000

Verwandte
Siehe Befehle,
StrCnv(),
StrSoundex1

Option bei StrCnv() wodurch die Zeichenkette in SOUNDINDEX II gewandelt wird.

Folgende Umsetzungen werden zusätzlich (neben StrSoundex1) vorgenommen:

b = p
ch = k
d = t
g = k
sch = s
v = f
z = s

_StrToANSI
Zeichenkette in ANSI-Zeichensatz umwandeln

Wert 2 /
0x00000002

Verwandte
Befehle,
Siehe StrCnv(),
_StrToOEM,
_StrToHTML,
_StrToUTF8

Option bei StrCnv() wodurch eine Zeichenkette in den ANSI-Zeichensatz (Windows Zeichensatz) umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den CONZEPT 16-Zeichensatz zu wandeln, wird die Option _StrFromANSI verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrToANSI);
```

_StrToBase64
Zeichenkette nach Base64 kodieren

Wert 262.144 /
0x00040000

Verwandte

Siehe Befehle, StrCnv(),
_StrFromBase64

Option bei StrCnv() wodurch eine Zeichenkette nach Base64 kodiert wird. Die Datenmenge vergrößert sich dabei um ein Drittel. Diese Option kann mit einer anderen _StrTo...- oder _StrFrom...Konstanten kombiniert werden.

Bei der Konvertierung werden nur die effektiv vorhandenen Zeichen gewandelt. Wird beispielsweise eine leere Zeichenkette umgewandelt, ist die Zielzeichenkette ebenfalls leer.

Beispiel:

```
tSubject # StrCnv(tSubject, _StrToUTF8 | _StrToBase64);
```

_StrToOEM
Zeichenkette in OEM-Zeichensatz umwandeln

Wert 1 /
0x00000001

Verwandte
Befehle,
Siehe StrCnv(),
_StrToANSI,
_StrToHTML,
_StrToUTF8

Option bei StrCnv() wodurch eine Zeichenkette in den OEM-Zeichensatz (PC Zeichensatz) umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den CONZEPT 16-Zeichensatz zu wandeln, wird die Option _StrFromOEM verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrToOEM);
```

_StrToOEM_852
Zeichenkette in OEM-Zeichensatz (osteuropäisch) umwandeln

Wert 11 /
0x0000000B

Verwandte
Befehle,
StrCnv(),

Siehe _StrToANSI,
_StrToHTML,
_StrToOEM,
_StrToUTF8

Option bei StrCnv() wodurch eine Zeichenkette in den OEM-Zeichensatz (PC Zeichensatz) umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den CONZEPT 16-Zeichensatz zu wandeln, wird die Option _StrFromOEM_852 verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrToOEM_852);
```

_StrToHTML
Zeichenkette in Unicode mit HTML-Kodierung umwandeln

Wert 7 /
0x00000007

Verwandte
Befehle,
Siehe StrCnv(),
_StrToOEM,
_StrToANSI,
_StrToUTF8

Option bei StrCnv() wodurch eine Zeichenkette in HTML-kodierten Unicode umgewandelt wird. Insbesondere bedeutet das, dass alle Sonderzeichen in die Form &#x...; gewandelt werden.

Um die Zeichenkette wieder in den CONZEPT 16-Zeichensatz zu wandeln, wird die Option _StrFromHTML verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrToHTML);
```

_StrToURI
Zeichenkette in UTF-8 und URI-Kodierung umwandeln

Wert 9 /
0x00000009

Verwandte

Siehe Befehle,
StrCnv(),
_StrToHTML

Option bei StrCnv() wodurch eine Zeichenkette nach URI-kodiertem UTF-8-Zeichensatz umgewandelt wird.

Werden bei der Internet-Kommunikation (Domain-Namen und URIs) Bezeichner mit Sonderzeichen verwendet, müssen diese in eine Zeichenkette gewandelt werden, in der nur zugelassene Zeichen enthalten sind. Dies wird durch eine entsprechende Kodierung erreicht, wie sie in den RFC 3490, 3491 und 3492 beschrieben wird.

Um die Zeichenkette wieder in den CONZEPT 16-Zeichensatz zu wandeln, wird die Option _StrFromURI verwendet.

Beispiel:

```
tURI # StrCnv(tIdentifier, _StrToURI);
```

_StrToUTF8
Zeichenkette in UTF-8-Zeichensatz umwandeln

Wert 5 /
0x00000005

Verwandte
Befehle,
Siehe StrCnv(),
_StrToOEM,
_StrToANSI,
_StrToHTML

Option bei StrCnv() wodurch eine Zeichenkette in den UTF-8-Zeichensatz umgewandelt wird. Über eine weitere Anweisung kann die Zeichenkette weiter gewandelt werden.

Um die Zeichenkette wieder in den CONZEPT 16-Zeichensatz zu wandeln, wird die Option _StrFromUTF8 verwendet.

Beispiel:

```
tLine # StrCnv(tLine, _StrToUTF8);
```

_StrUmlaut
Umlaute umwandeln

Wert 512 /
0x00000200

Verwandte

Siehe Befehle,
StrCnv(),
_StrUpper

Option bei StrCnv() wodurch Umlaute in der Zeichenkette umgewandelt werden.
Zusätzlich beinhaltet die Option eine Umwandlung in Großbuchstaben (siehe
_StrUpper).

Folgende Umsetzungen werden vorgenommen:

ä, Ä = AE

ö, Ö = OE

ü, Ü = UE

ß = SS

_StrUpper
Zeichenkette in Großbuchstaben wandeln

Wert 16 /
0x00000010

Verwandte

Befehle,

Siehe StrCnv(),
_StrUpper1252,
_StrLower

Option bei StrCnv() wodurch eine Zeichenkette in Großbuchstaben gewandelt wird. Es werden die Buchstaben a-z, sowie die Umlaute ä, ö und ü in Großbuchstaben gewandelt.

StrUpper1252
Zeichenkette in Großbuchstaben wandeln
Wert 65.536 /
0x00010000

Verwandte
Befehle,
Siehe StrCnv(),
StrUpper,
StrLower1252

Option bei StrCnv() wodurch die Zeichenkette in Großbuchstaben gewandelt wird. Es werden alle Kleinbuchstaben der Windows-Codepage 1252 in Großbuchstaben gewandelt.

Datumsfunktionen

Datumsfunktionen

Befehlsgruppen,

Siehe Befehlsliste,

Zeitfunktionen

Befehle

- DateDay
- DateDayOfWeek
- DateMake
- DateMonth
- DateYear
- SysDate

Datentypen

- date
- caltime



DateDay(date1) : int

Tag aus einem Datumswert ermitteln

date1 Datumswert

Resultat int Tag des Datums

Verwandte Befehle,

DateDayOfWeek(),

Siehe DateMonth(),

DateYear(),

DateMake()

Diese Funktion liefert aus dem Datum (date1) als Resultat den Tag des Datums. Diese Funktion kann auch benutzt werden, um ein leeres Datumsfeld zu erkennen, da ein leeres Datum den Wert 0 zurückgibt.

DateDayOfWeek(date1) : int



Wochentag aus einem Datumswert ermitteln

date1 Datumswert

Resultat int Wochentag des Datums

Verwandte Befehle,

Siehe DateDay(), DateMonth(),

DateYear(), DateMake()

Diese Funktion liefert aus dem Datum (date1) als Resultat den Wochentag des Datums. Dabei sind die Wochentage mit folgenden Werten definiert:

- 1 = Montag
- 2 = Dienstag
- 3 = Mittwoch
- 4 = Donnerstag
- 5 = Freitag
- 6 = Samstag
- 7 = Sonntag

DateMonth(date1) : int



Monat aus einem Datumswert ermitteln

date1 Datumswert

Resultat int Monat des Datums

Verwandte Befehle,

Siehe DateDay(),

DateYear(),

DateMake()

Diese Funktion liefert aus dem Datum (date1) als Resultat den Monat des Datums.



DateYear(date1) : int

Jahr aus einem Datumswert ermitteln

date1 Datumswert

Resultat int Jahr des Datums

Verwandte Befehle,

Siehe DateDay(),

DateMonth(),

DateMake()

Diese Funktion liefert aus dem Datum (date1) als Resultat das Jahr des Datums, es entspricht der Anzahl von Jahren seit 1900.

Beispiele:

```
DateYear(23.11.1984)    // 84DateYear(04.03.2007)    // 107
```

DateMake(int1, int2, int3) : date



Datum erzeugen

int1 Tageswert

int2 Monatswert

int3 Jahreswert

Resultat date Zusammengesetztes Datum

Verwandte Befehle,

Siehe DateDay(), DateMonth(),
DateYear()

Mit diesem Befehl kann aus den drei Werten ein Datum zusammengesetzt werden (int1 = Tag, int2 = Monat, int3 = Jahr). Die Werte müssen ein gültiges Datum ergeben.

Der Jahreswert (int3) kann entweder im Bereich von 0 bis 254 oder im Bereich von 1900 bis 2154 liegen.

Soll der Monatsletzte rechnerisch ermittelt werden, muss zunächst auf den 1. des folgenden Monats positioniert und dann ein Tag davon subtrahiert werden. Um einen Datumswert nach dem 31.12.1999 zu erzeugen, wird das zweistellige Jahr um 100 erhöht.

Bei der Angabe ungültiger Werte wird ein leeres Datum zurückgeliefert und der globale Fehlerwert auf ErrValueInvalid gesetzt.

Beispiele:

DateMake(13, 11, 74) // 13.11.1974 DateMake(7, 2, 102) // 07.02.2002 DateMake(30, 11, 1995)

Zeitfunktionen

Zeitfunktionen

Befehlsgruppen,

Siehe Befehlsliste,

Datumsfunktionen

Befehle

- TimeHour
- TimeHSec
- TimeMake
- TimeMin
- TimeSec

- SysTime
- SysTics

Datentypen

- time
- calttime



TimeHour(time1) : int

Stunden aus einem Zeitwert ermitteln

time1 Zeitwert

Resultat int Stunden des Zeitwerts

Verwandte Befehle,

Siehe TimeHSec(), TimeSec(),

TimeMin(), TimeMake()

Diese Funktion liefert aus dem Zeitwert (time1) als Resultat die Stunden des Zeitwerts.



TimeMin(time1) : int

Minuten aus einem Zeitwert ermitteln

time1 Zeitwert

Resultat int Minuten des Zeitwerts

Verwandte Befehle,

Siehe TimeHSec(), TimeSec(),

TimeHour(), TimeMake()

Diese Funktion liefert aus dem Zeitwert (time1) als Resultat die Minuten des Zeitwerts.



TimeSec(time1) : int

Sekunden aus einem Zeitwert ermitteln

time1 Zeitwert

Resultat int Sekunden des Zeitwerts

Verwandte Befehle,

Siehe TimeHSec(), TimeMin(),
 TimeHour(), TimeMake()

Diese Funktion liefert aus dem Zeitwert (time1) als Resultat die Sekunden des Zeitwerts.

TimeHSec(time1) : int




Hundertstelsekunden aus einem Zeitwert ermitteln

time1 Zeitwert

Resultat int Hundertstelsekunden des Zeitwerts

Siehe Verwandte Befehle, TimeSec(),
 TimeMin(), TimeHour(), TimeMake()

Diese Funktion liefert aus dem Zeitwert (time1) als Resultat die Hundertstelsekunden des Zeitwerts.

TimeMake(int1,
int2, int3, int4) : 
time

Zeitwert erzeugen

int1 Stunden

int2 Minuten

int3 Sekunden

int4 Hundertstelsekunden

Resultat time Zusammengesetzter
Zeitwert

Verwandte Befehle,

Siehe TimeHSec(), TimeSec(),

TimeMin(), TimeHour()

Mit diesem Befehl kann aus den vier Werten ein Zeitwert zusammengesetzt werden (int1 = Stunden, int2 = Minuten, int3 = Sekunden, int4 = Hundertstelsekunden). Die Werte müssen einen gültigen Zeitwert ergeben.

Bei der Angabe ungültiger Werte wird der Zeitwert 00:00 zurückgeliefert und der globale Fehlerwert auf ErrValueInvalid gesetzt.

Beispiele:

```
TimeMake(13, 11, 0, 0)    // 13:11:00.00TimeMake(22, 50, 59, 71) // 22:50:59.71
```

Farbfunktionen

Funktionen zur Umwandlung von Farben

Siehe [Befehlsgruppen](#),

[Befehlsliste](#)

Befehle

- [ColorMake](#)
- [ColorRgbMake](#)
- [WinColorOpacityGet](#)
- [WinColorOpacitySet](#)

ColorMake(int1[ byte2]) : color

Farbe erzeugen

int1 Farbwert

byte2 Transparenz
(optional)

Resultat color Farbe

Verwandte

Siehe Befehle, color,
ColorRgbMake()

Mit diesem Befehl kann aus einem Farbwert (int1) und einer Transparenz (byte2) eine Farbe zusammengesetzt werden. In (int1) kann ein Farbwert oder eine der _WinCol...-Konstanten angegeben werden. Der Befehl setzt die Eigenschaften der Farbe. Wird keine Transparenz angegeben, ist die Farbe deckend (Transparenz = 0).



Der Befehl ignoriert eine evtl. durch WinColorOpacitySet() gesetzte Deckkraft im Farbwert des Argumentes (int1).

Beispiele:

```
tColor # ColorMake(_WinColWhite, 128)      // Weiß mit 50 Transparenz
```

Mögliche Laufzeitfehler:

ErrValueInvalid In (int1) ist ein ungültiger Farbwert übergeben worden.

ColorRgbMake(byte1, byte2, byte3) : int



Farbwert erzeugen

byte1 Farbanteil rot

byte2 farbanteil
 grün

byte3 Farbanteil
 blau

Resultat int Farbwert

Verwandte

Siehe Befehle,
 color,
 ColorMake()

Mit diesem Befehl kann aus den drei Farbanteilen (rot, grün, blau) ein Farbwert erstellt werden (byte1 = rot, byte2 = grün, byte3 = blau). Der daraus resultierende Farbwert kann in eine der Farb-Eigenschaften geschrieben werden.

Beispiele:

```
ColorRgbMake(255, 255, 255) // weiß
ColorRgbMake(255, 0, 0)     // hellrot
```

WinColorOpacityGet(int1) : int
 Deckkraft einer Farbe ermitteln
 int1 Farbwert
 Resultat int Deckkraft in Prozent



Siehe [Verwandte Befehle](#),
[WinColorOpacitySet\(\)](#)

Dieser Befehl ermittelt die Deckkraft eines Farbwertes (int1). Die Deckkraft (int2) wird mit Werten zwischen 0 (transparent) und 100 (deckend) angegeben.


Als Farbwert (int1) kann ein beliebiger RGB-Wert (Siehe [ColorRgbMake\(\)](#)) oder eine der _WinCol-Farbkonstanten (z. B. _WinColLightRed) angegeben werden. Bei nicht erlaubten Farbwerten (_WinColUndefined, _WinColParent, _WinColTransparent) und Systemfarben (z. B. _WinColScrollBar) wird der Laufzeitfehler ErrValueInvalid ausgelöst.

Beispiel:

```
// Deckkraft von rot ermittelnOpacity # WinColorOpacityGet(_WinColLightRed);// Deckkraft einer F
```

Mögliche Laufzeitfehler:

ErrValueInvalid Ungültiger Farbwert (z. B. _WinColUndefined, _WinColParent, _WinColTransparent, _WinColScrollBar) angegeben.

WinColorOpacitySet(int1, ) : int

Deckkraft einer Farbe setzen

int1 Farbwert

int2 Deckkraft in Prozent

Resultat int Um Deckkraft erweiterter Farbwert

Siehe Verwandte Befehle,
WinColorOpacityGet()

Dieser Befehl setzt die Deckkraft (int2) eines Farbwertes (int1) und gibt den neuen Farbwert zurück. Die Deckkraft (int2) kann mit Werten zwischen 0 (transparent) und 100 (deckend) angegeben werden.

Als Farbwert (int1) kann ein beliebiger RGB-Wert (Siehe ColorRgbMake()) oder eine der _WinCol-Farbkonstanten (z. B. _WinColLightRed) angegeben werden. Handelt es sich bei der Farbkonstante um eine Systemfarbe (_WinColScrollBar etc.), dann wird die Systemfarbe durch den entsprechenden RGB-Wert ersetzt. Bei nicht erlaubten Farbwerten (_WinColUndefined, _WinColParent, _WinColTransparent) wird der Laufzeitfehler _ErrValueInvalid ausgelöst.

Ist das Argument Opacity < 0 oder > 100, wird es entsprechend in den Bereich 0 ... 100 verschoben.

Ist in einem Farbwert bereits ein Deckkraft gesetzt, wird diese durch den neuen Wert überschrieben.

Beispiel:

```
// Rote Farbe mit 50 % Deckkraft setzentColor # WinColorOpacitySet(_WinColLightRed, 50);// RGB-Fa
```

Mögliche Laufzeitfehler:

_ErrValueInvalid Ungültiger Farbwert (z. B. _WinColUndefined, _WinColParent, _WinColTransparent) angegeben.

Operatoren

Operatoren

Siehe Befehlsgruppen,
Befehlsliste

- Binäroperatoren
- Logische Operatoren
- Mathematische Operatoren
- Zuweisungsoperator

Logische Operatoren

Logische Operatoren

Siehe Befehlsgruppen,
Befehlsliste

- and
- Negation
- or
- Vergleichsoperatoren
- xor

Vergleichsoperatoren
Vergleichen von Ausdrücken

Logische

Siehe Operatoren,
Operatorpriorität

Syntax:

Ergebnis = <Ausdruck1> <Vergleichsoperator> <Ausdruck2>

Op.	Vergleich	Ergebnis = <u>true</u>	Ergebnis = <u>false</u>
=	gleich	(5 = 3 + 2) (tVar = tVar)	(1 + 1 = 1) ('A' = 'a')
= ^	gleich (ohne Groß-/Kleinschreibung)	(5 = ^ 3 + 2) ('A' = ^ 'a')	(1 + 1 = ^ 1) ('abc' = ^ 'ABD')
<> oder !=	ungleich	(1 + 1 != 1) ('A' <> 'a')	(5 <> 3 + 2) (tVar != tVar)
>	größer	(1 + 1 > 1) ('a' > 'A')	(5 > 3 + 2) (tVar > tVar)
<	kleiner	(1 < 1 + 1) ('A' < 'a')	(5 < 3 + 2) (tVar < tVar)
>=	größer oder gleich	(1 + 1 >= 1) (5 >= 3 + 2)	('A' >= 'a') (tVar >= tVar + 1)
<=	kleiner oder gleich	(1 <= 1 + 1) (5 <= 3 + 2)	('a' <= 'A') (tVar <= tVar - 1)
=*	<u>ähnlich</u>	('T000' =* 'T*') ('T01' =* 'T??') ('T0T0T' =* 'T**T')	('1. T' =* 'T*') ('T' =* 'T??') ('t000' =* 'T*')
=*^	<u>ähnlich</u> (ohne Groß-/Kleinschreibung)	('T000' =*^ 'T*') ('T01' =*^ 'T??') ('T0T0T' =*^ 'T**T') ('t000' =*^ 'T*')	('1. T' =*^ 'T*') ('T' =*^ 'T??')

and

Logische Konjunktion zwischen zwei Ausdrücken

Logische Operatoren,

Operatorpriorität, or,

Siehe xor,

Vergleichsoperatoren,

Binär-UND (&)

Syntax:

Ergebnis = <Ausdruck1> and <Ausdruck2>

Die Bedingung (a and b) wird dann wahr, wenn beide Ausdrücke wahr sind.

a \ b false true

false false false

true false true

Liefert der erste Ausdruck bereits das Ergebnis false, wird der zweite Ausdruck nicht ausgewertet. Dadurch kann in einer Abfrage die Existenz eines Objektes und der Wert einer Eigenschaft überprüft werden.

```
if (tHdlObj > 0 and tHdlObj->wpVisible) ...
```

Mit dem Binär-UND (&)-Operator lassen sich auch ganze Zahlen binär verarbeiten.

or

Logische Disjunktion zwischen zwei Ausdrücken

Logische Operatoren,

Operatorpriorität,

Siehe and, xor,

Vergleichsoperatoren,

Binär-ODER (|)

Syntax:

Ergebnis = <Ausdruck1> or <Ausdruck2>

Die Bedingung (a or b) wird dann wahr, wenn mindestens ein Ausdruck wahr ist.

a \ b false true

false false true

true true true

Liefert der erste Ausdruck bereits das Ergebnis true, wird der zweite Ausdruck nicht ausgewertet. Dadurch kann in einer Abfrage die Existenz eines Objekts und der Wert einer Eigenschaft überprüft werden.

```
if (tHdlObj <= 0 or !tHdlObj->wpVisible) ...
```

Mit dem Binär-ODER (|)-Operator lassen sich auch ganze Zahlen binär verarbeiten.

xor

Logische Exklusion zwischen zwei Ausdrücken

Logische Operatoren,

Operatorpriorität,

Siehe and, or,

Vergleichsoperatoren,

Binär-Exklusiv-ODER

(^)

Syntax:

Ergebnis = <Ausdruck1> xor <Ausdruck2>

Die Bedingung (a xor b) wird dann wahr, wenn genau ein Ausdruck wahr ist.

a \ b false true

false false true

true true false

Mit dem Binär-Exklusiv-ODER (^)-Operator lassen sich auch ganze Zahlen binär verarbeiten.

Negation (!)

Logische Negation eines Ausdrucks

Logische Operatoren,

Operatorpriorität,

Siehe and, or, xor,

Vergleichsoperatoren,

Binär-NICHT (~)

Syntax:

Ergebnis = !<Ausdruck>;

Die Bedingung !a wird dann wahr, wenn a nicht wahr ist.

a false true

!a true false

Mit dem Binär-NICHT (~)-Operator lassen sich auch ganze Zahlen binär verarbeiten.

Mathematische Operatoren

Mathematische Operatoren

Befehlsgruppen,

Siehe Befehlsliste,

Mathematische

Funktionen

- Addition
- div
- Division
- Modulo-Operator
- Multiplikation
- Subtraktion

Addition (+)

Addition von Zahlen oder Zeichenketten

Mathematische

Siehe Operatoren,

Operatorpriorität

Syntax:

Ergebnis # <Ausdruck1> + <Ausdruck2>

Die Addition ist für die Typen alpha, byte, word, int, bigint, float und decimal definiert.
Bei der Addition vom Typ alpha werden die Zeichenketten aneinander gehängt.

Subtraktion (-)

Subtraktion von Zahlen

Mathematische

Siehe Operatoren,

Operatorpriorität,

Syntax:

Ergebnis # <Ausdruck1> - <Ausdruck2>

Die Subtraktion ist für die Typen byte, word, int, bigint, float und decimal definiert.

Multiplikation (*)

Multiplikation von Zahlen

Mathematische

Siehe Operatoren,

Operatorpriorität

Syntax:

Ergebnis # <Ausdruck1> * <Ausdruck2>

Beide Operanden und das Ergebnis müssen den gleichen numerischen Typ besitzen.

Division (/)

Division von Zahlen

Mathematische

Siehe Operatoren,

Operatorpriorität,

div

Syntax:

Ergebnis # <Ausdruck1> / <Ausdruck2>

Mögliche Laufzeitfehler:

ErrDivisionByZero <Ausdruck2> ist 0.

Modulo-Operator (%)

Rest nach einer ganzzahligen Division von Zahlen

Mathematische

Siehe Operatoren,
Operatorpriorität,
div

Syntax:

Ergebnis # <Operand1> % <Operand2>

Bei der Verarbeitung wird der ganzzahlige Teil von <Operand1> durch <Operand2> (vom gleichen Typ) geteilt, wobei eventuell vorhandene Nachkommastellen bei <Operand2> abgeschnitten werden. Der bei der Division entstandene Rest ist das Resultat der Operation.

Die Berechnung findet nach dem symmetrischen Verfahren statt. Dieses unterscheidet sich bei negativen Werten von dem Ergebnis des mathematischen Verfahrens. Um das Ergebnis in der mathematischen Variante zu erhalten, kann wie folgt vorgegangen werden:

tA # -1.4285714085714\;f;tB # 100.0\;f;tRes # (tA % tB + tB) % tB;

Nähere Informationen dazu finden Sie im Wikipedia-Artikel [Division mit Rest](#).

Mögliche Laufzeitfehler:

[ErrDivisionByZero](#) <Operand2> ist 0.

div

Ganzzahligen Division von Zahlen

Mathematische

Siehe Operatoren,
Operatorpriorität,
Modulo-Operator

Syntax:

Ergebnis # <Ausdruck1> div <Ausdruck2>

Bei der ganzzahligen Division wird der Dividend <Ausdruck1> durch einen ganzzahligen Divisor <Ausdruck2> geteilt, wobei das Resultat der ebenfalls ganzzahlige Quotient ist.

Mögliche Laufzeitfehler:

ErrDivisionByZero <Ausdruck2> ist 0.

Binäroperatoren

Binäroperatoren

Siehe Befehlsgruppen,
Befehlsliste

- Binär-Exklusiv-ODER
- Binär-NICHT
- Binär-ODER
- Binär-UND
- Bit-Verschiebung Links
- Bit-Verschiebung Rechts

Binär-ODER (|)
 Binäre ODER-Verknüpfung
Binäroperatoren,
 Siehe Operatorpriorität,
or, Binär-UND
(&)

Syntax:

Ergebnis # <Ausdruck1> | <Ausdruck2>

Verknüpft jedes Bitpaar der beiden Operanden über ein inklusives Oder. Die Oder-Operation ergibt den Wert WAHR, wenn mindestens einer der Operanden WAHR ist.

Beispiele:

Ausdruck Binärdarstellung Ergebnis Binärdarstellung

7 4	0111 0100	7	0111
10 4	1010 0100	14	1110
9 14	1001 1110	15	1111

Diese Operation kann verwendet werden, um mehrere Konstanten zu kombinieren.
 Zum Beispiel beim Befehl RecRead():

```
...// Nächsten Datensatz lesen und sperrentErg # RecRead(KND.D.Kunden, KND.S.iNummer, _RecNext |
```

Binär-UND (&)

Binäre UND-Verknüpfung

Binärooperatoren,

Siehe Operatorpriorität,

and, Binär-ODER

()

Syntax:

Ergebnis # <Ausdruck1> & <Ausdruck2>

Verknüpft jedes Bitpaar der beiden Operanden über ein Und. Die Und-Operation ergibt den Wert WAHR, wenn beide Operanden WAHR sind.

Beispiele:

Ausdruck Binärdarstellung Ergebnis Binärdarstellung

7 & 4 0111 & 0100 4 0100

10 & 4 1010 & 0100 0 0000

9 & 14 1001 & 1110 8 1000

Diese Operation kann verwendet werden, um zu überprüfen, ob ein bestimmtes Attribut in einem Wert gesetzt ist. Zum Beispiel in der Ereignisfunktion

EvtMouseItem:

```
...if (aButton & _WinMouseShift > 0) // Maustaste wurde zusammen mit Umschalt-Taste gedrückt...
```

Binär-Exklusiv-ODER (^)
 Binäre Exklusiv-Oder-Verknüpfung
Binäroperatoren,
 Siehe Operatorpriorität,
xor

Syntax:

Ergebnis # <Ausdruck1> ^ <Ausdruck2>

Verknüpft jedes Bitpaar der beiden Operanden über ein exklusives Oder. Die exklusive Oder-Operation ergibt den Wert WAHR, wenn genau einer der beiden Operanden WAHR ist.

Beispiele:**Ausdruck Binärdarstellung Ergebnis Binärdarstellung**

7 ^ 4	0111 ^ 0100	3	0011
10 ^ 4	1010 ^ 0100	14	1110
9 ^ 14	1001 ^ 1110	7	0111

Binär-NICHT (~)
 Binärer Nicht-Operator
Binäroperatoren,
 Siehe Operatorpriorität,
Negation (!)

Syntax:

Ergebnis # ~ <Ausdruck>

Dieser Operand ist das Gegenstück zur logischen Negation. Alle nicht gesetzten Bits des Operanden werden gesetzt und alle gesetzten Bits werden zurückgesetzt.

Beispiele:

Ausdruck Binärdarstellung Ergebnis Binärdarstellung

~ 31	~ 00011111	224	11100000
~ 170	~ 10101010	85	01010101

Der Operator kann verwendet werden, um bestimmte Bits zurückzusetzen:

```
tFont:Attributes # tFont:Attributes & ~_WinFontAttrUnderline;
```

Bit-Verschiebung Rechts (>>)

Bit-Verschiebung Rechts (>>)

Binärooperatoren,

Siehe Operatorpriorität,

Bit-Verschiebung

Links

Syntax:

Ergebnis # <Ausdruck1> >> <Ausdruck2>

Schiebt das in <Ausdruck1> enthaltene Bitmuster um die Anzahl der Stellen nach rechts, die in <Ausdruck2> angegeben ist. Die höherwertigen Bits werden auf die Stelle der niederwertigen Bits verschoben. Das höchstwertige Bit wird kopiert und von der linken Seite nachgeschoben. Werte vom Typ byte und word werden intern als int behandelt, sodass hier von der linken Seite nicht gesetzte Bits nachgeschoben werden.

Beispiele:

Ausdruck	Binärdarstellung	Ergebnis	Binärdarstellung
4 >> 1	0100	2	0010
4 >> 3	0100	0	0000
12 >> 2	1100	3	0011
0x40000000 >> 1	0100...0000	0x20000000	0010...0000
0x80000000 >> 1	1000...0000	0xc0000000	1100...0000

Bit-Verschiebung Links (<<)

Bit-Verschiebung Links (<<)

Binäroperatoren,

Siehe Operatorpriorität,

Bit-Verschiebung

Rechts

Syntax:

Ergebnis # <Ausdruck1> << <Ausdruck2>

Schiebt das in <Ausdruck1> enthaltene Bitmuster um die Anzahl der Stellen nach links, die in <Ausdruck2> angegeben ist. Die niederwertigen Bits werden auf die Stelle der höherwertigen Bits verschoben. Von der rechten Seite werden nicht gesetzte Bits nachgeschoben.

Beispiele:

Ausdruck Binärdarstellung Ergebnis Binärdarstellung

2 << 1	0010	4	0100
3 << 2	0011	12	1100
5 << 3	0101	8	1000

Zuweisungsoperator (#)

Wertzuweisung

Siehe Befehlsgruppen,
Befehlsliste

Syntax:

<Variable/Feld/Eigenschaft> # <Ausdruck>

Beispiele:

"1,3,4" # 4712; // Feld (Nummer)fiCstNumber # 555; //

Operatorpriorität

Priorität der Operatoren

Siehe Operatoren

Die Operatoren werden in einer bestimmten Priorität behandelt. Operatoren mit einer höheren Priorität werden vor Operatoren mit einer niedrigeren Priorität ausgewertet. Bei Operatoren der gleichen Priorität wird der Ausdruck von links nach rechts ausgewertet.

Priorität Operator

7	<u>!</u> , <u>~</u>
6	<u>*</u> , <u>%</u> , <u>/</u> , <u>div</u>
5	<u>±</u> , <u>=</u>
4	<u><<</u> , <u>>></u>
3	<u>↓</u> , <u>&</u> , <u>^</u>
2	<u>Vergleichsoperatoren</u>
1	<u>and</u> , <u>or</u> , <u>xor</u>

Funktionen auf Datenbankelemente

Funktionen auf Datenbankelemente

Siehe Befehlsgruppen,
Befehlsliste

- Befehle für dynamische Selektionen
- Befehle für Storage-Objekte
- Benutzerbefehle
- Datenbankbefehle
- Datensatz- und Datenstrukturbefehle
- OEM-Kit-Befehle
- Selektionen
- Textbefehle

Datensatz- und Datenstrukturbefehle

Befehle zum Umgang mit Datensätzen und der Datenstruktur

Siehe [Befehlsgruppen](#),
[Befehlsliste](#)

Befehle

- [DtaBegin](#)
- [DtaCommit](#)
- [DtaRollback](#)
- [FileInfo](#)
- [FileInfoByName](#)
- [FileName](#)
- [Fld...](#)
- [Fld...ByName](#)
- [FldAttributes](#)
- [FldCompare](#)
- [FldCopy](#)
- [FldDef](#)
- [FldDefByName](#)
- [FldInfo](#)
- [FldInfoByName](#)
- [FldName](#)
- [KeyFldInfo](#)
- [KeyInfo](#)
- [KeyInfoByName](#)
- [KeyName](#)
- [LinkFldInfo](#)
- [LinkInfo](#)
- [LinkInfoByName](#)
- [LinkName](#)
- [RecBufClear](#)
- [RecBufCompare](#)
- [RecBufCompareFld](#)
- [RecBufCopy](#)
- [RecBufCreate](#)
- [RecBufDefault](#)
- [RecBufDestroy](#)
- [RecDelete](#)
- [RecDeleteAll](#)
- [RecFilterAdd](#)
- [RecFilterCreate](#)
- [RecFilterDestroy](#)
- [RecInfo](#)
- [RecInsert](#)
- [RecLink](#)
- [RecLinkInfo](#)
- [RecRead](#)
- [RecReplace](#)
- [ShrClear](#)
- [ShrCompare](#)
- [ShrCopy](#)

- SbrInfo
- SbrInfoByName
- SbrName
- SbrStatus

Konstanten

- FileEncrypted
- FileExists
- FileFldBuf
- FileId64
- FileKeyCount
- FileLinkCount
- FileMaster
- FileNumber
- FileOEMMark
- FileRecModified
- FileSbrCount
- FileSeqInsert
- FileTemp
- FileUserLevel
- FldExists
- FldFileNumber
- FldInputRight
- FldLen
- FldNumber
- FldOutputRight
- FldSbrNumber
- FldType
- FltAbove
- FltAboveEq
- FltAND
- FltBelow
- FltBelowEq
- FltEq
- FltLike
- FltOR
- FltScan
- FltUneq
- FltXOR
- KeyExists
- KeyFileNumber
- KeyFldAttributes
- KeyFldAttrReverse
- KeyFldAttrSoundex1
- KeyFldAttrSoundex2
- KeyFldAttrSpecialChars
- KeyFldAttrUmlaut
- KeyFldAttrUpperCase
- KeyFldCount
- KeyFldExists

- KeyFldFileNumber
- KeyFldMaxLen
- KeyFldNumber
- KeyFldSbrNumber
- KeyFldType
- KeyIsUnique
- KeyMaxLen
- KeyNumber
- LinkDestFileNumber
- LinkDestKeyNumber
- LinkExists
- LinkFileNumber
- LinkFldAttributes
- LinkFldAttrPosition
- LinkFldCount
- LinkFldExists
- LinkFldFileNumber
- LinkFldMaxLen
- LinkFldNumber
- LinkNumber
- RecCheckLock
- RecCount
- RecCountNext
- RecCountPos
- RecEarlyCommit
- RecFirst
- RecForceLock
- RecGetPos
- RecGetPosReverse
- RecGetPrime
- RecID
- RecKeyReverse
- RecLast
- RecLock
- RecLockedBy
- RecModified
- RecNext
- RecNoLoad
- RecPos
- RecPrev
- RecSetID
- RecSharedLock
- RecSingleLock
- RecTest
- RecUnlock
- RecUpdateCounter
- RecUseId
- SbrExists
- SbrFileNumber
- SbrFldCount
- SbrNumber

- SbrStatus

Dateibefehle

Befehle zum Ermitteln von Dateiinformationen

Siehe Befehlsgruppen,
Befehlsliste

Befehle

FileInfo() Dateiinformationen über Dateinummer ermitteln

FileInfoByName() Dateiinformationen über Dateinamen ermitteln

FileName() Dateinamen ermitteln

Konstanten

_FileExists Dateiexistenz

_FileNumber Dateinummer

_FileMaster Hauptdatei

_FileTemp temporäre Datei

_FileFldBuf eigene Feldpuffer

_FileSeqInsert sequentielles Einfügen

_FileEncrypted chiffrierte Speicherung

_FileOEMMark OEM-Kit-Markierung

_FileId64 64-Bit Datensatz-IDs

_FileSbrCount Teildatensatzanzahl

_FileKeyCount Schlüsselanzahl

_FileLinkCount Verknüpfungsanzahl

_FileUserLevel Benutzerberechtigung

FileName(int1) :



alpha

Dateiname abfragen

int1 Dateinummer

Resultat alpha Dateiname

Verwandte

Siehe Befehle,

FileInfoByName()

Diese Funktion ermittelt den Namen der angegebenen Datei.

Beispiel:

```
if (FileName(100) = 'KND.D.Kunden'){ ...}
```

Mögliche Laufzeitfehler:

ErrNoFile Datei ist nicht vorhanden



FileInfo(int1, int2) : int

Dateiinformatioren über die Dateinummer abfragen

int1 Dateinummer

Informationstyp

FileExists Dateiexistenz

FileMaster Hauptdatei

FileTemp temporäre Datei

FileFldBuf eigene Feldpuffer

FileSeqInsert sequentielles
Einfügen

FileEncrypted chiffrierte
Speicherung

int2

FileOEMMark OEM-Kit-Markierung

FileId64 64-Bit Datensatz-IDs

FileSbrCount Teildatensatzanzahl

FileKeyCount Schlüsselanzahl

FileLinkCount Verknüpfungsanzahl

FileUserLevel Benutzerberechtigung

FileRecModified Datensätze mit
Änderungszeitpunkt

Resultat int Dateiinformatioren

Siehe Verwandte Befehle, FileInfoByName(),
Beispiel - Verknüpfungen

Mit dieser Funktion können verschiedene Informationen zu einer Datei über die Dateinummer abgefragt werden.

Mögliche Laufzeitfehler:

ErrNoFile Datei ist nicht vorhanden (nicht bei Verwendung von FileExists)



FileInfoByName(alpha1, int2) : int

Dateiinformatioren über den Dateinamen abfragen

alpha1 Dateiname

Informationstyp

FileExists Dateiexistenz

FileNumber Dateinummer

FileMaster Hauptdatei

FileTemp temporäre Datei

FileFldBuf eigene Feldpuffer

FileSeqInsert sequentielles
Einfügen

int2 FileEncrypted chiffrierte
Speicherung

FileOEMMark OEM-Kit-Markierung

FileId64 64-Bit Datensatz-IDs

FileSbrCount Teildatensatzanzahl

FileKeyCount Schlüsselanzahl

FileLinkCount Verknüpfungsanzahl

FileUserLevel Benutzerberechtigung

FileRecModified Datensätze mit
Änderungszeitpunkt

Resultat int Dateiinformatioren

Siehe Verwandte Befehle, FileInfo()

Mit dieser Funktion können verschiedene Informationen zu einer Datei über den Dateinamen abgefragt werden.

Mögliche Laufzeitfehler:

ErrNoFile Datei ist nicht vorhanden (außer bei FileExists)

Konstanten für Dateibefehle

Konstanten für Dateibefehle

Siehe Dateibefehle

FileEncrypted chiffrierte Speicherung

FileExists Dateiexistenz

FileFldBuf eigene Feldpuffer

FileId64 64-Bit Datensatz-IDs

FileKeyCount Schlüsselanzahl

FileLinkCount Verknüpfungsanzahl

FileMaster Hauptdatei

FileNumber Dateinummer

FileOEMMark OEM-Kit-Markierung

FileSbrCount Teildatensatzanzahl

FileSeqInsert sequentielles Einfügen

FileTemp Temporäre Datei

FileUserLevel Benutzerberechtigung

_FileEncrypted
Chiffrierte Speicherung
Wert 10

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die festgestellt werden kann, ob die Daten chiffriert gespeichert werden.

Werden die Daten chiffriert, wird 1 zurückgegeben, sonst 0.

_FileExists
Dateiexistenz
Wert 0

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die die Existenz einer Datei ermittelt werden kann.

Falls die Datei existiert: Resultat = 1, andernfalls Resultat = 0.

_FileFldBuf

Eigene Feldpuffer bei untergeordneter Datei

Wert 13

Verwandte

Siehe Befehle, FileInfo(),

FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die festgestellt werden kann, ob die die untergeordnete Datei eigene Feldpuffer hat.

Sind die eigenen Feldpuffer aktiviert, wird 1 zurückgegeben, sonst 0.

_FileId64

64-Bit Datensatz-IDs

Wert 11

Verwandte

Siehe Befehle, FileInfo(),

FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die festgestellt werden kann, ob die Datensatz-ID mit bis zu 64-Bit gespeichert wird.

Sind 64-Bit Datensatz-IDs aktiviert, wird 1 zurückgegeben, sonst 0.

_FileKeyCount
Schlüsselanzahl
Wert 5

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die die Anzahl der Schlüssel einer Datei ermittelt werden kann.

_FileLinkCount
Verknüpfungsanzahl
Wert 6

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die die Anzahl der Verknüpfungen einer Datei ermittelt werden kann.

_FileMaster

Hauptdatei

Wert 3

Verwandte

Siehe Befehle, FileInfo(),

FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die die Hauptdatei einer Datei ermittelt werden kann.

_FileNumber
Dateinummer
Wert 1

Verwandte

Siehe Befehle,

FileInfoByName()

Option bei FileInfoByName() durch die die Dateinummer einer Datei ermittelt werden kann.

_FileRecModified

Dateiexistenz

Wert 14

Verwandte

Siehe Befehle, FileInfo(),

FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die ermittelt werden kann, ob bei Datensätzen der Datei der Änderungszeitpunkt gespeichert wird.

Der Änderungszeitpunkt eines Datensatzes kann über die Option _RecModified des Befehls RecInfo() ermittelt werden.

Falls der Änderungszeitpunkt gespeichert wird: Resultat = 1, andernfalls Resultat = 0.

FileSbrCount
Teildatensatzanzahl
Wert 4

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die die Anzahl der Teildatensätze einer Datei ermittelt werden kann.

_FileSeqInsert
Sequentielles Einfügen
Wert 12

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die festgestellt werden kann, ob die Datensätze sequentiell eingefügt werden.

Ist sequentielles Einfügen aktiv, wird 1 zurückgegeben, sonst 0.

_FileUserLevel
Benutzerberechtigung
Wert 7

Verwandte

Siehe Befehle,

FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die die Benutzerberechtigung (0 bis 255) des aktuellen Benutzers auf eine Datei ermittelt werden kann.

_FileTemp
Temporäre Datei
Wert 9

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die festgestellt werden kann, ob es sich um eine temporäre Datei handelt.

Ist die angegebene Datei eine temporäre Datei, wird 1 zurückgegeben, sonst 0.

_FileOEMMark
OEM-Kit-Markierung
Wert 8

Verwandte

Siehe Befehle, FileInfo(),
FileInfoByName()

Option bei FileInfo() und FileInfoByName() durch die die Markierung für das OEM-Kit ermittelt werden kann.

Falls die Markierung für das OEM-Kit markiert ist: Resultat = 1, andernfalls Resultat = 0.

Teildatensatzbefehle

Befehle zum Umgang mit Teildatensätzen


Siehe Befehlsgruppen,
Befehlsliste

Befehle

<u>SbrClear()</u>	Teildatensatz leeren
<u>SbrCopy()</u>	Teildatensatz kopieren
<u>SbrCompare()</u>	Teildatensätze vergleichen
<u>SbrInfo()</u>	Teildatensatzinformationen über Teildatensatznummer ermitteln
<u>SbrInfoByName()</u>	Teildatensatzinformationen über Teildatensatzname ermitteln
<u>SbrName()</u>	Teildatensatzname ermitteln
<u>SbrStatus()</u>	Teildatensatzstatus ermitteln

Konstanten

<u>_SbrExists</u>	Teildatensatzexistenz
<u>_SbrFileNumber</u>	Dateinummer
<u>_SbrFldCount</u>	Feldanzahl
<u>_SbrNumber</u>	Teildatensatznummer
<u>_SbrStatus</u>	Teildatensatzstatus

SbrName(int1, int2) : alpha 

Teildatensatzname ermitteln

int1 Dateinummer

int2 Teildatensatznummer

Resultat alpha Teildatensatzname

Siehe Verwandte Befehle

Diese Funktion ermittelt den Namen des angegebenen Teildatensatzes.

Mögliche Laufzeitfehler:

ErrNoSbr Teildatensatz ist nicht vorhanden



SbrInfo(int1, int2, int3) : int

Teildatensatzinformationen über Teildatensatznummer ermitteln

int1 Dateinummer /
 Datensatzpuffer-Deskriptor
 int2 Teildatensatznummer
 Informationstyp
 int3 SbrExists Teildatensatzexistenz
 SbrStatus Teildatensatzstatus
 SbrFldCount Feldanzahl

Resultat int Teildatensatzinformation

Siehe Verwandte Befehle,
 SbrInfoByName()

Mit dieser Funktion können verschiedene Informationen über einen Teildatensatz über die Teildatensatznummer ermittelt werden.

Die Rückgabewerte für die verschiedenen Informationen sind bei den Konstanten beschrieben.

Mögliche Laufzeitfehler:

ErrNoSbr Teildatensatz ist nicht vorhanden (nicht bei Verwendung von SbrExists)



SbrInfoByName(alpha1, int2) : int

Teildatensatzinformationen über Teildatensatzname ermitteln

alpha1 Teildatensatzname

Informationstyp

SbrExists Teildatensatzexistenz

SbrNumber Teildatensatznummer

int2

SbrFileNumber Dateinummer

SbrStatus Teildatensatzstatus

SbrFldCount Feldanzahl

Resultat int Teildatensatzinformation


Siehe Verwandte Befehle, SbrInfo()

Mit dieser Funktion können verschiedene Informationen über einen Teildatensatz über den Teildatensatznamen ermittelt werden.

Die Rückgabewerte für die verschiedenen Informationen sind bei den Konstanten beschrieben.

Mögliche Laufzeitfehler:

ErrNoSbr Teildatensatz ist nicht vorhanden (nicht bei Verwendung von SbrExists)

SbrStatus(int1, int2[, logic3]) : logic 
 Teildatensatzstatus setzen/ermitteln
 int1 Dateinummer
 int2 Teildatensatznummer
 logic3 Neuer Teildatensatzstatus
 (optional)

Resultat logic Aktueller Teildatensatzstatus

Siehe Verwandte Befehle


Mit dieser Anweisung kann zum einen die Aktivität des Teildatensatzes (int2) in der Datei (int1) ermittelt werden. Dabei werden nur zwei Argumente übergeben.

Zum anderen kann der Teildatensatz (sofern er bedingt ist und nicht feldabhängig) aktiviert (logic3 = true) oder deaktiviert (logic3 = false) werden. Der Status des Teildatensatzes ist nicht Bestandteil des Feldpuffers, wird also beim Kopieren des Feldpuffers mit RecBufCopy() oder SbrCopy() nicht mit kopiert. Der Status wird nur beim Speichern des Datensatzes ausgewertet und sollte auch erst kurz vor dem Speichern gesetzt werden.

Das Resultat ist der (neue) Status des Teildatensatzes.

Mögliche Laufzeitfehler:

ErrNoSbr Teildatensatz ist nicht vorhanden

SbrCompare(int1, int2, ) : logic

Teildatensätze vergleichen

int1 Dateinummer /
 Datensatzpuffer-Deskriptor
 (1. Teildatensatz)

int2 Teildatensatznummer (1.
 Teildatensatz)

int3 Dateinummer /
 Datensatzpuffer-Deskriptor
 (2. Teildatensatz)

int4 Teildatensatznummer (2.
 Teildatensatz)

Resultat logic Vergleichsresultat

Verwandte Befehle,

Siehe FldCompare(),
RecBufCompare()

Mit dieser Funktion wird der Inhalt des Teildatensatzes (int2) aus der Datei (int1) mit dem Inhalt des Teildatensatzes (int4) in der Datei (int3) verglichen. Enthalten die Teildatensätze unterschiedlich viele Felder, so werden nur die Felder verglichen, die in beiden Teildatensätzen vorhanden sind.


Das Vergleichsresultat ist true, wenn alle verglichenen Felder vom gleichen Typ sind und den gleichen Inhalt haben. Ansonsten ist das Resultat false.

Mögliche Laufzeitfehler:

_ErrNoFile Es wurde eine nicht vorhandene Datei in (int1) oder (int3) angegeben.

_ErrNoSbr Einer der Teildatensätze (int2) oder (int4) ist nicht vorhanden.

_ErrHdlInvalid Es wurde ein ungültiger Deskriptor in (int1) oder (int3) angegeben.

SbrCopy(int1, int2, )
int3, int4)

Teildatensatz kopieren


Dateinummer /
int1 Datensatzpuffer-Deskriptor
(Quellteildatensatz)
int2 Teildatensatznummer
(Quellteildatensatz)
Dateinummer /
int3 Datensatzpuffer-Deskriptor
(Zielteildatensatz)
int4 Teildatensatznummer
(Zielteildatensatz)

Siehe Verwandte Befehle,
FldCopy(), RecBufCopy()

Mit dieser Funktion wird der Inhalt des Teildatensatzes (int2) aus der Datei (int1) in den Teildatensatz (int4) der Datei (int3) kopiert. Das Kopieren des Inhaltes wird feldweise durchgeführt. Das Kopieren wird beendet, sobald zwei Felder unterschiedlichen Typs auftreten. Dadurch können auch Inhalte zwischen Teildatensätzen kopiert werden, die nur in den ersten Feldern übereinstimmen. Beim Kopieren von alphanumerischen Werten werden diese abgeschnitten, wenn das Zielfeld kürzer als der zu kopierende Wert ist.

Mögliche Laufzeitfehler:

ErrNoFile Es wurde eine nicht vorhandene Datei in (int1) oder (int3) angegeben.
ErrNoSbr Einer der Teildatensätze (int2) oder (int4) ist nicht vorhanden.
ErrHdlInvalid Es wurde ein ungültiger Deskriptor in (int1) oder (int3) angegeben.

SbrClear(int1, int2) 

Teildatensatz leeren

int1 Dateinummer oder
Deskriptor eines
Datensatzpuffers

int2 Teildatensatznummer

Siehe Verwandte Befehle,
RecBufClear()

Diese Funktion leert alle Felder des angegebenen Teildatensatzes (int2) der Datei (int1). Ist in (int1) ein Datensatzpuffer angegeben (siehe RecBufCreate()), wird der Teildatensatz (int2) in diesem geleert.

Mögliche Laufzeitfehler:

_ErrNoSbr Teildatensatz ist nicht vorhanden

Konstanten für Teildatensatzbefehle

Konstanten für Teildatensatzbefehle

Siehe Teildatensatzbefehle

<u>SbrExists</u>	Teildatensatzexistenz
<u>SbrNumber</u>	Teildatensatznummer
<u>SbrFileNumber</u>	Dateinummer
<u>SbrStatus</u>	Teildatensatzstatus
<u>SbrFldCount</u>	Feldanzahl

SbrExists
Teildatensatzexistenz
Wert 0

Verwandte

Siehe Befehle, SbrInfo(),
SbrInfoByName()

Option bei SbrInfo() und SbrInfoByName() durch die die Existenz eines
Teildatensatzes ermittelt werden kann.

Falls der Teildatensatz existiert: Resultat = 1, andernfalls Resultat = 0.

_SbrFileNumber

Dateinummer

Wert 1

Verwandte

Siehe Befehle,

SbrInfoByName()

Option bei SbrInfoByName() durch die die Dateinummer eines Teildatensatzes ermittelt werden kann.

SbrFldCount

Feldanzahl

Wert 4

Verwandte

Siehe Befehle, SbrInfo(),

SbrInfoByName()

Option bei SbrInfo() und SbrInfoByName() durch die die Feldanzahl eines Teildatensatzes ermittelt werden kann.

SbrNumber
Teildatensatznummer
Wert 2

Verwandte

Siehe Befehle,

SbrInfoByName()

Option bei SbrInfoByName() durch die die Nummer eines Teildatensatzes ermittelt werden kann.

SbrStatus
Teildatensatzstatus
Wert 3

Verwandte

Siehe Befehle, SbrInfo(),
SbrInfoByName()

Option bei SbrInfo() und SbrInfoByName() durch die der Status eines Teildatensatzes ermittelt werden kann.

Falls der Teildatensatz aktiv ist: Resultat = 1, anderenfalls Resultat = 0.

Feldbefehle

Befehle zum Umgang mit Feldern

Siehe Befehlsgruppen,
Befehlsliste

Befehle

<u>Fld...</u> ()	Feldinhalt über Feldnummer ermitteln
<u>Fld...ByName</u> ()	Feldinhalt über Feldname ermitteln
<u>FldAttributes</u> ()	ODBC-Attribute setzen/ermitteln
<u>FldCompare</u> ()	Feldinhalte vergleichen
<u>FldCopy</u> ()	Feldinhalt kopieren
<u>FldDef</u> ()	Feldinhalt über Feldnummer setzen
<u>FldDefByName</u> ()	Feldinhalt über Feldname setzen
<u>FldInfo</u> ()	Feldinformationen über Feldnummer ermitteln
<u>FldInfoByName</u> ()	Feldinformationen über Feldname ermitteln
<u>FldName</u> ()	Feldname ermitteln

Konstanten

<u>_FldExists</u>	Feldexistenz
<u>_FldFileNumber</u>	Dateinummer
<u>_FldInputRight</u>	Eingabeberechtigung
<u>_FldLen</u>	Feldlänge
<u>_FldNumber</u>	Feldnummer
<u>_FldOutputRight</u>	Ausgabeberechtigung
<u>_FldSbrNumber</u>	Teildatensatznummer
<u>_FldType</u>	Feldtyp



Fld...(int1, int2, int3) : var
Feldinhalt über Feldnummer ermitteln

int1 Dateinummer oder
 Datensatzpuffer-Deskriptor

int2 Teildatensatznummer

int3 Feldnummer

Resultat var Feldinhalt

Verwandte Befehle,

Siehe Fld...ByName(), FldDef(),
RecBufCreate()

Diese Funktionen ermöglichen es, ein Feld in einem Ausdruck nicht fest anzugeben, sondern während der Abarbeitung der Prozedur bestimmen zu lassen, welches Feld tatsächlich benutzt wird. Dabei sind die numerischen Parameter (int1), (int2) und (int3) die Dateinummer, die Teildatensatznummer und die Feldnummer des gewünschten Feldes.

Folgende Befehle können zum Auslesen der unterschiedlichen Feldtypen verwendet werden:

- FldAlpha()
- FldBigInt()
- FldDate()
- FldDecimal()
- FldFloat()
- FldInt()
- FldLogic()
- FldTime()
- FldWord()

Beispiel:

```
// Gibt den Wert eines ganzzahligen Feldes (16 Bit) mit der Nummer// 'Feldnummer' im Teildatensatz
```

Mögliche Laufzeitfehler:

<u>_ErrNoFld</u>	Teildatensatz (int2) oder Feld (int3) ist nicht vorhanden.
<u>_ErrFldType</u>	Feldtyp ist nicht korrekt.
<u>_ErrValueOverflow</u>	Der Inhalt eines <u>bigint</u> -Feldes wird einer <u>word</u> - oder <u>int</u> -Variablen zugewiesen und ist zu groß für diese.
<u>_ErrNoFile</u>	Es wurde eine nicht vorhandene Datei in (int1) angegeben.
<u>_ErrHdlInvalid</u>	Es wurde ein ungültiger Deskriptor in (int1) angegeben.



Fld...ByName(alpha1) : var
 Feldinhalt über Feldname ermitteln
 alpha1 Feldname
 Resultat var Feldinhalt

Verwandte

Siehe Befehle, Fld...(),
FldDefByName()

Diese Funktionen ermöglichen es, ein Feld in einem Ausdruck nicht fest anzugeben, sondern während der Abarbeitung der Prozedur bestimmen zu lassen, welches Feld tatsächlich benutzt wird. Dabei ist der alphanumerische Wert (alpha1) der Feldname des gewünschten Feldes.

Folgende Befehle können zum Auslesen der unterschiedlichen Feldtypen verwendet werden:

- FldAlphaByName()
- FldBigIntByName()
- FldDateByName()
- FldDecimalByName()
- FldFloatByName()
- FldIntByName()
- FldLogicByName()
- FldTimeByName()
- FldWordByName()

Beispiel

```
// Gibt den Wert des Datumsfeldes 'KND.dErstkontakt' zurück.tDate # FldDateByName('KND.dErstkontak
```

Mögliche Laufzeitfehler:

ErrNoFld Feld (alpha1) ist nicht vorhanden.

ErrValueOverflow Der Inhalt eines bigint-Feldes wird einer word- oder int-Variablen zugewiesen und ist zu groß für diese.

FldAttributes(int1, int2, int3[,int4]) :



int

ODBC-Attribute setzen/ermitteln

int1 Dateinummer

int2 Teildatensatznummer

int3 Feldnummer

Neue Feldattribute (optional)

_KeyFldAttrUpperCase Groß-/Kleinschreibung
beachten

int4 _KeyFldAttrUmlaut Umlaute beachten

_KeyFldAttrSpecialChars Sonderzeichen nicht
beachten

0 Parameter löschen

Resultat int Aktuelle Feldattribute

Siehe Verwandte Befehle, ODBC-Schnittstelle,
ODBC-Befehle



Dieser Befehl wird in der aktuellen Version 5.7 noch nicht unterstützt.

Beim Zugriff eines anderen Programms auf eine CONZEPT 16-Datenbank über die ODBC-Schnittstelle können unabhängig von den in der Datenbank vorliegenden Schlüsseln, Sortierfelder angegeben werden. Zum Beispiel in einem SELECT-Statement mit der Klausel ORDER BY.

Um die Sortierreihenfolge zu beeinflussen muss beim Einrichten der Datenquelle eine Startprozedur angegeben werden, in der mit dem Befehl FldAttributes() die Sortierparameter gesetzt werden. Sortierparameter können mit (int4 = 0) gelöscht werden.

Beispiele:

Die Datensätze sollen nach einem Suchwort sortiert ausgegeben werden. Das entsprechende ODBC-Statement lautet:

```
SELECT SWT_aSuchwort FROM SWT_D_Suchwort ORDER BY SWT_aSuchwort
```

Ohne die Verwendung der Anweisung FldAttributes() werden die Suchworte in folgender Reihenfolge zurückgegeben:

Aenderung

Zwischensumme

angeln

zeigen

Änderung

ändern

Die Reihenfolge entspricht der Wertigkeit der Zeichen in der ASCII-Tabelle.

Wird in der Startprozedur für das Feld SWT.aSuchwort die Anweisung FldAttributes(..., _KeyFldAttrUpperCase) angegeben, verändert sich die Reihenfolge

Kontakt

unter Verwendung des gleichen ODBC-Statements wie folgt:

Aenderung
angeln
zeigen
Zwischensumme
ändern
Änderung

Durch die Anweisung `FldAttributes(..., _KeyFldAttrUmlaut)` werden auch die Umlaute in alphabetischer Reihenfolge angezeigt:

ändern
Aenderung
Änderung
angeln
zeigen
Zwischensumme

FldCompare(int1, int2,
int3, int4, int5, int6) :



int

Feldinhalte vergleichen

int1 Dateinummer /
Datensatzpuffer-Deskriptor
(1. Feld)

int2 Teildatensatznummer (1.
Feld)

int3 Feldnummer (1. Feld)

int4 Dateinummer /
Datensatzpuffer-Deskriptor
(2. Feld)

int5 Teildatensatznummer (2.
Feld)

int6 Feldnummer (2. Feld)

Resultat int Vergleichsresultat

Siehe Verwandte Befehle,
RecBufCompare(),
RecBufCompareFld(),
SbrCompare()

Mit dieser Funktion wird der Inhalt des ersten Feldes mit dem Inhalt des zweiten Feldes verglichen. Folgende Vergleichsresultate sind möglich:

1 : 1. Feld > 2. Feld


0 : 1. Feld = 2. Feld

-1 : 1. Feld < 2. Feld

Der Vergleich der Feldinhalte findet über Dateien hinweg statt. Soll der Feldinhalt eines angelegten Feldpuffers mit dem Inhalt in den Feldpuffern der dazugehörigen Datei erfolgen, muss der Befehl RecBufCompareFld() verwendet werden.

Mögliche Laufzeitfehler:

- ErrNoFld Einer der beiden Teildatensätze (int2/int5) oder eines der beiden Felder (int3/int6) ist nicht vorhanden.
- ErrFldType Die Feldtypen sind unterschiedlich.
- ErrNoFile Es wurde eine nicht vorhandene Datei in (int1) oder (int4) angegeben.
- ErrHdlInvalid Es wurde ein ungültiger Deskriptor in (int1) oder (int4) angegeben.

FldCopy(int1, int2,
int3, int4, int5, 
int6)

Feldinhalt kopieren

Dateinummer /
int1 Datensatzpuffer-Deskriptor
(Quellfeld)
int2 Teildatensatznummer
(Quellfeld)
int3 Feldnummer (Quellfeld)
Dateinummer /
int4 Datensatzpuffer-Deskriptor
(Zielfeld)
int5 Teildatensatznummer
(Zieleld)
int6 Feldnummer (Zielfeld)

Siehe Verwandte Befehle,
RecBufCopy(), SbrCopy()

Mit dieser Funktion wird der Inhalt des ersten Feldes in das zweite Feld übertragen. Ist der Typ beider Felder unterschiedlich, findet keine Übertragung statt. Beim Kopieren von alphanumerischen Werten werden diese abgeschnitten, wenn das Zielfeld kürzer ist, als der zu kopierende Wert.

Mögliche Laufzeitfehler:

<u>ErrNoFld</u>	Einer der beiden Teildatensätze (int2/int5) oder eines der beiden Felder (int3/int6) ist nicht vorhanden.
<u>ErrNoFile</u>	Es wurde eine nicht vorhandene Datei in (int1) oder (int4) angegeben.
<u>ErrHdlInvalid</u>	Es wurde ein ungültiger Deskriptor in (int1) oder (int4) angegeben.

FldDef(int1, int2, int3, var4)



Feldinhalt über Feldnummer setzen

int1 Dateinummer oder
Datensatzpuffer-Deskriptor

int2 Teildatensatznummer

int3 Feldnummer

var4 Wert

Siehe Verwandte Befehle, Fld...(),
FldDefByName()

Einem variabel adressierten Feld wird durch diese Funktion ein Wert zugewiesen. Die Argumente (int1), (int2) und (int3) bezeichnen die Datei, den Teildatensatz und die Feldnummer des Feldes, dem der Wert (var4) zugewiesen werden soll. Es ist wichtig, dass der Typ des berechneten Feldes mit dem Typ des zuzuweisenden Werts übereinstimmt, da sonst der Laufzeitfehler _ErrFldType erfolgt.

Soll ein Feld geleert werden, kann unabhängig vom Feldtyp der Wert NULL angegeben werden.



Wird einem Feld vom Typ word bzw. vom Typ int ein Wert außerhalb des Bereiches 0 bis 65535 (bei word) bzw. -2147483648 bis 2147483647 (bei int) zugewiesen, werden nur die hinteren 2 bzw. 4 Byte beachtet.

Beispiel

```
// Schreibt den Wert 49 in das Feld mit der Nummer 'FldNo' im Teildatensatz 1 der Datei 46.FldNo
```

Mögliche Laufzeitfehler:

<u>_ErrNoFld</u>	Teildatensatz (int2) oder Feld (int3) ist nicht vorhanden.
<u>_ErrFldType</u>	Feldtyp ist nicht korrekt.
<u>_ErrValueOverflow</u>	Beim Setzen eines <u>word</u> - oder <u>int</u> -Feldes wurde eine <u>bigint</u> -Variable angegeben und ist zu groß für dieses.
<u>_ErrStringOverflow</u>	Beim Setzen eines <u>alpha</u> -Feldes ist der Wert (var4) zu lang.
<u>_ErrNoFile</u>	Es wurde eine nicht vorhandene Datei in (int1) angegeben.
<u>_ErrHdlInvalid</u>	Es wurde ein ungültiger Deskriptor in (int1) angegeben.



FldInfo(int1, int2, int3, int4) : int

Feldinformationen über Feldnummer ermitteln

int1 Dateinummer

int2 Teildatensatznummer

int3 Feldnummer

Informationstyp

FldExists Feld vorhanden

FldInputRight Eingabeberechtigung

int4 FldLen Feldlänge

FldOutputRight Ausgabeberechtigung

FldType Feldtyp

Resultat int Feldinformationen

Siehe Verwandte Befehle, FldInfoByName()

Mit dieser Funktion können verschiedene Informationen über ein Feld über dessen Nummer abgefragt werden. Details zu den abgefragten Informationen befinden sich bei der Beschreibung der zu übergebenen Konstanten.

Mögliche Laufzeitfehler:

ErrNoFld Feld ist nicht vorhanden (nicht bei Verwendung von FldExists)



FldInfoByName(alpha1, int2) : int

Feldinformationen über Feldname ermitteln

alpha1 Feldname

Informationstyp

FldExists Feld vorhanden

FldFileNumber Dateinummer

FldInputRight Eingabeberechtigung

int2 FldLen Feldlänge

FldNumber Feldnummer

FldOutputRight Ausgabeberechtigung

FldSbrNumber Teildatensatznummer

FldType Feldtyp


Resultat int Feldinformationen

Siehe Verwandte Befehle, FldInfo()

Mit dieser Funktion können verschiedene Informationen über ein Feld über dessen Name abgefragt werden. Details zu den abgefragten Informationen befinden sich bei der Beschreibung der zu übergebenen Konstanten.

Mögliche Laufzeitfehler:

ErrNoFld Feld ist nicht vorhanden (nicht bei Verwendung von FldExists)

FldName(int1, int2, ) : alpha

Feldname ermitteln

int1 Dateinummer

int2 Teildatensatznummer

int3 Feldnummer

Resultat alpha Feldname

Siehe Verwandte Befehle

Die Funktion ermittelt den Namen eines Feldes.

Mögliche Laufzeitfehler:

_ErrNoFld Feld ist nicht vorhanden

Konstanten für Feldbefehle

Konstanten für Feldbefehle

Siehe Feldbefehle

<u>FldExists</u>	Feldexistenz
<u>FldFileNumber</u>	Dateinummer
<u>FldInputRight</u>	Eingabeberechtigung
<u>FldLen</u>	Feldlänge
<u>FldNumber</u>	Feldnummer
<u>FldOutputRight</u>	Ausgabeberechtigung
<u>FldSbrNumber</u>	Teildatensatznummer
<u>FldType</u>	Feldtyp

FldExists
Feldexistenz
Wert 0

Verwandte

Siehe Befehle, FldInfo(),
FldInfoByName()

Option bei FldInfo() und FldInfoByName() durch die die Existenz eines Feldes ermittelt werden kann.

Falls das Feld existiert: Resultat = 1, andernfalls Resultat = 0.

_FldFileNumber

Dateinummer

Wert 1

Verwandte

Siehe Befehle,

FldInfoByName()

Option bei FldInfoByName() durch die die Dateinummer eines Feldes ermittelt werden kann.

_FldInputRight
Eingabeberechtigung
Wert 6

Verwandte

Siehe Befehle, FldInfo(),
FldInfoByName()

Option bei FldInfo() und FldInfoByName() durch die die Eingabeberechtigung eines Feldes ermittelt werden kann.

_FldLen
Feldlänge
Wert 5

Verwandte

Siehe Befehle, FldInfo(),
FldInfoByName()

Option bei FldInfo() und FldInfoByName() durch die die Feldlänge eines Feldes ermittelt werden kann.

Bei alphanumerischen Feldern wird die maximale Länge zurückgeliefert.

_FldNumber

Feldnummer

Wert 3

Verwandte

Siehe Befehle,

FldInfoByName()

Option bei FldInfoByName() durch die die Feldnummer eines Feldes ermittelt werden kann.

_FldOutputRight
Ausgabeberechtigung
Wert 7

Verwandte

Siehe Befehle, FldInfo(),
FldInfoByName()

Option bei FldInfo() und FldInfoByName() durch die die Ausgabeberechtigung eines Feldes ermittelt werden kann.

FldSbrNumber
Teildatensatznummer
Wert 2

Verwandte

Siehe Befehle,

FldInfoByName()

Option bei FldInfoByName() durch die die Teildatensatznummer eines Feldes ermittelt werden kann.

_FldType
 Feldtyp
 Wert 4

Verwandte

Siehe Befehle, FldInfo(),
FldInfoByName()

Option bei FldInfo() und FldInfoByName() durch die der Feldtyp eines Feldes ermittelt werden kann.

Der ermittelte Wert kann mit folgenden Konstanten verglichen werden:

<u>_TypeAlpha</u>	Alphanumerisch
<u>_TypeBigInt</u>	Ganzzahlig (64 Bit)
<u>_TypeDate</u>	Datum
<u>_TypeDecimal</u>	Dezimal
<u>_TypeFloat</u>	Gleitkomma
<u>_TypeInt</u>	Ganzzahlig (32 Bit)
<u>_TypeLogic</u>	Logisch
<u>_TypeTime</u>	Zeit
<u>_TypeWord</u>	Ganzzahlig (16 Bit)

Schlüsselbefehle

Befehle zum Ermitteln von Schlüsselinformationen


Siehe Befehlsgruppen,
Befehlsliste

Befehle

<u>KeyInfo()</u>	Schlüsselinformationen über Schlüsselnummer ermitteln
<u>KeyInfoByName()</u>	Schlüsselinformationen über Schlüsselname ermitteln
<u>KeyName()</u>	Schlüsselname ermitteln
<u>KeyFldInfo()</u>	Schlüsselfeldinformationen ermitteln

Konstanten

<u>_KeyExists</u>	Schlüsselexistenz
<u>_KeyFileNumber</u>	Dateinummer
<u>_KeyNumber</u>	Schlüsselnummer
<u>_KeyIsUnique</u>	Schlüsseleindeutigkeit
<u>_KeyFldAttributes</u>	Schlüsselfeldattribute
<u>_KeyFldAttrReverse</u>	Absteigende Sortierung
<u>_KeyFldAttrSoundex1</u>	Umwandlung in Soundex Stufe 1
<u>_KeyFldAttrSoundex2</u>	Umwandlung in Soundex Stufe 2
<u>_KeyFldAttrSpecialChars</u>	Sortierung ohne Sonderzeichen
<u>_KeyFldAttrUmlaut</u>	Umlaute in alphabetischer Sortierung
<u>_KeyFldAttrUpperCase</u>	Groß-/Kleinwandlung
<u>_KeyFldCount</u>	Schlüsselfeldanzahl
<u>_KeyFldExists</u>	Schlüsselfeldexistenz
<u>_KeyFldFileNumber</u>	Dateinummer
<u>_KeyFldMaxLen</u>	Maximale Feldlänge
<u>_KeyFldNumber</u>	Feldnummer
<u>_KeyFldSbrNumber</u>	Teildatensatznummer
<u>_KeyFldType</u>	Feldtyp

KeyName(int1, int2) : 

alpha

Schlüsselname ermitteln

int1 Dateinummer

int2 Schlüsselnummer

Resultat alpha Schlüsselname

Siehe Verwandte Befehle

Mit dieser Funktion kann der Name des angegebenen Schlüssels ermittelt werden. In den Parametern wird die Dateinummer (int1) und die Schlüsselnummer (int2) angegeben.

Mögliche Laufzeitfehler:

ErrNoKey Schlüssel ist nicht vorhanden



KeyInfo(int1, int2, int3) : int

Schlüsselinformationen über Schlüsselnummer ermitteln

int1 Dateinummer

int2 Schlüsselnummer

Informationstyp

KeyExists Schlüsselexistenz

KeyFldCount Schlüsselfeldanzahl

int3 KeyIsUnique Schlüsseleindeutigkeit

KeyMaxLen Maximale
Schlüssellänge

Resultat int Schlüsselinformation

Verwandte Befehle,

Siehe KeyInfoByName(), KeyFldInfo(),
Schlüssel

Mit dieser Funktion können verschiedene Informationen über einen Schlüssel über die Schlüsselnummer ermittelt werden.

Mögliche Laufzeitfehler:

ErrNoKey Schlüssel ist nicht vorhanden (nicht bei Verwendung von KeyExists)



KeyInfoByName(alpha1, int2) : int

Schlüsselinformationen über Schlüsselname ermitteln

alpha1 Schlüsselname

Informationstyp

KeyExists Schlüsselexistenz

KeyNumber Schlüsselnummer

int2 KeyFileNumber Dateinummer

KeyFldCount Schlüsselfeldanzahl

KeyIsUnique Schlüsseleindeutigkeit

KeyMaxLen Maximale
Schlüssellänge


Resultat int Schlüsselinformation

Siehe Verwandte Befehle, KeyInfo(),
KeyFldInfo(), Schlüssel

Mit dieser Funktion können verschiedene Informationen über einen Schlüssel über den Schlüsselname ermittelt werden.

Mögliche Laufzeitfehler:

ErrNoKey Schlüssel ist nicht vorhanden (nicht bei Verwendung von KeyExists)

KeyFldInfo(int1, int2, int3, int4) : int 

Schlüsselfeldinformationen ermitteln

int1 Dateinummer

int2 Schlüsselnummer

int3 Schlüsselfeldnummer

Informationstyp

KeyFldExists Schlüsselfeldexistenz

KeyFldNumber Feldnummer

int4 KeyFldSbrNumber Teildatensatznummer

KeyFldFileNumber Dateinummer

KeyFldType Feldtyp

KeyFldAttributes Feldattribute

KeyFldMaxLen Maximale Feldlänge

Resultat int Schlüsselinformation

Siehe Verwandte Befehle, KeyInfo(), Schlüssel

Mit dieser Funktion können verschiedene Informationen über ein Schlüsselfeld ermittelt werden.

Mögliche Laufzeitfehler:

ErrNoKey Schlüssel ist nicht vorhanden

ErrNoKeyFld Schlüsselfeld ist nicht vorhanden (nicht bei Verwendung von KeyFldExists)

Konstanten für Schlüsselbefehle

Konstanten für Schlüsselbefehle

Siehe Schlüsselbefehle

<u>KeyExists</u>	Schlüsselexistenz
<u>KeyFileNumber</u>	Dateinummer
<u>KeyNumber</u>	Schlüsselnummer
<u>KeyIsUnique</u>	Schlüsseleindeutigkeit
<u>KeyFldAttributes</u>	Schlüsselfeldattribute
<u>KeyFldAttrReverse</u>	Absteigende Sortierung
<u>KeyFldAttrSoundex1</u>	Umwandlung in Soundex Stufe 1
<u>KeyFldAttrSoundex2</u>	Umwandlung in Soundex Stufe 2
<u>KeyFldAttrSpecialChars</u>	Sortierung ohne Sonderzeichen
<u>KeyFldAttrUmlaut</u>	Umlaute in alphabetischer Sortierung
<u>KeyFldAttrUpperCase</u>	Groß-/Kleinwandlung
<u>KeyFldCount</u>	Schlüsselfeldanzahl
<u>KeyFldExists</u>	Schlüsselfeldexistenz
<u>KeyFldFileNumber</u>	Dateinummer
<u>KeyFldMaxLen</u>	Maximale Feldlänge
<u>KeyFldNumber</u>	Feldnummer
<u>KeyFldSbrNumber</u>	Teildatensatznummer
<u>KeyFldType</u>	Feldtyp

_KeyExists

Schlüsselexistenz

Wert 0

Verwandte

Befehle,

Siehe KeyInfo(),

KeyInfoByName(),

Schlüssel

Option bei KeyInfo() und KeyInfoByName() durch die die Existenz eines Schlüssels ermittelt werden kann.

Falls der Schlüssel existiert: Resultat = 1, andernfalls Resultat = 0.

_KeyFileNumber

Dateinummer

Wert 1

Verwandte

Siehe Befehle,

KeyInfoByName(),

Schlüssel

Option bei KeyInfoByName() durch die die Dateinummer eines Schlüssels ermittelt werden kann.

_KeyNumber
Schlüsselnummer
Wert 2

Verwandte
Siehe Befehle,
KeyInfoByName(),
Schlüssel

Option bei KeyInfoByName() durch die die Schlüsselnummer eines Schlüssels ermittelt werden kann.

_KeyIsUnique
Schlüsseleindeutigkeit
Wert 4

Verwandte

Befehle,

Siehe KeyInfo(),
KeyInfoByName(),
Schlüssel

Option bei KeyInfo() und KeyInfoByName() durch die die Eindeutigkeit eines Schlüssels ermittelt werden kann.

Falls der Schlüssel eindeutig ist: Resultat = 1, andernfalls Resultat = 0.

_KeyFldAttributes
Schlüselfeldattribute
Wert 5

Verwandte
Siehe Befehle,
KeyFldInfo(),
Schlüssel

Option bei KeyFldInfo() durch die die Attribute eines Schlüsselfeldes ermittelt werden können.

Folgende Attribute sind definiert:

<u>_KeyFldAttrReverse</u>	Absteigende Sortierung
<u>_KeyFldAttrSoundex1</u>	Umwandlung in Soundex Stufe 1
<u>_KeyFldAttrSoundex2</u>	Umwandlung in Soundex Stufe 2
<u>_KeyFldAttrSpecialChars</u>	Sortierung ohne Sonderzeichen
<u>_KeyFldAttrUmlaut</u>	Umlaute in alphabetischer Sortierung
<u>_KeyFldAttrUpperCase</u>	Groß-/Kleinwandlung

Die jeweiligen Attribute bestehen aus einzelnen Bits des Resultats. Um ein Attribut abzufragen, muss das Resultat mit einer binären Binär-UND (&)-Verknüpfung kombiniert werden.

Beispiel:

```
// Überprüfen ob das Schlüsselfeld 5 im Schlüssel 3 der// Datei 1 mit absteigender Sortierung der
```


_KeyFldAttrReverse
Absteigende Sortierung
Wert 64 / 0x40

Verwandte
Befehle,
Siehe KeyFldInfo(),
LinkFldInfo(),
SelAddSortFld(),
Schlüssel

Rückgabewert von KeyFldInfo() mit der Option _KeyFldAttributes und LinkFldInfo() mit der Option _LinkFldAttributes durch den angezeigt wird, dass für das Schlüsselfeld eine absteigende Sortierung durchgeführt wird.

Bei der Anweisung SelAddSortFld() kann diese Option angegeben werden, um eine umgekehrte Sortierung der Selektion zu erhalten.

_KeyFldAttrSoundex1
Umwandlung in Soundex Stufe 1
Wert 16 / 0x10

Verwandte
Befehle,
Siehe KeyFldInfo(),
LinkFldInfo(),
SelAddSortFld(),
Schlüssel

Rückgabewert von KeyFldInfo() mit der Option _KeyFldAttributes und LinkFldInfo() mit der Option _LinkFldAttributes durch den angezeigt wird, dass für das Schlüsselfeld eine Umwandlung in Soundex Stufe 1 durchgeführt wird.

Bei der Anweisung SelAddSortFld() kann diese Option angegeben werden, um eine Sortierung nach Soundex Stufe 1 zu erhalten.

_KeyFldAttrSoundex2
Umwandlung in Soundex Stufe 2
Wert 32 / 0x20

Verwandte
Befehle,
Siehe KeyFldInfo(),
LinkFldInfo(),
SelAddSortFld(),
Schlüssel

Rückgabewert von KeyFldInfo() mit der Option _KeyFldAttributes und LinkFldInfo() mit der Option _LinkFldAttributes durch den angezeigt wird, dass für das Schlüsselfeld eine Umwandlung in Soundex Stufe 2 durchgeführt wird.

Bei der Anweisung SelAddSortFld() kann diese Option angegeben werden, um eine Sortierung nach Soundex Stufe 2 zu erhalten.

_KeyFldAttrSpecialChars
Sortierung ohne Sonderzeichen
Wert 8 / 0x08

Verwandte
Befehle,
Siehe KeyFldInfo(),
LinkFldInfo(),
SelAddSortFld(),
Schlüssel

Option bei FldAttributes() durch die eine Sortierung ohne Sonderzeichen durchgeführt wird.

Rückgabewert von KeyFldInfo() mit der Option _KeyFldAttributes und LinkFldInfo() mit der Option _LinkFldAttributes durch den angezeigt wird, dass für das Schlüsselfeld eine Sortierung ohne Sonderzeichen durchgeführt wird.

Bei der Anweisung SelAddSortFld() kann diese Konstante angegeben werden, wenn die Sortierung innerhalb der Selektion ohne Sonderzeichen erfolgen soll.

_KeyFldAttrUmlaut
Umlaute in alphabetischer Sortierung
Wert 4 / 0x04

Verwandte
Befehle,
Siehe KeyFldInfo(),
LinkFldInfo(),
SelAddSortFld(),
Schlüssel

Option bei FldAttributes() durch die eine Sortierung der Umlaute in alphabetischer Reihenfolge durchgeführt wird.

Rückgabewert von KeyFldInfo() mit der Option _KeyFldAttributes und LinkFldInfo() mit der Option _LinkFldAttributes durch den angezeigt wird, dass für das Schlüsselfeld eine Sortierung der Umlaute in alphabetischer Reihenfolge durchgeführt wird.

Bei der Anweisung SelAddSortFld() wird diese Konstante übergeben, wenn eine Sortierung innerhalb der Selektion ohne Umlaute erfolgen soll.

_KeyFldAttrUpperCase
Groß-/Kleinwandlung
Wert 2 / 0x02

Verwandte
Befehle,
Siehe KeyFldInfo(),
LinkFldInfo(),
SelAddSortFld(),
Schlüssel

Option bei FldAttributes() durch die eine Groß-/Kleinwandlung des Feldinhaltes durchgeführt wird.

Rückgabewert von KeyFldInfo() mit der Option _KeyFldAttributes und LinkFldInfo() mit der Option _LinkFldAttributes durch den angezeigt wird, dass für das Schlüsselfeld eine Groß-/Kleinwandlung durchgeführt wird.

Bei der Anweisung SelAddSortFld() kann diese Konstante angegeben werden, wenn innerhalb der Selektionsmenge eine Sortierung unabhängig von der Groß-/Kleinschreibung erfolgen soll.

_KeyFldCount
Schlüselfeldanzahl
Wert 3

Verwandte

Befehle,

Siehe KeyInfo(),
KeyInfoByName(),
Schlüssel

Option bei KeyInfo() und KeyInfoByName() durch die die Anzahl der Schlüsselfelder ermittelt werden kann.

_KeyFldExists
Schlüselfeldexistenz
Wert 0

Verwandte
Siehe Befehle,
KeyFldInfo(),
Schlüssel

Option bei KeyFldInfo() durch die die Existenz eines Schlüsselfeldes ermittelt werden kann.

Falls das Schlüsselfeld existiert: Resultat = 1, andernfalls Resultat = 0.

_KeyFldFileNumber

Dateinummer

Wert 1

Verwandte

Siehe Befehle,

KeyFldInfo(),

Schlüssel

Option bei KeyFldInfo() durch die die Dateinummer ermittelt werden kann.

_KeyFldMaxLen
Maximale Feldlänge
Wert 6

Verwandte
Siehe Befehle,
KeyFldInfo(),
Schlüssel

Option bei KeyFldInfo() durch die die maximale Feldlänge ermittelt werden kann.

Falls das Schlüsselfeld nicht alphanumerisch ist oder keine Maximallänge definiert ist:
Resultat = 0.

_KeyFldNumber

Feldnummer

Wert 3

Verwandte

Siehe Befehle,

KeyFldInfo(),

Schlüssel

Option bei KeyFldInfo() durch die die Feldnummer ermittelt werden kann.

_KeyFldSbrNumber
Teildatensatznummer
Wert 2

Verwandte

Siehe Befehle,
KeyFldInfo(),
Schlüssel

Option bei KeyFldInfo() durch die die Teildatensatznummer ermittelt werden kann.

_KeyFldType

Feldtyp

Wert 4

Verwandte

Siehe Befehle,

KeyFldInfo(),

Schlüssel

Option bei KeyFldInfo() durch die der Feldtyp ermittelt werden kann.

Der ermittelte Wert kann mit folgenden Konstanten verglichen werden:

<u>_TypeAlpha</u>	Alphanumerisch
<u>_TypeBigInt</u>	Ganzzahlig (64 Bit)
<u>_TypeDate</u>	Datum
<u>_TypeDecimal</u>	Dezimal
<u>_TypeFloat</u>	Gleitkomma
<u>_TypeInt</u>	Ganzzahlig (32 Bit)
<u>_TypeLogic</u>	Logisch
<u>_TypeTime</u>	Zeit
<u>_TypeWord</u>	Ganzzahlig (16 Bit)

_KeyMaxLen
Lange Schlüsselwerte
Wert 5

Verwandte
Siehe Befehle,
KeyInfoByName(),
Schlüssel

Option bei KeyInfo() und KeyInfoByName() durch die die maximale Schlüssellänge ermittelt werden kann. Sofern die Option "Lange Schlüsselwerte" in der Datenstruktureditor aktiviert wurde, wird 950 ansonsten 240 zurückgegeben.

Filter

Beschreibung von Datensatzfiltern

Möglichkeiten der

Siehe Datensatzfilterung
(Blog)

Mit Hilfe von Filtern kann die Datenmenge eingeschränkt werden. Mit Filtern werden Bedingungen an die Schlüsselfelder eines Schlüssels geknüpft. Beim Zugriff über den Schlüssel werden vom Server nur Datensätze zurückgegeben, die die angegebenen Filterkriterien erfüllen.

Im Vergleich zu Verknüpfungen oder Selektionen sollte ein Filter nur dann verwendet werden, wenn die Anzahl der Datensätze, die die Filterkriterien erfüllen, groß ist. Der Server muss bei einem Zugriff auf einen Datensatz solange Datensätze lesen, bis er einen Datensatz gefunden hat, der zur Treffermenge gehört. Hat die Treffermenge zum Beispiel nur einen Datensatz und dieser ist der letzte in der Schlüsselreihenfolge, muss beim Lesen des ersten Datensatzes über diesen Filter die gesamte Datei gelesen werden. Dadurch entsteht der Eindruck einer langsamen Applikation.

Filter können nur auf Schlüsselfelder eines Schlüssels definiert werden. Eine Filterung nach Informationen, die nicht in einem Schlüssel verwendet werden oder Kombinationen von Feldinhalten von Schlüsselfeldern mehrerer Schlüssel, ist nicht möglich.

Bedingungen

Alle einschränkende Felder befinden sich in einem Schlüssel.
Es wird nur auf einen Schlüsselwert eingeschränkt.

Alle einschränkende Felder befinden sich in einem Schlüssel.
Es gibt mehrere gültige Schlüsselwerte (zum Beispiel einen Bereich).

Die Bedingungen müssen nicht geklammert werden.

-

Dieses Kapitel gliedert sich in folgende Abschnitte:

- Filter definieren
- Zugriff über Filter
- Filter löschen
- Befehle
- Konstanten

-> **Objekt**

-> Verknüpfung

-> Filter

-> Selektion

Filter definieren

Filter werden mit der Anweisung RecFilterCreate() erzeugt. Bei der Anweisung werden die Datei und der Schlüssel angegeben. Die einzelnen Filter-Kriterien werden mit der Anweisung RecFilterAdd() definiert.

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(2, _FltAND, _FltLikeC
```

In diesem Beispiel werden alle Datensätze gefiltert, deren Bezeichnungen mit einem "M" beginnen. Die Bezeichnung steht im zweiten Schlüsselfeld des Schlüssels keyArtType. Es können weitere Kriterien auf demselben Schlüsselfeld oder einem

anderen Schlüsselfeld des gleichen Schlüssels definiert werden:

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(1, _FltAND, _FltEq, 'S')
```

Schlüsselfeld 1 entspricht der Artikelgruppe (S steht für Sachbuch), Schlüsselfeld 2 enthält die Artikelbezeichnung. Mit diesem Filter werden alle Sachbücher deren Artikelbezeichnungen mit einem "M" beginnen und ein "p" enthalten gefiltert.

Die einzelnen Filterkriterien können nicht nur mit _FltAND kombiniert werden:

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(1, _FltAND, _FltEq, 'S')
```

Da eine Klammerung nicht möglich ist, ist die Reihenfolge der Angabe der Kriterien entscheidend dafür, welche Datensätze gefiltert werden. Der oben angegebenen Filter enthält nicht alle Sachbücher und Bücher, deren Bezeichnungen mit "M" beginnen und ein "p" beinhalten, sondern alle Sachbücher, die ein "p" in der Bezeichnung haben und alle Bücher deren Bezeichnungen mit "M" beginnen und ein "p" beinhalten. Werden die Kriterien in der Reihenfolge geändert, werden andere Datensätze gefiltert. Mit folgender Definition bekommen wir die gewünschte Menge:

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(2, _FltAND, _FltLikeC)
```

Zugriff über Filter

Der Zugriff auf die gefilterte Menge kann über die Eigenschaft DbFilter erfolgen (Objekte RecList, RecListPopup und PrintDocRecord). Der Filterdeskriptor muss dabei solange zur Verfügung stehen, wie das Objekt auf den Filter zugreift. In den Eigenschaften DbFileNo und DbKeyNo müssen die Datei und der Schlüssel angegeben werden, für den der Filter erzeugt wurde.

Ebenso kann bei den Anweisungen RecRead() und RecLink() ein Filterdeskriptor übergeben werden. Auch hier müssen angegebene Datei, Schlüssel und Filter zusammenpassen.

Im folgenden Beispiel werden alle Datensätze gelesen, die ein angegebenes Filterkriterium erfüllen:

```
...for tErr # RecRead(tblArtArticle, keyArtType, _RecFirst, tFilter);loop tErr # RecRead(tblAr
```

Bei einem Zugriff über eine Verknüpfung (RecLink()), werden alle verknüpften Datensätze gefiltert. Der Filter muss dabei auf dem Schlüssel definiert sein, der bei der Verknüpfung angegeben wurde.

Filter löschen

Wird der Filter nicht mehr benötigt (die Objekte, bei denen der Filter angegeben wurde, müssen geschlossen sein), kann der Deskriptor mit der Anweisung RecFilterDestroy() gelöscht werden. Ein Zugriff über den Filter ist anschließend nicht mehr möglich.

```
...tFilter->RecFilterDestroy();...
```


Befehle:

- RecFilterAdd
- RecFilterCreate
- RecFilterDestroy

Konstanten für Filterbefehle

- _FltAbove
- _FltAboveEq
- _FltAND
- _FltBelow
- _FltBelowEq
- _FltEq
- _FltLike
- _FltLikeCI
- _FltOR
- _FltScan
- _FltUneq
- _FltXOR

Verknüpfungen

Verknüpfen von Datensätzen

Befehlsgruppen,

Befehlsliste,

Verknüpfung

Siehe bearbeiten,

Möglichkeiten der

Datensatzfilterung

(Blog)

Ausgehend von einem Datensatz können mit der Verknüpfung Datensätze in einer anderen Datei gelesen werden. Verknüpfungen sind Bestandteile der Datenstruktur und werden im Datenstruktureditor über Verknüpfung bearbeiten erstellt und verändert.

Ein Beispiel für die Verwendung von Verknüpfungen ist die Kunden-Ansprechpartner-Relation. Dabei werden die Kunden und die Ansprechpartner in unterschiedlichen Dateien gespeichert. Durch die Verknüpfung können ausgehend von einem Kunden alle seine Ansprechpartner gelesen werden.

Voraussetzungen für eine Verknüpfung

Um eine Verknüpfung verwenden zu können, werden eine Quell-Datei und eine Ziel-Datei benötigt. Quelle und Ziel einer Verknüpfung können nicht die gleiche Datei sein. In der Ziel-Datei muss eine Information aus der Quell-Datei vorliegen (der Fremdschlüssel) und dieses Feld muss das erste Feld eines Schlüssels sein. Bei einer 1:n-Verknüpfung ist der Schlüssel mehrdeutig, bei einer 1:1-Verknüpfung eindeutig. In der Quell-Datei kann dann eine Verknüpfung erstellt werden. In dieser werden die Ziel-Datei und der Schlüssel angegeben. Die Verknüpfungsfelder befinden sich in der Quell-Datei und bestimmen die Schlüsselwerte für den Schlüssel in der Ziel-Datei.

Beispiel:

Kunden und Ansprechpartner sind in den Dateien tblCstCustomer und tblConContact gespeichert. Die Kunden werden über eine Kundennummer fiCstId eindeutig identifiziert. Damit eine Verknüpfung erstellt werden kann, wird die Kundennummer ebenfalls in der Datei der Ansprechpartner in dem Feld fiConCstId gespeichert. Für dieses Feld wird ein Schlüssel keyConCstId definiert (siehe Schlüssel bearbeiten). Der Schlüssel ist mehrdeutig, da ein Kunde mehrere Ansprechpartner haben kann (1:n-Relation).

In der Datei tblCstCustomer wird die Verknüpfung lnkCstCon angelegt. In die Ziel-Datei tblConContact wird über den Schlüssel keyConCstId verknüpft. Als Verknüpfungsfeld wird fiCstId angegeben (siehe Verknüpfung bearbeiten).

Besteht der Schlüssel in der Ziel-Datei aus mehreren Schlüsselfeldern, können auch mehrere Verknüpfungsfelder angegeben werden. Der Schlüsselwert wird in der Reihenfolge der Verknüpfungsfelder gebildet. Die verknüpften Datensätze werden in der Reihenfolge des Schlüssels gelesen. In der Verknüpfung können Felder angegeben werden, die nur für die Positionierung innerhalb der Reihenfolge verwendet werden (siehe Verknüpfung bearbeiten), diese schränken den Zugriff nicht ein.

Zugriff auf verknüpfte Datensätze

Der Zugriff auf verknüpfte Datensätze erfolgt immer in zwei Schritten. Zunächst muss ein Datensatz in der Quell-Datei gelesen bzw. die Verknüpfungsfelder gesetzt werden. Die Datensätze können anschließend über die Verknüpfung in der Ziel-Datei gelesen werden. Verknüpfte Datensätze können in den Objekten RecList, RecListPopup und PrintDocRecord angezeigt werden (siehe Eigenschaft DbLinkFileNo). Prozedural erfolgt der Zugriff mit der Anweisung RecLink(). Es werden nur die Datensätze in der Ziel-Datei ermittelt, die den gleichen Schlüsselwert besitzen, der in den Verknüpfungsfeldern der Quell-Datei angegeben ist.

Beispiel:

Folgende Schleife liest alle Ansprechpartner eines Kunden:

```
... fiCstId # 1; // Verknüpfungsfeld in der Quelldatei for tErr # RecLink(tblConContact, tb
```

Verwendung von Verknüpfungen

Neben der Abbildung von Relationen zwischen Datenbanktabellen kann die Verknüpfung zur Einschränkung der zu lesenden Datensätze verwendet werden. Mit Hilfe der Verknüpfung können alle Datensätze in der Zieldatei gelesen werden, die einen bestimmten Schlüsselwert besitzen. Die Felder müssen in einem Schlüssel der Ziel-Datei angegeben sein (siehe auch Filtern und Selektionen).

Bedingungen

Alle einschränkenden Felder befinden sich in einem Schlüssel.
Es wird nur auf einen Schlüsselwert eingeschränkt.

Alle einschränkenden Felder befinden sich in einem Schlüssel.
Es gibt mehrere gültige Schlüsselwerte (zum Beispiel einen Bereich).

Die Bedingungen müssen nicht geklammert werden.

-

n:m-Verknüpfungen kommen vergleichsweise zu 1:n- und 1:1-Verknüpfungen selten vor. Sie können durch zwei 1:n-Verknüpfungen abgebildet werden. Dazu wird in einer weiteren Tabelle eine Verbindung zwischen zwei 1:n-Verknüpfungen definiert.

-> **Objekt**

-> Verknüpfung

-> Filter

-> Selektion

Auf diese Weise können zum Beispiel alle Artikel, die in einem Jahr verkauft wurden, über Verknüpfungen ermittelt werden. Wird diese Information nur selten benötigt, ist eine Selektion besser geeignet.

Befehle

- LinkFldInfo
- LinkInfo
- LinkInfoByName
- LinkName
- RecLink
- RecLinkInfo

Konstanten

- LinkDestFileNumber
- LinkDestKeyNumber
- LinkExists
- LinkFileNumber
- LinkFldAttributes
- LinkFldAttrPosition
- LinkFldCount
- LinkFldExists
- LinkFldFileNumber
- LinkFldMaxLen
- LinkFldNumber
- LinkFldSbrNumber
- LinkFldType
- LinkNumber



LinkInfoByName(alpha1, int2) : int

Verknüpfungsinformationen über Verknüpfungsname ermitteln

alpha1 Verknüpfungsname

Informationstyp

LinkExists Verknüpfungsexistenz

LinkNumber Verknüpfungsnummer

LinkFileNumber Dateinummer

int2 LinkDestFileNumber Dateinummer der
Zieldatei

LinkDestKeyNumber Schlüsselnummer der
Zieldatei

LinkFldCount Verknüpfungsfeldanzahl


Resultat int Verknüpfungsinformation

Siehe Verwandte Befehle, LinkInfo()

Mit dieser Funktion können verschiedene Informationen über eine Verknüpfung über den Verknüpfungsname ermittelt werden. Die Bedeutung des Rückgabewertes ist bei der entsprechenden Konstante erläutert.

Mögliche Laufzeitfehler:

ErrNoLink Verknüpfung ist nicht vorhanden (nicht bei Verwendung von LinkExists)

LinkName(int1, int2) : alpha 

Verknüpfungsname ermitteln

int1 Dateinummer

int2 Verknüpfungsnummer

Resultat alpha Verknüpfungsname

Siehe Verwandte Befehle

Mit dieser Funktion kann der Name einer Verknüpfung ermittelt werden.

Mögliche Laufzeitfehler:

ErrNoLink Verknüpfung ist nicht vorhanden



LinkFldInfo(int1, int2, int3, int4) : int

Verknüpfungsfeldinformationen ermitteln

int1 Dateinummer

int2 Verknüpfungsnummer

int3 Verknüpfungsfeldnummer

Informationstyp

LinkFldExists Verknüpfungsfeldexistenz

LinkFldNumber Feldnummer

int4 LinkFldSbrNumber Teildatensatznummer

LinkFldFileNumber Dateinummer

LinkFldType Feldtyp

LinkFldAttributes Feldattribute

LinkFldMaxLen Maximale Feldlänge

Resultat int Verknüpfungsinformation

Siehe Verwandte Befehle

Mit dieser Funktion können verschiedene Informationen über ein Verknüpfungsfeld ermittelt werden. Die Bedeutung der Rückgabewerte wird bei den übergebenen Konstanten erläutert.

Mögliche Laufzeitfehler:

ErrNoLink Verknüpfung ist nicht vorhanden

ErrNoLinkFld Verknüpfungsfeld ist nicht vorhanden (nicht bei Verwendung von LinkFldExists)

Konstanten für Verknüpfungsbefehle

Konstanten für Verknüpfungsbefehle

Siehe Verknüpfungsbefehle

<u>LinkExists</u>	Verknüpfungsexistenz
<u>LinkFileNumber</u>	Dateinummer
<u>LinkNumber</u>	Verknüpfungsnummer
<u>LinkDestFileNumber</u>	Dateinummer der Zieldatei
<u>LinkDestKeyNumber</u>	Schlüsselnummer der Zieldatei
<u>LinkFldAttributes</u>	Verknüpfungsfeldattribute
<u>LinkFldAttrPosition</u>	Zugriffspositionierung
<u>LinkFldCount</u>	Verknüpfungsfeldanzahl
<u>LinkFldExists</u>	Verknüpfungsfeldexistenz
<u>LinkFldMaxLen</u>	Maximale Feldlänge
<u>LinkFldNumber</u>	Feldnummer
<u>LinkFldFileNumber</u>	Dateinummer
<u>LinkFldSbrNumber</u>	Teildatensatznummer
<u>LinkFldType</u>	Feldtyp

LinkExists

Verknüpfungsexistenz

Wert 0

Verwandte Befehle,

Siehe LinkInfo(),

LinkInfoByName()

Option bei LinkInfo() und LinkInfoByName() durch die die Existenz einer Verknüpfung ermittelt werden kann.

Falls die Verknüpfung existiert: Resultat = 1, andernfalls Resultat = 0.

LinkNumber

Verknüpfungsnummer

Wert 2

Siehe Verwandte Befehle,
LinkInfoByName()

Option bei LinkInfoByName() durch die die Verknüpfungsnummer ermittelt werden kann.

LinkFileNumber

Dateinummer

Wert 1

Siehe Verwandte Befehle,
LinkInfoByName()

Option bei LinkInfoByName() durch die die Dateinummer ermittelt werden kann.

_LinkDestFileNumber
Dateinummer der Zielfdatei
Wert 4

Verwandte Befehle,

Siehe LinkInfo(),
LinkInfoByName()

Option bei LinkInfo() und LinkInfoByName() durch die die Dateinummer der Zielfdatei ermittelt werden kann.

_LinkDestKeyNumber

Schlüsselnummer der Zielfeld

Wert 5

Verwandte Befehle,

Siehe LinkInfo(),

LinkInfoByName()

Option bei LinkInfo() und LinkInfoByName() durch die die Schlüsselnummer der Zielfeld ermittelt werden kann.

LinkFldAttributes
Verknüpfungsfeldattribute
Wert 5

Verwandte
Siehe Befehle,
LinkFldInfo(),
KeyFldAttributes

Option bei LinkFldInfo() durch die die Attribute eines Verknüpfungsfeldes ermittelt werden können.

Folgende Attribute sind definiert:

<u>KeyFldAttrReverse</u>	Absteigende Sortierung
<u>KeyFldAttrSoundex1</u>	Umwandlung in Soundex Stufe 1
<u>KeyFldAttrSoundex2</u>	Umwandlung in Soundex Stufe 2
<u>KeyFldAttrSpecialChars</u>	Sortierung ohne Sonderzeichen
<u>KeyFldAttrUmlaut</u>	Umlaute in alphabetischer Sortierung
<u>KeyFldAttrUpperCase</u>	Groß-/Kleinwandlung
<u>LinkFldAttrPosition</u>	Zugriffspositionierung

_LinkFldAttrPosition
Zugriffspositionierung
Wert 128 / 0x80

Verwandte

Siehe Befehle,

LinkFldInfo()

Rückgabewert von LinkFldInfo() mit der Option _LinkFldAttributes durch den angezeigt wird, dass für das Schlüsselfeld nur eine Zugriffspositionierung durchgeführt wird.

LinkFldCount

Verknüpfungsfeldanzahl

Wert 3

Verwandte Befehle,

Siehe LinkInfo(),

LinkInfoByName()

Option bei LinkInfo() und LinkInfoByName() durch die die Anzahl der Verknüpfungsfelder ermittelt werden kann.

LinkFldExists
Verknüpfungsfeldexistenz
Wert 0

Verwandte

Siehe Befehle,

LinkFldInfo()

Option bei LinkFldInfo() durch die die Existenz eines Verknüpfungsfeldes ermittelt werden kann.

LinkFldFileNumber

Dateinummer

Wert 1

Verwandte

Siehe Befehle,

LinkFldInfo()

Option bei LinkFldInfo() durch die die Dateinummer ermittelt werden kann.

LinkFldMaxLen
Maximale Feldlänge
Wert 6

Verwandte

Siehe Befehle,

LinkFldInfo()

Option bei LinkFldInfo() durch die die maximale Feldlänge ermittelt werden kann.

Falls das Verknüpfungsfeld nicht alphanumerisch ist oder keine Maximallänge definiert ist: Resultat = 0.

LinkFldNumber

Feldnummer

Wert 3

Verwandte

Siehe Befehle,

LinkFldInfo()

Option bei LinkFldInfo() durch die die Feldnummer ermittelt werden kann.

LinkFldSbrNumber
Teildatensatznummer
Wert 2

Verwandte

Siehe Befehle,

LinkFldInfo()

Option bei LinkFldInfo() durch die die Teildatensatznummer ermittelt werden kann.

_LinkFldType

Feldtyp

Wert 4

Verwandte

Siehe Befehle,

LinkFldInfo()

Option bei LinkFldInfo() durch die der Feldtyp ermittelt werden kann.

Der ermittelte Wert kann mit folgenden Konstanten verglichen werden:

<u>_TypeAlpha</u>	Alphanumerisch
<u>_TypeBigInt</u>	Ganzzahlig (64 Bit)
<u>_TypeDate</u>	Datum
<u>_TypeDecimal</u>	Dezimal
<u>_TypeFloat</u>	Gleitkomma
<u>_TypeInt</u>	Ganzzahlig (32 Bit)
<u>_TypeLogic</u>	Logisch
<u>_TypeTime</u>	Zeit
<u>_TypeWord</u>	Ganzzahlig (16 Bit)

Transaktionsbefehle

Befehle zum Durchführen von Transaktionen

Siehe Befehlsgruppen,
Befehlsliste

DtaBegin() Transaktion starten

DtaCommit() Transaktion beenden

DtaRollback() Transaktion zurücksetzen

DtaBegin()



Transaktion starten

VerwandteBefehle,

Siehe

DtaCommit(),DtaRollback()

Ab dem Aufruf dieser Systemfunktion ist eine Transaktion aktiviert, alle nachfolgenden Datenoperationen werden als Einheit betrachtet, bis der Befehl DtaCommit() oder DtaRollback() erfolgt. Innerhalb einer Transaktion können weitere Transaktionen gestartet werden. Die Anzahl der geschachtelten Transaktionen kann mit der Anweisung DbInfo(_DbaDtaLevel) ermittelt werden.

Transaktionen, die mit DtaBegin() gestartet wurden, können nicht mit den Befehlen aus den A- Prozeduren (TransOff() und TransBrk()) beendet werden.

Alle Datensätze, die an einer Transaktion beteiligt sind, bleiben bis zum Ende der Transaktion gesperrt. Andere Benutzer können auf diese Datensätze zugreifen, aber keine Änderungen vornehmen. Transaktionen, die einen längeren Zeitraum beanspruchen, sollten nach Möglichkeit vermieden werden, da andere Benutzer zu lange auf diese Datensätze warten müssen.



Bei der Programmierung mit Transaktionen ist darauf zu achten, dass jedem DtaBegin() ein DtaCommit() oder ein DtaRollback() folgt, da sonst alle weiteren Änderungen in der Datenbank (einschließlich der Prozeduren) bis zur Rückkehr in den Designer oder zur Abmeldung des Benutzers verloren gehen, da die Transaktion nicht beendet wurde. Die unvollständige Transaktion wird zurückgesetzt.



Im Gegensatz zu den Datensatzbefehlen erfolgen Sperren innerhalb von Transaktionen auf Datenbanksegmente. Ein Datenbanksegment kann, je nach Größe der Datensätze, mehrere Datensätze der gleichen Tabelle beinhalten. Wird ein Datensatz innerhalb einer Transaktion verändert, wird das gesamte Segment gesperrt, also auch der Datensatz, der nicht an der Transaktion beteiligt ist. Soll, während ein Benutzer ein Segment per Transaktion gesperrt hat, von einem anderen Benutzer ein Datensatz in dem Segment eingefügt, geändert oder gelöscht werden, kommt es zu einem Wartezustand.

Änderungen an den folgenden Datenbankinhalten werden von der Transaktion berücksichtigt:

- Datensätze
- Prozeduren
- Texte
- Binäre Verzeichnisse und Objekte
- Dialog-Objekte
- Bilder in der Ressourcenverwaltung
- Selektionen

Der Wartezustand wird bei der Verwendung des CONZEPT 16-Clients nach wenigen Sekunden durch ein Fenster angezeigt, dass am Rand des Bildschirms eingeblendet wird:



Im Fenster wird der eigene Datenbankbenutzer und die eigene User-ID angezeigt. Darunter wird der Datenbankbenutzer, die ID und der Rechner des Benutzers angezeigt, auf den gewartet werden muss.

Das Fenster kann mit der Maus am Rand des Bildschirms verschoben werden. Der Titeltext und die Vordergrundfarbe des Titeltexes kann über die Eigenschaften Caption und CaptionColor des _App-Objekts verändert werden.

Die Operation innerhalb der zweiten Transaktion kann erst fortgeführt werden, wenn das Ergebnis der ersten Transaktion feststeht. Durch die Sperrung auf Segmentebene kann es ebenfalls beim Einfügen eines neuen Datensatzes zu einem Wartezustand kommen, wenn der neue Datensatz in das gesperrte Segment eingefügt werden soll.

Um unnötige Wartezustände zu vermeiden, können über den Parameter _RecEarlyCommit einzelne Befehle von der Transaktion ausgenommen werden.

Kommt es bei zwei oder mehreren Transaktionen zu einer Verklemmung (mindestens zwei Transaktionen warten auf ein Segment, das von der jeweils anderen Transaktion gesperrt ist), wird die Transaktion, die bislang am wenigsten Zeit benötigt hat, abgebrochen. Änderungen in den Datensätzen werden nicht durchgeführt und der Befehl, an dem die Transaktion abgebrochen wurde, gibt den Rückgabewert rDeadlock zurück. Anschließend wird die Prozedur fortgesetzt. Mit der Auswertung des Rückgabewertes kann der Programmierer den Programmzustand zu Beginn der Transaktion wieder herstellen und die Transaktion erneut starten.

DtaCommit()



Transaktion beenden

Verwandte

Siehe Befehle,

DtaBegin(),

DtaRollback()

Diese Funktion beendet eine mit DtaBegin() angefangene Transaktion. Alle während der Transaktion vorgenommenen Datenänderungen werden endgültig gespeichert.

Transaktionen, die mit TransOn in einer A- Prozedur gestartet wurden, können nicht mit diesem Befehl übernommen werden. Es muss ein Aufruf in eine A- Prozedur (CallOld()) mit dem Befehl TransOff() durchgeführt werden.



Änderungen an den folgenden Datenbankinhalten werden von der Transaktion berücksichtigt:

- Datensätze
- Prozeduren
- Texte
- Binäre Verzeichnisse und Objekte
- Dialog-Objekte
- Bilder in der Ressourcenverwaltung
- Selektionen



Kehrt eine Prozedur mit offenen Transaktionen in den Designer zurück, werden die offenen Transaktionen abgebrochen.

DtaRollback(logic1)



Transaktion zurücksetzen

Alle

logic1 Transaktionen
zurücksetzenVerwandteSiehe Befehle,DtaBegin(),DtaCommit()

Diese Funktion bricht eine mit DtaBegin() angefangene Transaktion ab. Alle während der Transaktion vorgenommenen Datenänderungen werden rückgängig gemacht und der Zustand vor dem Transaktionsbeginn ist wieder hergestellt.



Änderungen an den folgenden Datenbankinhalten werden von der Transaktion berücksichtigt:

- Datensätze
- Prozeduren
- Texte
- Binäre Verzeichnisse und Objekte
- Dialog-Objekte
- Bilder in der Ressourcenverwaltung
- Selektionen



Sperren auf Datensätzen, Texten, binären Objekten und Selektionen werden nicht zurückgesetzt.

Transaktionen, die mit TransOn() in einer A- Prozedur gestartet wurden, können nicht mit diesem Befehl verworfen werden. Es muss ein Aufruf in einer A- Prozedur (CallOld()) mit dem Befehl TransBrk() durchgeführt werden.

Bei geschachtelten Transaktionen kann durch (logic1) bestimmt werden, ob alle offenen Transaktionen (true) oder nur die letzte Transaktion (false) zurückgesetzt werden.

Kommt es bei der Verarbeitung von mehreren Transaktionen zu einer Verklemmung (mindestens zwei Transaktionen warten auf ein Segment, dass von der jeweils anderen Transaktion gesperrt ist), wird die Transaktion, die weniger Zeit benötigt hat, abgebrochen. Dieser Abbruch entspricht einem DtaRollback(). Der Befehl, der nicht ausgeführt werden konnte gibt den Rückgabewert rDeadlock zurück. Es werden lediglich die Änderungen in den Datensätzen verworfen. Der Programmierer kann den Rückgabewert auswerten und den Programmzustand bei Beginn der Transaktion wieder herstellen (zum Beispiel extern geschriebene Dateien zurücksetzen, globale Variablen wieder herstellen usw.), um anschließend die Transaktion erneut durchzuführen.

Datensatzbefehle

Befehle zum Umgang mit Datensätzen

Siehe Befehlsgruppen,
Befehlsliste

Befehle

- RecDelete
- RecDeleteAll
- RecInfo
- RecInsert
- RecLink
- RecLinkInfo
- RecRead
- RecReplace

Konstanten für Datensatzbefehle

- RecCheckLock
- RecCount
- RecCountNext
- RecCountPos
- RecEarlyCommit
- RecFirst
- RecForceLock
- RecGetPos
- RecGetPosReverse
- RecGetPrime
- RecID
- RecKeyReverse
- RecLast
- RecLen
- RecLenPacked
- RecLock
- RecLockedBy
- RecModified
- RecNext
- RecNextID
- RecNoLoad
- RecPos
- RecPrev
- RecSetID
- RecSharedLock
- RecSingleLock
- RecTest
- RecUnlock
- RecUpdateCounter
- RecUseId

Konstanten für Filterbefehle

<u>FltAbove</u>	Größer-Vergleich
<u>FltAboveEq</u>	Größer/Gleich-Vergleich
<u>FltAND</u>	UND-Verknüpfung
<u>FltBelow</u>	Kleiner-Vergleich
<u>FltBelowEq</u>	Kleiner/Gleich-Vergleich
<u>FltEq</u>	Gleich-Vergleich
<u>FltLike</u>	Ähnlich-Überprüfung
<u>FltOR</u>	ODER-Verknüpfung
<u>FltScan</u>	Enthält-Überprüfung
<u>FltXOR</u>	Exklusive ODER-Verknüpfung

Datensatzpuffer

Zugriff auf die Datensätze über den Datensatzpuffer

Der Zugriff unter CONZEPT 16 erfolgt immer auf einen Datensatz, nicht auf eine Menge von Datensätzen. Wird ein Datensatz gelesen (zum Beispiel mit dem Befehl RecRead()), befindet er sich anschließend im Datensatzpuffer. Der Inhalt des Datensatzes kann ausgelesen werden, indem die Namen der Felder (vgl. Abschnitt Feld bearbeiten) wie Variablen des entsprechenden Feldtyps verwendet werden.

Auf die gleiche Weise kann auch der Inhalt eines Datensatzes verändert werden. Dem Datensatzpuffer wird einfach ein neuer Wert zugewiesen. Die Änderung befindet sich dann allerdings noch nicht in der Datenbank. Dazu muss der Datensatz noch mit dem Befehl RecReplace() oder RecInsert() in die Datenbank übertragen werden.

Beispiel:

```
// Ändern des DatensatzpuffersfiCstId      # 1;faCstName    # 'vectorsoft AG';faCstStreet # 'Selige
```

Der Datensatzpuffer wird bei den verschiedenen Befehlen durch die Nummer bzw. dem Namen der Datei repräsentiert.

Der Entwickler kann eigene Datensatzpuffer anlegen. Dies erfolgt mit dem Befehl RecBufCreate(). Der von dem Befehl zurückgegebene Deskriptor kann ebenfalls anstelle der Dateinummer an die verschiedenen Datensatzbefehle übergeben werden. Es wird dann nicht der Standard-Datensatzpuffer der Datei verwendet, sondern der von dem Entwickler angelegte. Verschiedene Objekte können über eigene Datensatzpuffer verfügen. Diese können über die Eigenschaft DbRecBuf ermittelt werden. Der Zugriff auf die Felder eines angelegten oder ermittelten Datensatzpuffers ist ebenfalls möglich. Dazu wird der Deskriptor gefolgt von einem Pfeil und dem Namen des Feldes angegeben.

Beispiel:

```
// Erzeugen des eigenen DatensatzpufferstHdlRecBuf # RecBufCreate(tblCstCustomer);// Ändern des D
```

Damit die Standard-Datensatzpuffer und die vom Entwickler angelegten Datensatzpuffer mit den gleichen Funktionen verarbeitet werden können, kann mit dem Befehl RecBufDefault() der Deskriptor der Standard-Datensatzpuffer ermittelt werden.

Befehle:

- RecBufClear
- RecBufCompare
- RecBufCompareFld
- RecBufCopy
- RecBufCreate
- RecBufDefault
- RecBufDestroy

RecBufClear(int1[,
logic1])



Datensatzpuffer leeren

int1 Dateinummer /
 Datensatzpuffer-Deskriptor

logic2 Datensatz-ID ebenfalls
 leeren (optional)

Siehe Verwandte Befehle,
RecBufCreate(), SbrClear()

Diese Funktion leert alle Felder einer Datei oder eines, z. B. mit RecBufCreate() erzeugten, Datensatzpuffers (int1). Soll die Datensatz-ID ebenfalls geleert werden, muss in (logic2) true angegeben werden, ansonsten false.



Wird der Befehl RecBufClear() auf einen Datensatzpuffer angewendet, wird die Datensatz-ID aus kompatibilitätsgründen ohne die Angabe des Parameters (logic2) geleert.

Nach der Anweisung sind die Felder mit dem Wert NULL initialisiert.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Deskriptor ungültig

RecBufCompare(int1, int2) :



logic

Datensatzpuffer vergleichen

int1 1. Dateinummer / 1.
Datensatzpuffer-Deskriptor

int2 2. Dateinummer / 2.
Datensatzpuffer-Deskriptor

Resultat logic Vergleichsresultat

Verwandte Befehle,

Siehe RecBufCompareFld(),

SbrCompare(),

FldCompare()

Diese Funktion vergleicht den Pufferinhalt der 1. Datei (int1) mit dem Pufferinhalt der 2. Datei (int2).

Falls die beiden Pufferstrukturen und inhalte übereinstimmen: Vergleichsresultat = true, ansonsten Vergleichsresultat = false.

Intern führt RecBufCompare() für jeden Teildatensatz, der in beiden Dateien existiert, ein SbrCompare() durch.

Mögliche Laufzeitfehler:

_ErrNoFile Eine der beiden Dateien ist nicht vorhanden.

_ErrHdlInvalid Einer der beiden Deskriptoren ist ungültig.

RecBufCompareFld(int1, int2,
int3[,int4]) : logic



Datensatzpufferfelder vergleichen

int1 1. Dateinummer /
 Datensatzpuffer-Deskriptor

int2 Teildatensatznummer

int3 Feldnummer

 2. Dateinummer /
int4 Datensatzpuffer-Deskriptor
 (optional)

Resultat logic Vergleichsresultat

Siehe Verwandte Befehle,
 RecBufCompare(),
 SbrCompare(),
 FldCompare()

Diese Funktion vergleicht den Inhalt des Feldes (int3) im Teildatensatz (int2) des ersten Datensatzpuffer (int1) mit dem Inhalt des entsprechenden Feldes im zweiten Datensatzpuffer (int4). Wird der zweite Datensatzpuffer (int4) nicht angegeben, wird das Feld mit dem entsprechenden Feld der globalen Feldpuffer verglichen.

Falls die beiden Feldinhalte übereinstimmen ist das Vergleichsresultat true, andernfalls false.


Mögliche Laufzeitfehler:

_ErrNoSbr Teildatensatz (int2) ist nicht vorhanden.

_ErrNoFld Feld (int3) ist nicht vorhanden.

_ErrNoFile Eine der beiden Dateien (int1) oder (int4) ist nicht vorhanden.

_ErrHdlInvalid Einer der beiden Deskriptoren (int1) oder (int4) ist ungültig.

RecBufCopy(int1, int2[, 
logic3])

Datensatzpuffer kopieren

int1 Dateinummer /
 Datensatzpuffer-Deskriptor
 (Quelle)

int2 Dateinummer /
 Datensatzpuffer-Deskriptor
 (Ziel)

logic3 Datensatz-ID mitkopieren
 (optional)

Siehe [Verwandte Befehle](#),
[SbrCopy\(\)](#), [FldCopy\(\)](#)

Diese Funktion kopiert den Pufferinhalt der Quelle (int1) in die Puffer des Ziels (int2). Quelle und Ziel können dabei entweder eine Datei oder ein mit [RecBufDefault\(\)](#) bzw. [RecBufCreate\(\)](#) erzeugter Datensatzpuffer-Deskriptor sein. Mit dem Parameter (logic3) kann optional definiert werden, ob die [Datensatz-ID](#) ebenfalls mitkopiert werden soll ([true](#)) oder nicht ([false](#)).



Wird der Parameter (logic3) nicht angegeben, wird die Datensatz-ID aus kompatibilitätsgründen in jedem Fall mitkopiert.


Der Status eines Teildatensatzes (aktiv oder nicht aktiv) ist nicht Bestandteil des Datensatzpuffers und wird daher beim Kopieren des Datensatzpuffers nicht mit kopiert (siehe auch [SbrStatus\(\)](#)).

Das Kopieren des Inhaltes wird feldweise durchgeführt. Das Kopieren wird beendet, sobald zwei Felder unterschiedlichen Typs auftreten. Dadurch können auch Inhalte zwischen Dateien bzw. Datensatzpuffern kopiert werden, die nur in den ersten Feldern übereinstimmen. Beim Kopieren von alphanumerischen Werten werden diese abgeschnitten, wenn das Zielfeld kürzer als der zu kopierende Wert ist.

Mögliche Laufzeitfehler:

[_ErrNoFile](#) Eine der beiden Dateien ist nicht vorhanden.

[_ErrHdlInvalid](#) Einer der beiden Deskriptoren ist ungültig.

RecBufCreate(int1) : handle 
 Datensatzpuffer erzeugen
 int1 Dateinummer
 Resultat handle Datensatzpuffer-Deskriptor

Siehe Verwandte Befehle,
RecBufDestroy()

Mit dieser Funktion wird ein zusätzlicher Datensatzpuffer für die Datei (int1) im Hauptspeicher angelegt. Mit dem Deskriptor des Puffers können anschließend die Funktionen RecBufCopy(), RecBufCompare() und RecBufCompareFld() aufgerufen werden. Die Puffer können auch mit folgenden Datensatzbefehlen verwendet werden:

- RecRead()
- RecInsert()
- RecReplace()
- RecDelete()
- RecInfo()
- RecLinkInfo()

Wenn der Puffer nicht mehr benötigt wird, muss er mit RecBufDestroy() entfernt werden.

Die Anweisung gibt den Fehler _ErrGeneric zurück, falls die Datei kein einziges Feld enthält.

Nach der Anweisung sind die Felder mit dem Wert NULL initialisiert.

Mit Hilfe des Feldnamens kann direkt auf den entsprechenden Feldinhalt eines Datensatzpuffers sowohl lesend als auch schreibend zugegriffen werden.

Beispiel:

```
tHdl # RecBufCreate(100);tHdl->fiCstNo # 671;tHdl->faCstName # 'TEST';if (tHdl->fdCstContact < 0)
```

Da der Datensatzpuffer erst zur Laufzeit für eine bestimmte Datei angelegt wird, kann die Gültigkeit des Feldnamens nicht bei der Übersetzung geprüft werden.

Es entsteht ein Laufzeitfehler, wenn der Datensatzpuffer nicht angelegt ist oder das Feld nicht zur Datei des Puffers gehört.

Über die Funktion HdlInfo() mit der Option _HdlSubType kann ausgehend vom Deskriptor des Datensatzpuffers die Dateinummer ermittelt werden.

Die Verwendung von Dateien aus verbundenen Datenbanken (siehe DbaConnect()) ist möglich.

Mögliche Laufzeitfehler:

_ErrNoFile Datei ist nicht vorhanden



RecBufDefault(int1) : handle

Deskriptor des Standard-Datensatzpuffers

int1 Dateinummer

Resultat handle Deskriptor auf
Standard-Datensatzpuffer

Siehe Verwandte Befehle

Diese Anweisung gibt den Deskriptor auf den Standard-Datensatzpuffer der in (int1) übergebenen Datei zurück. Er erlaubt die Übergabe als Deskriptor auch für den Datensatzpuffer, der sonst über die Feldnamen angesprochen wird. So ist eine einheitliche Verarbeitung von selbst angelegten und bereits vorhandenen Standard-Datensatzpuffern möglich.


Die Verwendung von Dateien aus verbundenen Datenbanken (siehe DbConnect()) ist möglich.

Beispiel:

```
tHdlBuf # RecBufDefault(ADR.D.Adressen);tErg # tHdlBuf->RecRead(ADR.S.ID, 0);// entspricht der Ar
```

Mögliche Laufzeitfehler:

ErrNoFile Die angegebene Datei existiert nicht.

obj -> RecBufDestroy() 

Datensatzpuffer löschen

obj Datensatzpuffer-Deskriptor

Siehe Verwandte Befehle,

RecBufCreate()

Diese Funktion entfernt einen mit RecBufCreate() erzeugten Datensatzpuffer aus dem Hauptspeicher. Der Deskriptor kann anschließend nicht mehr verwendet werden.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Deskriptor ungültig

Filter

Beschreibung von Datensatzfiltern

Möglichkeiten der

Siehe Datensatzfilterung
(Blog)

Mit Hilfe von Filtern kann die Datenmenge eingeschränkt werden. Mit Filtern werden Bedingungen an die Schlüsselfelder eines Schlüssels geknüpft. Beim Zugriff über den Schlüssel werden vom Server nur Datensätze zurückgegeben, die die angegebenen Filterkriterien erfüllen.

Im Vergleich zu Verknüpfungen oder Selektionen sollte ein Filter nur dann verwendet werden, wenn die Anzahl der Datensätze, die die Filterkriterien erfüllen, groß ist. Der Server muss bei einem Zugriff auf einen Datensatz solange Datensätze lesen, bis er einen Datensatz gefunden hat, der zur Treffermenge gehört. Hat die Treffermenge zum Beispiel nur einen Datensatz und dieser ist der letzte in der Schlüsselreihenfolge, muss beim Lesen des ersten Datensatzes über diesen Filter die gesamte Datei gelesen werden. Dadurch entsteht der Eindruck einer langsamen Applikation.

Filter können nur auf Schlüsselfelder eines Schlüssels definiert werden. Eine Filterung nach Informationen, die nicht in einem Schlüssel verwendet werden oder Kombinationen von Feldinhalten von Schlüsselfeldern mehrerer Schlüssel, ist nicht möglich.

Bedingungen

Alle einschränkenden Felder befinden sich in einem Schlüssel.
Es wird nur auf einen Schlüsselwert eingeschränkt.

Alle einschränkenden Felder befinden sich in einem Schlüssel.
Es gibt mehrere gültige Schlüsselwerte (zum Beispiel einen Bereich).

Die Bedingungen müssen nicht geklammert werden.

-

Dieses Kapitel gliedert sich in folgende Abschnitte:

- Filter definieren
- Zugriff über Filter
- Filter löschen
- Befehle
- Konstanten

-> **Objekt**

-> Verknüpfung

-> Filter

-> Selektion

Filter definieren

Filter werden mit der Anweisung RecFilterCreate() erzeugt. Bei der Anweisung werden die Datei und der Schlüssel angegeben. Die einzelnen Filter-Kriterien werden mit der Anweisung RecFilterAdd() definiert.

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(2, _FltAND, _FltLikeC
```

In diesem Beispiel werden alle Datensätze gefiltert, deren Bezeichnungen mit einem "M" beginnen. Die Bezeichnung steht im zweiten Schlüsselfeld des Schlüssels keyArtType. Es können weitere Kriterien auf demselben Schlüsselfeld oder einem

anderen Schlüsselfeld des gleichen Schlüssels definiert werden:

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(1, _FltAND, _FltEq, 'S'
```

Schlüsselfeld 1 entspricht der Artikelgruppe (S steht für Sachbuch), Schlüsselfeld 2 enthält die Artikelbezeichnung. Mit diesem Filter werden alle Sachbücher deren Artikelbezeichnungen mit einem "M" beginnen und ein "p" enthalten gefiltert.

Die einzelnen Filterkriterien können nicht nur mit _FltAND kombiniert werden:

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(1, _FltAND, _FltEq, 'S'
```

Da eine Klammerung nicht möglich ist, ist die Reihenfolge der Angabe der Kriterien entscheidend dafür, welche Datensätze gefiltert werden. Der oben angegebenen Filter enthält nicht alle Sachbücher und Bücher, deren Bezeichnungen mit "M" beginnen und ein "p" beinhalten, sondern alle Sachbücher, die ein "p" in der Bezeichnung haben und alle Bücher deren Bezeichnungen mit "M" beginnen und ein "p" beinhalten. Werden die Kriterien in der Reihenfolge geändert, werden andere Datensätze gefiltert. Mit folgender Definition bekommen wir die gewünschte Menge:

```
tFilter # RecFilterCreate(tblArtArticle, keyArtType);tFilter->RecFilterAdd(2, _FltAND, _FltLikeC
```

Zugriff über Filter

Der Zugriff auf die gefilterte Menge kann über die Eigenschaft DbFilter erfolgen (Objekte RecList, RecListPopup und PrintDocRecord). Der Filterdeskriptor muss dabei solange zur Verfügung stehen, wie das Objekt auf den Filter zugreift. In den Eigenschaften DbFileNo und DbKeyNo müssen die Datei und der Schlüssel angegeben werden, für den der Filter erzeugt wurde.

Ebenso kann bei den Anweisungen RecRead() und RecLink() ein Filterdeskriptor übergeben werden. Auch hier müssen angegebene Datei, Schlüssel und Filter zusammenpassen.

Im folgenden Beispiel werden alle Datensätze gelesen, die ein angegebenes Filterkriterium erfüllen:

```
...for tErr # RecRead(tblArtArticle, keyArtType, _RecFirst, tFilter);loop tErr # RecRead(tblAr
```

Bei einem Zugriff über eine Verknüpfung (RecLink()), werden alle verknüpften Datensätze gefiltert. Der Filter muss dabei auf dem Schlüssel definiert sein, der bei der Verknüpfung angegeben wurde.

Filter löschen

Wird der Filter nicht mehr benötigt (die Objekte, bei denen der Filter angegeben wurde, müssen geschlossen sein), kann der Deskriptor mit der Anweisung RecFilterDestroy() gelöscht werden. Ein Zugriff über den Filter ist anschließend nicht mehr möglich.


```
...tFilter->RecFilterDestroy();...
```


Befehle:

- RecFilterAdd
- RecFilterCreate
- RecFilterDestroy

Konstanten für Filterbefehle

- _FltAbove
- _FltAboveEq
- _FltAND
- _FltBelow
- _FltBelowEq
- _FltEq
- _FltLike
- _FltLikeCI
- _FltOR
- _FltScan
- _FltUneq
- _FltXOR

obj -> RecFilterAdd(int1, )
 int2, int3, var4)

Datensatzfilter hinzufügen

obj Filter-Deskriptor

int1 Schlüsselfeldnummer

Kombination

FltAND UND-Verknüpfung

int2 FltOR ODER-Verknüpfung

FltXOR Exklusive

ODER-Verknüpfung

Vergleichsoperationen

FltAboveEq Größer/Gleich-Vergleich

FltAbove Größer-Vergleich

FltBelowEq Kleiner/Gleich-Vergleich

FltBelow Kleiner-Vergleich

FltEq Gleich-Vergleich

int3 FltLike Ähnlich-Überprüfung

FltLikeCI Ähnlich-Überprüfung

ohne Berücksichtigung
der

Groß-/Kleinschreibung

FltScan Enthält-Überprüfung

FltUneq Ungleich-Vergleich

var4 Vergleichswert

Siehe Verwandte Befehle,

RecFilterCreate()

Mit dieser Funktion werden Schlüsselkriterien zu einem Filter hinzugefügt (siehe RecFilterCreate()). Bis zu 100 Kriterien können für einen Filter definiert werden.

In (obj) wird der Deskriptor des Filters angegeben, in (int1) die Nummer des Schlüsselfelds, mit dem verglichen werden soll. Das neue Filterkriterium wird durch eine logische Operation in (int2) mit dem vorhergehenden Kriterium verbunden. Beim ersten Filterkriterium muss immer FltAND angegeben werden.

In (int3) steht die Art des Vergleichs. FltScan, FltLike und FltLikeCI können nur bei alphanumerischen Werten benutzt werden. Der eigentliche Vergleichswert wird in (var4) übergeben. Der Typ des Vergleichswerts muss mit dem Typ des Schlüsselfelds übereinstimmen.

Beispiel:


```
tFltHdl # RecFilterCreate(100, 2);// erstes Schlüsselfeld enthält "GmbH"RecFilterAdd(tFltHdl, 1,
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Filter-Deskriptor ungültig

ErrNoKeyFld Schlüsselfeld nicht vorhanden

ErrFldType Vergleichswerttyp nicht mit Schlüsselfeldtyp übereinstimmend

RecFilterCreate(int1, ) : handle

Datensatzfilter erzeugen

int1 Dateinummer oder
 Datensatzpuffer-Deskriptor

int2 Schlüsselnummer

Resultat handle Filter-Deskriptor

Verwandte Befehle,

RecFilterAdd(),

Siehe RecFilterDestroy(),

RecRead(), RecLink(),

DbFilter

Durch einen Filter kann der Zugriff über einen Schlüssel oder eine Verknüpfung auf ausgewählte Datensätze beschränkt werden. Filter werden für einen Schlüssel erzeugt. Alle Felder, die in diesem Schlüssel enthalten sind können mit der Anweisung RecFilterAdd() mit Kriterien belegt werden. Bei einem Zugriff über diesen Schlüssel werden dann nur noch die Datensätze, die den Kriterien genügen, gefunden. Sollte sich kein einziger Datensatz in der Datei befinden, der den Kriterien entspricht, erscheint die Datei als leer.

Beim Zugriff über eine Verknüpfung können ebenfalls Filter verwendet werden. Der Zugriff in die verknüpfte Datei erfolgt über den Schlüssel, der bei der Verknüpfung angegeben ist. Ist dieser Schlüssel mit einem Filter belegt, wird die Menge der verknüpften Datensätze durch das Filterkriterium eingeschränkt.

Ist ein Filter auf einen Schlüssel gesetzt, wirkt er sich nicht bei einem Zugriff über einen anderen Schlüssel aus.

Das Argument (int1) bezeichnet die Datei, (int2) den Schlüssel, auf den der Filter gelegt werden soll. Das Resultat ist der Deskriptor eines neuen Filters.

Um die Anzeige von Datensätzen zum Beispiel in einem RecList-Objekt einzuschränken, muss die Eigenschaft DbFilter auf den Filterdeskriptor gesetzt werden.

Beispiel 1:

```
// Dialog ladentFrame # WinOpen('Frm.AdrAuswahl', _WinOpenDialog);// Filter erzeugen, nur Adresse
```

Beispiel 2:

In der Datenstruktur sind zwei Dateien KND.D.Kunden und ASP.D.Ansprchpartner vorhanden. In der Kunden-Datei ist eine Verknüpfung definiert, die einen Schlüssel verwendet, in dem neben der Kunden-Nummer, der Name und die Anrede des Ansprechpartners enthalten ist.

Über folgenden Programmausschnitt werden, ausgehend vom Kunden, alle weiblichen Ansprechpartner ermittelt.

```
// Filter erzeugentFilter # RecFilterCreate(ASP.D.Ansprchpartner, ASP.S.KndIdNameAn);// Filterkri
```



Filter sollten verwendet werden, wenn die Anzahl der Datensätze, die das Filterkriterium erfüllen, hoch ist und sich diese Datensätze gleichmäßig über die Schlüsselreihenfolge verteilen. Als Alternative können auch Selektionen oder Verknüpfungen zum Einschränken der Anzahl der Datensätze verwendet werden.

Mögliche Laufzeitfehler:

ErrNoFile Datei (int1) nicht vorhanden

ErrNoKey Schlüssel (int2) nicht vorhanden

obj ->

RecFilterDestroy()

Datensatzfilter löschen

obj Filter-Deskriptor

Verwandte

Siehe Befehle,

RecFilterCreate()

Mit dieser Funktion wird ein mit RecFilterCreate() erzeugter Filter wieder gelöscht.
Nicht mehr benötigte Filter sollten in jedem Fall gelöscht werden, da der
CONZEPT 16-Server für den Filter Hauptspeicher belegt.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Filter-Deskriptor ungültig



Konstanten für Filterbefehle

Konstanten für Filterbefehle

Siehe Datensatzbefehle

<u>_FltAbove</u>	Größer-Vergleich
<u>_FltAboveEq</u>	Größer/Gleich-Vergleich
<u>_FltAND</u>	UND-Verknüpfung
<u>_FltBelow</u>	Kleiner-Vergleich
<u>_FltBelowEq</u>	Kleiner/Gleich-Vergleich
<u>_FltEq</u>	Gleich-Vergleich
<u>_FltLike</u>	Ähnlich-Überprüfung
<u>_FltLikeCI</u>	Ähnlich-Überprüfung ohne Berücksichtigung der Groß-/Kleinschreibung
<u>_FltOR</u>	ODER-Verknüpfung
<u>_FltScan</u>	Enthält-Überprüfung
<u>_FltXOR</u>	Exklusive ODER-Verknüpfung

_FltAbove
Größer-Vergleich
Wert 1

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die ein Größer-Vergleich (>) durchgeführt werden kann.

_FltAboveEq
Größer/Gleich-Vergleich
Wert 0

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die ein Größer/Gleich-Vergleich (\geq) durchgeführt werden kann.

_FltAND

UND-Verknüpfung

Wert 1

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die eine UND-Verknüpfung durchgeführt werden kann.

_FltBelow

Kleiner-Vergleich

Wert 2

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die ein Kleiner-Vergleich (<) durchgeführt werden kann.

_FltBelowEq
Kleiner/Gleich-Vergleich
Wert 3

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die ein Kleiner/Gleich-Vergleich (\leq) durchgeführt werden kann.

_FltEq

Gleich-Vergleich

Wert 6

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die ein Gleich-Vergleich (=) durchgeführt werden kann.

_FltLike
Ähnlich-Überprüfung
Wert 5

Verwandte

Befehle,

Siehe RecFilterAdd(),

_FltLikeCI,

_FltScan

Option bei RecFilterAdd() durch die eine Ähnlich-Überprüfung durchgeführt werden kann (analog Ähnlichkeitsoperator (=*)).

_FltLikeCI

Ähnlich-Überprüfung ohne Berücksichtigung der Groß-/Kleinschreibung

Wert 7

Verwandte

Befehle,

Siehe RecFilterAdd(),

_FltScan,

_FltLike

Option bei RecFilterAdd() durch die eine Ähnlich-Überprüfung ohne Berücksichtigung der Groß-/Kleinschreibung durchgeführt werden kann (analog Ähnlichkeitsoperator (=*^)).

_FltOR
ÖDER-Verknüpfung
Wert 2

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die eine ÖDER-Verknüpfung durchgeführt werden kann.

_FltScan
Enthält-Überprüfung
Wert 4

Verwandte
Befehle,
Siehe RecFilterAdd(),
_FltLike

Option bei RecFilterAdd() durch die eine Enthält-Überprüfung durchgeführt werden kann. Die Groß- und Kleinschreibung ist dabei nicht relevant.

_FltUneq

Ungleich-Vergleich

Wert 9

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die ein Ungleich-Vergleich (!=) durchgeführt werden kann.

_FltXOR
Exklusive ODER-Verknüpfung
Wert 3

Verwandte

Siehe Befehle,

RecFilterAdd()

Option bei RecFilterAdd() durch die eine Exklusive ODER-Verknüpfung durchgeführt werden kann.

RecDelete(int1, int2) : int




Datensatz löschen

int1 Dateinummer oder Deskriptor eines
 Datensatzpuffers
 Optionen
 0 Aktueller Datensatz
 (Feldpuffer des ersten
 Schlüssels)

RecFirst Erster Datensatz
 RecPrev Vorheriger Datensatz
 int2 RecNext Nächster Datensatz
 RecLast Letzter Datensatz
 RecUseId Datensatz-ID zur
 Löschung verwenden
 RecEarlyCommit Änderung trotz
 Transaktion
 durchführen

 Löschesultat

rOk Löschen erfolgreich
 rLocked Datensatz gesperrt
 rNoKey Schlüssel nicht
 vorhanden, nächsten
 Datensatz geladen
 rLastRec Schlüssel nicht
 vorhanden, letzten
 Datensatz geladen

Resultat int rNoRec Kein Datensatz 
 vorhanden
 rNoRights Benutzerrechte
 nicht ausreichend
 rDeadlock Verklemmung
 aufgetreten
 ErrDbComm Verbindungsabbruch
 zu einer
 verbundenen
 Datenbank
 aufgetreten.

Verwandte Befehle, RecInsert(),
 Siehe RecBufCreate(), RecDeleteAll(),
 Datensatzpuffer, Beispiel

Mit dieser Funktion kann ein Datensatz in der übergebenen Datei oder in dem übergebenen Datensatzpuffer (int1) gelöscht werden. Dabei wird immer über den ersten Schlüssel zugegriffen. Um einen Satz zu löschen, müssen lediglich die Felder des ersten Schlüssels definiert sein. Die Resultate des Zugriffs sind dabei analog zu RecRead() über den ersten Schlüssel mit der Option RecLock. Der Satz wird nur bei dem Ergebnis rOk gelöscht.

Kontakt

In (int2) wird angegeben welcher Datensatz gelöscht werden soll. Wird hier RecUseId angegeben, wird kein Schlüsselwert aus den Feldinhalten der Felder des ersten Schlüssels gebildet, sondern die Datensatz-ID verwendet. Die Datensatz-ID wird automatisch beim Lesen eines Datensatzes global gesetzt. Sie kann ebenfalls über die Anweisung RecInfo(..., RecSetID, ...) gesetzt werden.

Bei der Angabe von RecEarlyCommit können innerhalb einer Transaktion (siehe DtaBegin()) Änderungen am Datenbestand vorgenommen werden, ohne dass es zu Wartezuständen bei anderen Transaktionen kommt.

Die Feldpuffer werden durch den Befehl RecDelete() nicht verändert. Soll nach dem Löschen eines Datensatzes der nächste Datensatz gelesen werden, reicht ein RecRead(<Dateinummer>, <Schlüsselnummer>, 0) aus. Es wird versucht auf den Datensatz im Feldpuffer zuzugreifen, da dieser Datensatz nicht existiert, wird der nächste Datensatz gelesen und das Ergebnis rNoKey zurückgegeben.

Beispiele:

```
// Löschen über ersten SchlüsselADR.iID # 1024;tErg # RecDelete(ADR.D.Adressen, 0);tHdlBuf # RecE
```

Mögliche Laufzeitfehler:

ErrNoFile Datei nicht vorhanden