

Kontakt

Der Text ersetzt den Text, der mit der Markierung (\$ctxDocEdit->cpiSelLength) selektiert ist. Ist keine Markierung vorhanden, wird der Text an der aktuellen Cursorposition \$ctxDocEdit->cpiSelStart eingefügt.

Bei der Kombination mit der Option WinDocLoadMix werden beim Laden des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.



WinDocLoadMix kann nicht zusammen mit WinDocLoadDoc oder WinDocLoadDocX angegeben werden.

Als Rückgabewert kann neben den Fehlerkonstanten aus dem Bereich der externe Dateien der Wert ErrGeneric zurückgegeben werden, wenn ein interner Fehler aufgetreten ist. Bei der Rückgabe von ErrOk ist kein Fehler aufgetreten.



Intern wird die Funktion LoadFromMemory der Text-Control-Bibliothek aufgerufen. Dabei wird die Eigenschaft LoadSaveAttribute der Bibliothek beachtet. Die Eigenschaft kann mit \$ctxDocEdit->cpiLoadSaveAttribute gelesen und gesetzt werden. Nähere Informationen finden Sie auf der Hersteller-Seite des Moduls.

Beispiel:

```
// externes Word-Dokument laden $ctxDocEdit->WinDocLoadName(_WinStreamNameFile, _WinDocLoadDoc, _S
```

Mögliche Laufzeitfehler

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein CtxDocEdit-Objekt
ErrValueInvalid Argument (int1) oder (int2) enthält ungültige Werte oder der Wert von (alpha3) ist leer und FileName ist nicht angegeben.

obj -> WinDocPrint([alpha1[, int2[,
int3]]) : int



Inhalt des CtxDocEdit-Objektes drucken

obj Objekt
 (CtxDocEdit-Objekt)
alpha1 Seitenauswahl
 (optional)
 Deskriptor eines
int2 PrintDevice-Objektes
 (optional)
int3 Anzahl der exemplare
 (optional)

Resultat int Fehlerwert

Siehe Verwandte Befehle

Mit diesem Befehl wird der Inhalt des CtxDocEdit-Objektes gedruckt. Der Deskriptor des CtxDocEdit-Objektes wird in (obj) übergeben.

Im Parameter (alpha1) kann eine Seitenauswahl definiert werden. Ist dieser Parameter leer, wird der gesamte Inhalt des Objektes gedruckt. Es kann eine der Anweisungen 'range', 'odd' oder 'even' übergeben werden:

• range()

Dieser Eintrag definiert einen Bereich oder eine Aufzählung von Seiten die gedruckt werden soll. In Klammern wird die Aufzählung der Seitennummern, ein Bereich von Seiten oder eine Aufzählung von Seitenbereichen angegeben. Bereichsüberlappungen sind nicht erlaubt und führen zu einem Laufzeitfehler.

Wert	Resultat
'range()'	Alle Seiten
'range(1-10)'	Seiten 1-10
'range(1,3,5-9)'	Seiten 1, 3 und 5-9
'range(1-10,4-7)'	Laufzeitfehler!

• odd()

Dieser Eintrag definiert einen Bereich ungerader Seiten die gedruckt werden sollen.

Wert	Resultat
'odd()'	Alle ungeraden Seiten
'odd(1,10)'	Alle ungeraden Seiten im Bereich 1-10
'odd(1-10)'	Alle ungeraden Seiten im Bereich 1-10

• even()

Dieser Eintrag definiert einen Bereich gerader Seiten die gedruckt werden sollen.

Wert	Resultat
------	----------

'even()' Alle geraden Seiten
'even(1,10)' Alle geraden Seiten im Bereich 1-10
'even(1-10)' Alle geraden Seiten im Bereich 1-10

Im Parameter (int2) kann der Deskriptor eines PrintDevice-Objektes übergeben werden. Wird dieser Parameter nicht angegeben, erfolgt der Druck auf dem Standarddrucker.



Die Eigenschaften ZoomFactor, QualityX und QualityY vom PrintDevice haben auf diesen Befehl keine Wirkung.

Die Anzahl der zu druckenden Exemplare kann im Parameter (int3) festgelegt werden. Ist der Parameter nicht angegeben, wird die Eigenschaft Copies vom PrintDevice-Objekt ausgewertet. Ist auch diese nicht definiert, wird der Inhalt ein Mal gedruckt.

Beispiel:

```
// Seiten 1, 3 und 5 bis 7 des Dokumentes auf dem Drucker in tPrtDevice 2 Mal drucken$ctxDocEdit-
```

Folgende Fehlerwerte werden von der Funktion zurückgegeben:

_ErrGeneric Allgemeiner Fehler beim Drucken.
_ErrOutOfMemory Speicher für die Interpretation der Seitenauswahl oder den Drucker konnte nicht angefordert werden.

Mögliche Laufzeitfehler

_ErrHdlInvalid Bei (obj) handelt es sich nicht um ein CtxDocEdit-Objekt bzw. bei (int2) nicht um ein PrintDevice-Objekt.
_ErrValueInvalid Argument (alpha1) enthält ungültige Werte.

obj -> WinDocSaveBin(handle1, int2[, int3[, alpha4]]) :
int



Text aus CtxDocEdit-Objekt in binärem Objekt speichern

obj Objekt (CtxDocEdit-Objekt)

handle1 Deskriptor des binären Objekts

Modus

_WinDocSaveAscii ASCII-Text
sichern

_WinDocSaveRtf RTF-Text
sichern

_WinDocSaveHtml HTML-Format
sichern

_WinDocSaveDoc Inhalt im .doc
Format
sichern

int2 _WinDocSaveDocX Inhalt im
.docx Format
sichern

_WinDocSavePdf Inhalt im
PDF-Format
sichern

_WinDocSaveMix Text mit
Daten
mischen

_WinDocSaveOEM OEM-Text
sichern

_WinDocSaveMark Markierten
Bereich
sichern

int3 Kompressionsfaktor (optional)

alpha4 Verschlüsselungscode (optional)

Resultat int Fehlerwert

Verwandte Befehle,

Siehe WinDocSaveName(),

WinDocLoadBin(), BinOpen()

Mit diesem Befehl wird ein Text aus einem CtxDocEdit-Objekt in einem binären Objekt gespeichert. Der Deskriptor des CtxDocEdit-Objektes wird in (obj), der Deskriptor des binären Objekts in (handle1) übergeben. Das binäre Objekt (handle1) muss beim Öffnen mit _BinLock oder _BinSingleLock gesperrt werden.

Der Parameter (int2) bestimmt das zu schreibende Format. Folgende Konstanten können angegeben werden:

- **_WinDocSaveAscii**

ASCII-Text sichern

- **_WinDocSaveRtf**

RTF-Text sichern

- **WinDocSaveHtml**

HTML-Format sichern

- **WinDocSaveDoc**

Inhalt im .doc Format sichern

- **WinDocSaveDocX**

Inhalt im .docx Format sichern

- **WinDocSavePdf**

Inhalt im PDF-Format sichern

- **WinDocSaveOEM**

OEM-Text sichern

- **WinDocSaveMix**

Text mit Daten mischen

- **WinDocSaveMark**

Markierten Bereich sichern

Die Parameter zum Zielformat können mit WinDocSaveMark kombiniert werden, um einen markierten Teil innerhalb des Textes zu sichern.

Bei der Kombination mit der Option WinDocSaveMix werden beim Speichern des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.



WinDocSaveMix kann nicht zusammen mit WinDocSaveDoc, WinDocSaveDocX oder WinDocSavePdf angegeben werden.

Als Rückgabewert kann neben den Fehlerkonstanten aus dem Bereich der binären Objekte der Wert ErrGeneric zurückgegeben werden, wenn ein interner Fehler aufgetreten ist. Bei der Rückgabe von ErrOk ist kein Fehler aufgetreten.



Intern wird die Funktion SaveToMemory der Text-Control-Bibliothek aufgerufen. Dabei wird die Eigenschaft LoadSaveAttribute der Bibliothek beachtet. Die Eigenschaft kann mit `$ctxDocEdit->cpiLoadSaveAttribute` gelesen und gesetzt werden. Nähere Informationen finden Sie auf der Hersteller-Seite des Moduls.

Beispiel:

```
// Word-Dokument als binäres Objekt speichern $ctxDocEdit->WinDocSaveBin(tBinFileHdl, _WinDocSaveBin)
```

Mögliche Laufzeitfehler

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein CtxDocEdit-Objekt bzw. bei (handle1) nicht um ein binäres Objekt.

ErrValueInvalid Argument (int2) enthält ungültige Werte.

obj -> WinDocSaveName(int1, int2[,
alpha3]) : int

Text aus CtxDocEdit-Objekt speichern
obj Objekt (CtxDocEdit-Objekt)

Ziel des Textes

WinStreamNameText Text in
einen
internen
Text
schreiben

int1 WinStreamNameFile Text in
eine
externe
Datei
schreiben

Modus für interne und externe
Texte

WinDocSaveAscii ASCII-Text
sichern

WinDocSaveRtf RTF-Text
sichern

WinDocSaveHtml HTML-Format
sichern

WinDocSaveOEM OEM-Text
sichern

WinDocSaveMix Text mit
Daten
mischen

WinDocSaveMark Markierten
Bereich
sichern

int2 zusätzlicher Modus für externe
Texte

WinDocSaveDoc Inhalt im .doc
Format
sichern

WinDocSaveDocX Inhalt im
.docx Format
sichern

WinDocSavePdf Inhalt im
PDF-Format
sichern

WinDocSaveAuto Textformat
anhand der
Dateiendung
automatisch
erkennen

alpha3 Zielname



Resultat int Fehlerwert

Siehe Verwandte Befehle,
WinDocSaveBin(),
WinDocLoadName()

Mit diesem Befehl wird ein Text aus einem CtxDocEdit-Objekt gespeichert. Der Deskriptor des Objektes wird in (obj) übergeben.

In (int1) wird das Ziel des Textes definiert. Der Name des Ziels wird in (alpha3) angegeben.

Folgende Ziele können angegeben werden:

- **_WinStreamNameText**

Der Text wird in einem internen Text gespeichert. Der Name des Textes wird in (alpha3) übergeben.

- **_WinStreamNameFile**

Der Text wird in einer externen Datei gespeichert. Der Name des Textes kann in (alpha3) übergeben werden. Ist er nicht angegeben, wird er der Eigenschaft FileName entnommen.

Der Parameter (int2) bestimmt das zu schreibende Format. Folgende Konstanten können übergeben werden:

Modus für interne und externe Texte

- **_WinDocSaveAscii**

ASCII-Text sichern

- **_WinDocSaveRtf**

RTF-Text sichern

- **_WinDocSaveHtml**

HTML-Format sichern

- **_WinDocSaveOEM**

OEM-Text sichern

- **_WinDocSaveMix**

Text mit Daten mischen

- **_WinDocSaveMark**

Markierten Bereich sichern

zusätzlicher Modus für externe Texte

- **_WinDocSaveDoc**

Inhalt im .doc Format sichern

- **WinDocSaveDocX**

Inhalt im .docx Format sichern

- **WinDocSavePdf**

Inhalt im PDF-Format sichern

- **WinDocSaveAuto**

Textformat anhand der Dateieindung automatisch erkennen

Die Parameter zum Zielformat können mit WinDocSaveMark kombiniert werden, um einen markierten Teil innerhalb des Textes zu sichern.

Bei der Kombination mit der Option WinDocSaveMix werden beim Speichern des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.



WinDocSaveMix kann nicht zusammen mit WinDocSaveDoc, WinDocSaveDocX oder WinDocSavePdf angegeben werden.

Als Rückgabewert kann neben den Fehlerkonstanten aus dem Bereich der externe Dateien der Wert ErrGeneric zurückgegeben werden, wenn ein interner Fehler aufgetreten ist. Bei der Rückgabe von ErrOk ist kein Fehler aufgetreten.



Intern wird die Funktion SaveToMemory der Text-Control-Bibliothek aufgerufen. Dabei wird die Eigenschaft LoadSaveAttribute der Bibliothek beachtet. Die Eigenschaft kann mit `$ctxDocEdit->cpiloadSaveAttribute` gelesen und gesetzt werden. Nähere Informationen finden Sie auf der Hersteller-Seite des Moduls.

Beispiel:

```
// Dokument extern als PDF speichern$ctxDocEdit->WinDocSaveName(_WinStreamNameFile, _WinDocSavePo
```

Mögliche Laufzeitfehler

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein CtxDocEdit-Objekt
ErrValueInvalid Argument (int1) oder (int2) enthält ungültige Werte oder der Wert von (alpha3) ist leer und FileName ist nicht angegeben.



obj -> WinDocUserDictAddName(alpha1) : int
CtxDocEdit-Objekt ein benutzerdefiniertes Wörterbuch hinzufügen
obj Objekt (CtxDocEdit-Objekt)
alpha1 Pfad und Name des Wörterbuchs

Fehlercode

ErrOk

Wörterbuch hinzugefügt

Resultat int ErrGeneric CtxDocEdit-Erweiterung nicht
installiert

ErrFsiNoFile Wörterbuch nicht vorhanden

Siehe Verwandte Befehle, WinDocUserDictRemoveName(),
Blog

Fügt dem CtxDocEdit-Objekt (obj) ein benutzerdefiniertes Wörterbuch hinzu. Das Wörterbuch muss im .tlx-Format gespeichert sein (siehe Aufbau einer tlx-Datei).

In (alpha1) steht der Pfad zu der .tlx-Datei. Sofern die .tlx-Datei im Verzeichnis der wspell.ocx liegt, genügt der Dateiname. Die wspell.ocx liegt standardmäßig im Installationspfad von CONZEPT 16 unter Common\CtxDocEdit\Spell\.

Beim Hinzufügen wird geprüft, ob die Datei existiert. Wenn die Datei nicht existiert kann sie auch nicht dem CtxDocEdit-Objekt hinzugefügt werden. In dem Fall wird ErrFsiNoFile zurückgegeben. Ein ungültiges Dateiformat wird nicht erkannt.

Die CtxDocEdit-Erweiterung muss installiert sein (siehe InstallCtxDocEdit).

Die Wörterbücher werden intern kommasepariert in einer Eigenschaft gespeichert, die Eigenschaft hat eine Maximallänge von 1023 Zeichen, wird diese überschritten wird der Laufzeitfehler ErrStringOverflow ausgelöst.

Folgende Fehlerwerte werden von der Funktion zurückgegeben:

ErrOk Das Wörterbuch wurde dem CtxDocEdit-Objekt (obj) hinzugefügt.

ErrGeneric Die CtxDocEdit-Erweiterung (siehe InstallCtxDocEdit) ist nicht
installiert.

ErrFsiNoFile Die Datei (alpha1) existiert nicht.

Beispiele:

```
// Datei liegt in C:\MyDictionaries$DocEdit->WinDocUserDictAddName('C:\MyDictionaries\Example.tlx')
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der Deskriptor des CtxDocEdit-Objektes (obj) ist ungültig.

ErrStringOverflow Dem CtxDocEdit-Objekt (obj) wurden zu viele Wörterbücher
hinzugefügt.



obj -> WinDocUserDictRemoveName(alpha1)

Benutzerdefiniertes Wörterbuch aus CtxDocEdit-Objekt entfernen

obj Objekt
 (CtxDocEdit-Objekt)

alpha1 Pfad und Name des
 Wörterbuchs

Verwandte Befehle,

Siehe WinDocUserDictAddName(),
Blog

Entfernt ein benutzerdefiniertes Wörterbuch aus dem CtxDocEdit-Objekt.

In (alpha1) steht der Pfad zu der .tlx-Datei. Sofern die .tlx-Datei im Verzeichnis der wspell.ocx liegt, genügt der Dateiname. Die wspell.ocx liegt standardmäßig im Installationspfad von CONZEPT 16 unter Common\CtxDocEdit\Spell\.

Die CtxDocEdit-Erweiterung muss installiert sein (siehe InstallCtxDocEdit).

Die Funktion besitzt keinen Rückgabewert.

Beispiele:

```
// Datei liegt in C:\MyDictionaries$DocEdit->WinDocUserDictRemoveName('C:\MyDictionaries\Example.
```

Mögliche Laufzeitfehler:

__ErrHdllInvalid Der Deskriptor des CtxDocEdit-Objektes (obj) ist ungültig.

Konstanten für CtxDocEdit-Befehle
Konstanten für CtxDocEdit-Befehle
Siehe CtxDocEdit-Befehle

- WinDocLoadAscii
- WinDocLoadAuto
- WinDocLoadDoc
- WinDocLoadDocX
- WinDocLoadHtml
- WinDocLoadInsert
- WinDocLoadMix
- WinDocLoadOEM
- WinDocLoadRtf
- WinDocSaveAscii
- WinDocSaveAuto
- WinDocSaveDoc
- WinDocSaveDocX
- WinDocSaveHtml
- WinDocSaveMark
- WinDocSaveMix
- WinDocSaveOEM
- WinDocSavePdf
- WinDocSaveRtf
- WinStreamNameFile
- WinStreamNameText

WinDocLoadAscii
Text im ASCII-Format laden
Wert 2 / 0x0002

Siehe WinDocLoadName(),
WinDocLoadBin()

Der Text wird im ASCII-Format geladen.

WinDocLoadAuto
Format automatisch bestimmen
Wert 0 / 0x0000

Siehe WinDocLoadName(),
WinDocLoadBin()

Das Format des zu ladenden Textes wird automatisch anhand der Dateiendung bestimmt. Dies funktioniert nur beim Laden von externen Texten.

WinDocLoadDoc
Text im .doc-Format laden
Wert 512 / 0x0200
Siehe WinDocLoadName(),
WinDocLoadBin()
Der Text wird im .doc-Format geladen.

WinDocLoadDocX

Text im .docx-Format laden

Wert 1.024 / 0x0400

Siehe WinDocLoadName(),

WinDocLoadBin()

Der Text wird im .docx-Format geladen.

WinDocLoadHtml

Text im HTML-Format laden

Wert 256 / 0x0100

Siehe WinDocLoadName(),

WinDocLoadBin()

Der Text wird im HTML-Format geladen.

_WinDocLoadInsert

Text einfügen

Wert 16 / 0x0010

Siehe WinDocLoadName(),

WinDocLoadBin()

Der zu ladende Text wird in einen bestehenden Text eingefügt.

_WinDocLoadMix

Text mit Daten beim Laden mischen

Wert 8 / 0x0008

Siehe WinDocLoadName(),

WinDocLoadBin()

Beim Laden des Textes werden die enthaltenen Platzhalter durch den entsprechenden Inhalt ersetzt. Die Platzhalter sind mit Markierungszeichen geklammert, die in der Eigenschaft DocMixMarker des _App-Objektes definiert werden kann.

Weitere Hinweise befinden sich im Abschnitt Text und Daten mischen.

WinDocLoadOEM

Text im OEM-Format laden

Wert 4 / 0x0004

Siehe WinDocLoadName(),

WinDocLoadBin()

Der Text wird im OEM-Format geladen.

WinDocLoadRtf
Text im RTF-Format laden
Wert 1 / 0x0001
Siehe WinDocLoadName(),
WinDocLoadBin()
Der Text wird im RTF-Format geladen.

_WinDocSaveAscii
Speichern im ASCII-Format
Wert 2 / 0x0002

Siehe WinDocSaveName(),
WinDocSaveBin()

Der Text wird im ASCII-Format gespeichert. Die Formatierungen gehen dabei verloren.

_WinDocSaveAuto

Format automatisch bestimmen

Wert 0 / 0x0000

Siehe WinDocSaveName()

Das Format wird automatisch anhand der Dateiendung bestimmt. Dazu muss die Eigenschaft FileName gesetzt sein.

WinDocSaveDoc

Speichern im .doc-Format

Wert 512 / 0x0200

Siehe WinDocSaveName(),
WinDocSaveBin()

Der Text wird im .doc-Format gespeichert.

_WinDocSaveDocX

Speichern im .docx-Format

Wert 1.024 / 0x0400

Siehe WinDocSaveName(),
WinDocSaveBin()

Der Text wird im .docx-Format gespeichert.

_WinDocSaveHtml
Speichern im HTML-Format
Wert 256 / 0x0100

Siehe WinDocSaveName(),
WinDocSaveBin()

Der Text wird im HTML-Format gespeichert.

_WinDocSaveMark

Markierung speichern

Wert 16 / 0x0010

Siehe WinDocSaveName(),

WinDocSaveBin()

Der markierte Bereich wird gespeichert.

_WinDocSaveMix

Text mit Daten beim Laden mischen

Wert 8 / 0x0008

Siehe WinDocSaveName(),
WinDocSaveBin()

Beim Speichern des Textes werden die enthaltenen Platzhalter durch den entsprechenden Inhalt ersetzt. Die Platzhalter sind mit Markierungszeichen geklammert, die in der Eigenschaft DocMixMarker des _App-Objektes definiert werden können.

Weitere Hinweise befinden sich im Abschnitt Text und Daten mischen.

_WinDocSaveOEM
Speichern im OEM-Format
Wert 4 / 0x0004

Siehe WinDocSaveName(),
WinDocSaveBin()

Der Text wird im OEM-Format gespeichert. Die Formatierungen gehen dabei verloren.

_WinDocSavePdf

Speichern im PDF-Format

Wert 2.048 / 0x0800

Siehe WinDocSaveName(),
WinDocSaveBin()

Der Text wird im PDF-Format gespeichert.

_WinDocSaveRtf

Speichern im RTF-Format

Wert 1 / 0x0001

Siehe WinDocSaveName(),
WinDocSaveBin()

Der Text wird im RTF-Format gespeichert.

_WinStreamNameFile

Quelle des Textes

Wert 5

StreamSource,

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfSave(),

WinRtfSaveName()

Wird die Eigenschaft StreamSource auf den Wert _WinStreamNameFile gesetzt, wird der Inhalt der in der Eigenschaft FileName angegebenen externen Datei dargestellt.

Die Konstante wird ebenfalls verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad(), WinRtfLoadName(), WinRtfSave(), WinRtfSaveName(), WinDocLoadName() und WinDocSaveName() anzugeben.

_WinStreamNameText

Quelle des Textes

Wert 4

StreamSource,

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfSave(),

WinRtfSaveName()

Wird die Eigenschaft StreamSource auf den Wert _WinStreamNameText gesetzt, wird der in der Eigenschaft FileName angegebene interne Text dargestellt.

Die Konstante wird ebenfalls verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad(), WinRtfLoadName(), WinRtfSave(), WinRtfSaveName(), WinDocLoadName() und WinDocSaveName() anzugeben.

DataList-Befehle

Befehle zum Bearbeiten eines DataList-Objekts

Verwandte

Befehle,

Siehe DataList,

Befehlsgruppen,

Befehlsliste

Befehle

- WinLstCellGet
- WinLstCellSet
- WinLstDatLineAdd
- WinLstDatLineInfo
- WinLstDatLineRemove
- WinLstEdit

Konstanten

- _WinLstDatInfoCount
- _WinLstDatLineAll
- _WinLstDatLineCurrent
- _WinLstDatLineLast
- _WinLstDatLineSelected
- _WinLstDatModeDefault
- _WinLstDatModeSortInfo

obj ->

WinLstDatLineInfo(int1)

: int

Zeilen-Informationen

obj Objekt

Option

int1 WinLstDatInfoCount Anzahl
 der
 Zeilen

Resultat int Zeilen-Information


Siehe DataList-Befehle,
 Verwandte Befehle

Wird WinLstDatInfoCount übergeben, liefert dieser Befehl die Anzahl der Zeilen zurück, die sich momentan in dem Objekt DataList befinden.

Mögliche Laufzeitfehler:

ErrHdlInvalid DataList (obj) ungültig

obj ->

WinLstDatLineAdd(var1, ) : int

Zeile hinzufügen

obj Objekt

var1 Daten

Zeile (optional)

WinLstDatLineCurrent eine Zeile
wird
hinter der
aktuellen
Zeile
int2 eingefügt
WinLstDatLineLast eine Zeile
wird am
Ende
angefügt
(default)

Resultat int Zeilennummer

WinLstDatLineRemove(),

Siehe DataList-Befehle, Verwandte
Befehle

Mit diesem Befehl werden dem Objekt DataList Zeilen hinzugefügt. Mit (var1) wird die erste Zelle der Zeile direkt gesetzt. (var1) kann von verschiedenem Typ sein, muss aber mit der Eigenschaft ClmType der 1. Spalte übereinstimmen. Als Zeile (int2) kann entweder die Zeilennummer direkt angegeben, oder eine der beiden definierten Konstanten verwendet werden.

Als Resultat wird die neue Zeilennummer zurückgeliefert.

Beispiel:

```
WinOpen('FrmDataList', _WinOpenDialog);tDataList # $DataList;for    tLine # 1;loop    Inc(tLine);unt
```

Mögliche Laufzeitfehler:

ErrHdlInvalid DataList (obj) ungültig

obj ->

WinLstDatLineRemove

: logic

Zeile löschen

obj Objekt

Zeile

WinLstDatLineCurrent Aktuelle
Zeile wird
gelöscht

WinLstDatLineLast Letzte
Zeile wird
gelöscht

int1 WinLstDatLineAll Alle Zeilen
werden
gelöscht

WinLstDatLineSelected Selektierte
Zeilen
werden
gelöscht

Resultat logic Funktion erfolgreich
durchgeführt

Siehe WinLstDatLineAdd(),
DataList-Befehle, Verwandte
Befehle

Mit diesem Befehl kann eine oder mehrere Zeilen des Objekts DataList gelöscht werden.

Sollen mit der Option WinLstDatLineSelected mehrere Zeilen gelöscht werden, muss beim DataList-Objekt die Mehrfachauswahl (MultiSelect) aktiviert sein.

Wurde die Funktion erfolgreich durchgeführt ist das Ergebnis true andernfalls false.


Beispiel

```
// DataList-Objekt leerentHdlData->WinLstDatLineRemove(_WinLstDatLineAll);// Selektierte Zeilen l
```

Mögliche Laufzeitfehler:

ErrHdlInvalid DataList (obj) ist ungültig

obj ->

WinLstCellGet(var1[,
int2[, int3[, int4]]]) : 

logic

Spalte lesen

obj Objekt

var1	Daten
------	-------

int2 Spalte (optional)

Zeile (optional)

Nummer
der zu
lesenden
Zeile

int3

WinLstDatLineCurrent Aktuelle
Zeile wird
gelesen

<u>WinLstDatLineLast</u>	Letzte Zeile wird gelesen (default)
--------------------------	--

Modus (optional)

<u>WinLstDatModeDefault</u>	Anzeigewerte ermitteln (default)
-----------------------------	--

int4

WinLstDatModeSortInfo Sortiertwerte ermitteln

Resultat logic Funktion erfolgreich durchgeführt

Verwandte Befehle,

Siehe [WinLstCellSet\(\)](#), [DataList-Befehle](#),
[Blog](#)

Mit diesem Befehl wird der Inhalt oder der Sortierwert einer Spalte des Objekts DataList gelesen.

In (var1) wird der Inhalt oder der Sortierwert der angegebene Spalte übertragen. Die Variable (var1) kann von verschiedenem Typ sein, muss aber mit der Eigenschaft ClmType für den Inhalt bzw. mit der Eigenschaft ClmTypeSort für den Sortierwert der angegebenen Spalte übereinstimmen.

In (int2) kann die Position der Spalte angegeben werden. Die Position lässt sich ausgehend vom Deskriptor der Spalte über WinInfo() mit der Option WinItem ermitteln.

Bei der Zeile (int3) kann entweder die Zeile direkt, oder eine der beiden Konstanten angegeben werden. In Verbindung mit den Ereignissen EvtKeyItem, EvtMouseItem und EvtLstDataInit kann in (int3) der Parameter aID angegeben werden, da in dieser Variable die Zeilennummer steht.

Im Argument (int4) kann der Abfragemodus angegeben werden.

Kontakt

Wurde die Funktion erfolgreich durchgeführt ist das Ergebnis true andernfalls false.


Beispiele:

```
// Erste Spalte der aktuellen Zeile lesentHdlDataList->WinLstCellGet(tCellContent); // Zweite Spalte
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid DataList (obj) ungültig

obj ->

WinLstCellSet(var1[,  int2[, int3[, int4]]) :

logic

Spalte setzen

obj Objekt

var1 Daten

int2 Spalte (optional)

Zeile (optional)

WinLstDatLineCurrent Aktuelle
Zeile wird
geändert

int3 WinLstDatLineLast Letzte
Zeile wird
geändert
(default)

Modus (optional)

WinLstDatModeDefault Anzeigewerte
definieren
(default)

int4 WinLstDatModeSortInfo Sortiertwerte
definieren

Resultat logic Funktion erfolgreich
durchgeführt

Verwandte Befehle,

Siehe WinLstCellGet(), DataList-Befehle,
Blog

Mit diesem Befehl wird der Inhalt oder der Sortierwert einer Spalte des Objekts DataList gesetzt. Mit (var1) wird die angegebene Spalte der Zeile gesetzt. (var1) kann von verschiedenem Typ sein, muss aber mit der Eigenschaft ClmType für den Inhalt bzw. mit der Eigenschaft ClmTypeSort für den Sortierwert der angegebenen Spalte übereinstimmen.

In (int2) kann die Position der Spalte angegeben werden. Die Position lässt sich ausgehend vom Deskriptor der Spalte über WinInfo() mit der Option WinItem ermitteln.

Bei der Zeile (int3) kann entweder die Zeile direkt, oder eine der beiden Konstanten angegeben werden.

Im Argument (int4) kann der Einfügemodus angegeben werden. Ist die Eigenschaft ClmTypeSort bei einer Spalte gesetzt, sollten für jede Zeile der Spalte auch Sortierwerte definiert werden, da das Ergebnis der Sortierung sonst nicht definiert ist.

Wurde die Funktion erfolgreich durchgeführt ist das Ergebnis true andernfalls false.

Beispiel:

```
// Datentyp für SortiervergleichClm->wpClmTypeSort # _TypeBigInt;...// Sortierungswert für Spalte
```

Mögliche Laufzeitfehler:

ErrHdlInvalid DataList (obj) ungültig

Konstanten für DataList-Befehle
Konstanten für DataList-Befehle
Siehe DataList-Befehle

Konstanten

- WinLstDatInfoCount
- WinLstDatLineAll
- WinLstDatLineCurrent
- WinLstDatLineLast
- WinLstDatLineSelected
- WinLstDatModeDefault
- WinLstDatModeSortInfo

WinLstDatInfoCount

Anzahl Zeilen

Wert 1

Siehe Verwandte
Befehle

Option bei WinLstDatLineInfo(). Es wird die Anzahl der Zeilen, die sich in der Liste befinden, zurückgegeben.

WinLstDatLineAll

Alle Zeilen

Wert -3

Siehe Verwandte
Befehle

Option bei WinLstDatLineRemove(). Es werden alle Zeilen gelöscht.

_WinLstDatLineCurrent
Bedeutung je nach Befehl
Wert -1

Siehe Verwandte
Befehle

Option bei folgenden Befehlen:

<u>WinLstDatLineAdd()</u>	Zeile wird hinter der aktuellen Zeile angefügt
<u>WinLstDatLineRemove()</u>	aktuelle Zeile wird gelöscht
<u>WinLstCellSet()</u>	aktuelle Zeile wird geändert
<u>WinLstCellGet()</u>	aktuelle Zeile wird gelesen

_WinLstDatLineLast
Bedeutung je nach Befehl
Wert -2

Siehe Verwandte
Befehle

Option bei folgenden Befehlen:

<u>WinLstDatLineAdd()</u>	Zeile wird am Ende angefügt
<u>WinLstDatLineRemove()</u>	letzte Zeile wird gelöscht
<u>WinLstCellSet()</u>	letzte Zeile wird geändert
<u>WinLstCellGet()</u>	letzte Zeile wird gelesen

_WinLstDatLineSelected
Selektierte Zeilen löschen
Wert -4

Siehe Verwandte
Befehle

Option bei WinLstDatLineRemove() - Sofern bei dem DataList-Objekt die Eigenschaft MultiSelect gesetzt ist, werden alle selektierten Datensätze des DataList-Objektes entfernt.

WinLstDatModeDefault
Daten angeben / abfragen
Wert 0

Verwandte

Siehe Befehle,
WinLstCellGet(),
WinLstCellSet()

Modus bei WinLstCellGet() und WinLstCellSet(), mit dem die Daten einer Zelle ermittelt und gesetzt werden können. Dies ist der Standardmodus.

WinLstDatModeSortInfo
Sortierwert angeben / abfragen
Wert 1

Verwandte

Siehe Befehle,
WinLstCellGet(),
WinLstCellSet()

Modus bei WinLstCellGet() und WinLstCellSet(), mit dem der Sortierwert einer Zelle ermittelt und gesetzt werden kann.

Dialog-Befehle

Befehle zum Laden/Anzeigen von Dialog-Objekten

Verwandte

Siehe Befehle,
Befehlsgruppen,
Befehlsliste

Befehle

- WinAdd
- WinAddByName
- WinClose
- WinDialog
- WinDialogBox
- WinDialogResult
- WinDialogRun
- WinOpen
- WinSave
- WinUrmDialog

Konstanten

- WinAddHidden
- WinDialogAlwaysOnTop
- WinDialogApp
- WinDialogAsync
- WinDialogBoxDefault
- WinDialogBoxUseFont
- WinDialogBoxUseFontButton
- WinDialogBoxUseTextButton
- WinDialogCenter
- WinDialogCenterScreen
- WinDialogCenterScreenX
- WinDialogCenterScreenY
- WinDialogCenterX
- WinDialogCenterY
- WinDialogCreateHidden
- WinDialogMaximized
- WinDialogMinimized
- WinDialogNoActivate
- WinDialogOK
- WinDialogOKCancel
- WinDialogUtf8
- WinDialogYesNo
- WinDialogYesNoCancel
- WinIcoApplication
- WinIcoError
- WinIcoExternFile
- WinIcoInformation
- WinIcoQuestion
- WinIcoWarning
- WinOpenDialog

- WinOpenEventsOff
- WinOpenLock
- WinOpenUnicode
- WinUpdScrollPos
- WinUrmFlagHelpTipOnSysProps



obj -> WinAdd(handle1[, int2[, handle3]]) : int
Oberflächenobjekt in ein anderes Oberflächenobjekt laden

obj Elternojekt

handle1 einzufügendes Oberflächenobjekt


Optionen (optional)

WinAddHidden MDI-Fenster
unsichtbar
laden

int2 WinDialogMaximized MDI-Fenster
maximiert
laden

WinDialogMinimized MDI-Fenster
minimiert
laden

handle3 Deskriptor des nachfolgenden
Objektes (optional)

Resultat int Laderesultat 
ErrOk Laden erfolgreich

Siehe Verwandte Befehle,
WinAddByName(), WinCreate(),
WinRemove(), WinDestroy(),
Ereignisabläufe MdiFrame

Der Befehl fügt dem angegebenen Elternojekt (obj) das Oberflächenobjekt (handle1) hinzu. Bei dem Oberflächenobjekt muss es sich um einen MdiFrame oder um ein mit WinCreate() angelegtes Objekt handeln. Das Objekt (handle1) kann bereits weitere mit WinCreate() und WinAdd() untergeordnete Oberflächenobjekte enthalten.

Ist das Objekt (handle1) ein MdiFrame, kann über den optionalen Parameter (int2) mit der Konstanten WinAddHidden angegeben werden, dass das MdiFrame-Objekt unsichtbar geladen werden soll. In diesem Fall muss das MdiFrame-Objekt später mit dem Befehl WinUpdate() mit dem Parameter WinUpdOn gezeichnet werden. Mit den Optionen WinDialogMaximized bzw. WinDialogMinimized kann der MdiFrame maximiert oder minimiert werden.

Sind dem MDI-Fenster über die Eigenschaft DbRecBuf eigene Feldpuffer zugeordnet worden, stehen diese nach diesem Befehl zur Verfügung und können initialisiert werden.

Wurde das Oberflächenobjekt (handle1) mit WinCreate() erstellt, kann das nachfolgende Objekt im Argument (handle3) angegeben und somit die Objektreihenfolge definiert werden.



Das Objekt (handle1) darf zuvor nicht bereits mit WinAdd() zu einem anderen Objekt hinzugefügt werden, außer es wurde anschließend mit WinRemove() wieder entfernt.

Bei dem Oberflächenobjekt (handle1) kann es sich nicht um ein Frame-, AppFrame- oder TrayFrame-Objekt handeln. Um diese zu starten, muss, auch für dynamisch erstellte, weiterhin der Befehl WinDialogRun() verwendet werden.

Wird das Elternobjekt (obj) bereits angezeigt, erfolgt auch die Darstellung des hinzugefügten Oberflächenobjektes (handle1) direkt nach der Ausführung des Befehls.

Als Resultat kann der Fehlerwert ErrType zurückgegeben werden, wenn das hinzuzufügende Objekt (handle1) nicht in das Elternobjekt (obj) eingefügt werden kann. Das Resultat ist ErrExists, wenn das hinzuzufügende Objekt (handle1) bereits einem Objekt hinzugefügt wurde. Konnte ein MdiFrame nicht zu einem AppFrame hinzugefügt werden, ist das Resultat ErrOutOfMemory. Das Resultat ist ErrUnavailable, wenn eine Spalte vor einer bestehenden Spalte eingefügt wird und die Liste bereits Inhalt besitzt. Ist kein Fehler aufgetreten, wird ErrOk zurückgegeben.

Hinweise für Spalten-Objekte

Dynamische Spalten-Objekte können auch einer nicht dynamisch erstellten DataList oder RecList hinzugefügt werden. Diese darf auch bereits nicht dynamisch erstellte Spalten enthalten.

Ein Spalten-Objekt besitzt eine Anzeigeposition und eine Indexposition. Die Anzeigeposition definiert, wo die Spalte innerhalb der Liste angezeigt wird und kann sich z. B. durch Benutzerinteraktionen mit der Spalte (z. B. Verschieben der Spalte durch den Anwender) ändern. Die Anzeigeposition kann durch die Eigenschaft ClmOrder gesetzt oder auch abgefragt werden.

Die Indexposition definiert, welche Position die Zelle im Datensatz (Zeile) besitzt. Diese Position wird den Befehlen WinLstCellSet() und WinLstCellGet() übergeben, wenn die Daten einer Zelle gesetzt oder abgefragt werden sollen. Die Indexposition kann mit dem Befehl WinInfo() und der Option WinItem ermittelt werden. Bei WinAdd() unter Angabe einer nachfolgenden Spalte erhält die neu hinzugefügte Spalte die die Indexposition der nachfolgenden Spalte. Die Anzeigeposition wird über die Eigenschaft ClmOrder definiert.



Die Angabe eines nachfolgenden Spalten-Objektes ist nicht zulässig, wenn die Liste bereits einen Inhalt besitzt (Rückgabewert ErrUnavailable). Es können jedoch Spalten am Ende eingefügt werden (WinAdd() ohne nachfolgendes Objekt).

Nachdem ein Spalten-Objekt einer DataList hinzugefügt wurde, hat diese zunächst keinen Inhalt. Mit dem Befehl WinLstCellSet() kann der Spalteninhalt entsprechend des durch ClmType definierten Spalten-Typs gesetzt werden. Der Befehl WinLstCellGet() liefert den Wert false, wenn die Zelle der Spalte noch keinen Inhalt besitzt.

Hinweise für GroupColumn-Objekte

Dynamische GroupColumn-Objekte können auch einem nicht dynamisch erstellten RecView- oder einem nicht dynamisch erstellten, übergeordneten GroupColumn-Objekt hinzugefügt werden.

Ein GroupColumn-Objekt besitzt eine Anzeige- und eine Indexposition. Die Anzeigeposition definiert, wo das GroupColumn-Objekt im RecView bzw. im

übergeordneten GroupColumn-Objekt angezeigt wird. Die Anzeigeposition kann durch die Eigenschaft VisibleOrder gesetzt und abgefragt werden.

Die Indexposition definiert eine eindeutige fortlaufende Nummer für die Eigenschaft SelectorItem bzw. SelectorSubItem.

Die Angabe eines nachfolgenden GroupColumn-Objektes ist nicht zulässig, wenn das RecView bereits einen Inhalt besitzt. Es können jedoch GroupColumn-Objekte am Ende eingefügt werden (WinAdd() ohne nachfolgendes Objekt).

Wird das RecView-Objekt angezeigt, während WinAdd() für ein GroupColumn-Objekt durchgeführt wird, dann hat dies zur Folge, dass das Ereignis EvtLstGroupInit aufgerufen wird.

Hinweise zum PopupList-Objekt

Das Objekt kann den Eingabeobjekten (Edit, IntEdit, ...) hinzugefügt werden. Dem PopupList-Objekt kann wiederum ein DataListPopup, RecListPopup- oder StoListPopup-Objekt hinzugefügt werden.

Einem GroupColumn-Objekt kann ein weiteres GroupColumn-Objekt untergeordnet werden. Dies ist jedoch nur zulässig, wenn das übergeordnete Objekt nicht bereits einem GroupColumn-Objekt untergeordnet ist und das unterzuordnende GroupColumn-Objekt seinerseits keine untergeordneten GroupColumn-Objekte enthält.

Beispiele:

```
// MDI-Frame 'Addresses' laden tMdiFrame # WinOpen('Addresses');// Wenn hinzufügen des MDI-Fensters
```


Mögliche Laufzeitfehler:

<u>_ErrHdlInvalid</u>	Elternobjekt (obj) oder hinzuzufügendes Oberflächenobjekt (handle1) ungültig oder das Oberflächenobjekt (handle1) ist nicht dynamisch erstellt worden.
<u>_ErrMemExhausted</u>	Fenstererstellung ist fehlgeschlagen.
<u>_ErrIllegalOp</u>	Das nachfolgende Objekt (handle3) ist angegeben, jedoch kein Oberflächenobjekt oder kein Kindobjekt von dem angegebenen Elternobjekt (obj). Das hinzuzufügende Objekt (handle1) ist ein <u>Frame</u> , <u>MdiFrame</u> oder <u>AppFrame</u> und ein nachfolgendes Objekt (handle3) ist angegeben.



obj -> WinAddByName(alpha1[, int2]) : int
MDI-Fenster über Name in Applikations-Fenster laden

obj Applikations-Fenster
alpha1 MDI-Fenstername
Optionen (optional)
 WinAddHidden MDI-Fenster
 unsichtbar
 laden
int2 WinDialogMaximized MDI-Fenster
 maximiert
 laden
 WinDialogMinimized MDI-Fenster
 minimiert
 laden

Resultat int MDI-Fenster 

Siehe Verwandte Befehle, WinAdd(),
Ereignisabläufe MdiFrame

Der Befehl lädt das MdiFrame-Objekt (alpha1) in den Ausgabebereich des AppFrame-Objekts (obj). Über den optionalen Parameter (int2) kann über die Konstante WinAddHidden angegeben werden, ob das MdiFrame-Objekt unsichtbar geladen werden soll. In diesem Fall muss das MdiFrame-Objekt später mit dem Befehl WinUpdate() mit dem Parameter WinUpdOn gezeichnet werden.

Je nach verwendetem Betriebssystem stehen unterschiedlich viele Ressourcen (Benutzer- und GDI-Objekte) zur Darstellung des Fensters zur Verfügung. Ab dem Betriebssystem Windows 2000 wird nach dem Laden des Fensters die verbleibenden Ressourcen überprüft. Stehen weniger als 10% zur Verfügung, wird der Dialog nicht geladen. Die Anweisung gibt 0 als Deskriptor zurück und der globale Fehlerwert wird auf ErrOutOfMemory gesetzt. Die zur Verfügung stehenden Benutzer- und GDI-Objekte können über den Info-Dialog und die Eigenschaften ObjectsUserLimit und ObjectsGDILimit ermittelt werden.

Sind dem MDI-Fenster über die Eigenschaft DbRecBuf eigene Feldpuffer zugeordnet worden, stehen diese nach diesem Befehl zur Verfügung und können initialisiert werden.

Als Rückgabewert wird der Deskriptor des MDI-Fensters zurückgegeben. Konnte das MdiFrame-Objekt nicht erfolgreich zum AppFrame-Objekt hinzugefügt werden, ist das Resultat 0.

Beispiel:

```
// MDI-Fenster 'MdiFrame' in Applikations-Fenster $MdiApp hinzufügenMdi # $MdiApp->WinAddByName(
```

Mögliche Laufzeitfehler:

ErrStringOverflow MDI-Fenstername (alpha1) länger als 40 Zeichen

obj ->

WinClose() :



logic

Fenster entladen

obj Fenster

Resultat logic Entladeerfolg

Verwandte

Siehe Befehle,

WinOpen(),

WinDialog()

Mit diesem Befehl wird ein Fenster, das mit WinOpen() oder WinDialog() geladen wurde, geschlossen. Als (obj) wird der Deskriptor des zu schließenden Fensters übergeben.

Ein mit WinOpen() geöffnetes und mit WinDialogRun() aufgerufenes Fenster kann entweder durch den Benutzer (zum Beispiel durch Drücken der Schließen-Schaltfläche) oder prozedural durch Ausführung des Befehls WinClose() geschlossen werden. In beiden Fällen befindet sich das Fenster noch im Speicher und es kann auf die Eigenschaften der enthaltenen Objekte zugegriffen werden. Das Fenster wird erst durch die erneute Ausführung von WinClose() entladen.

Beispiele:

```
// Fenster 'Message' ladenWinOpen('Message');...// Fenster $Message entladen$Message->WinClose();
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Fenster (obj) ungültig

WinDialog(alpha1[, int2[,
handle3]]) : int



Fenster laden und ausführen

alpha1 Fenstername


int2 Optionen (optional; siehe Text)

handle3 Elternfenster (optional)

Schaltflächen-ID

_WinIdClose Schließen-Schaltfläche
gedrückt

_WinIdOk OK-Schaltfläche
gedrückt

Resultat int _WinIdCancel Abbrechen-Schaltfläche 
gedrückt

_WinIdYes Ja-Schaltfläche
gedrückt

_WinIdNo Nein-Schaltfläche
gedrückt

Siehe Verwandte Befehle, WinDialogRun()

Dieser Befehl lädt das Fenster (alpha1) und zeigt es an. Ist kein Dialog-Objekt mit dem angegebenen Namen vorhanden oder ist die Berechtigung des Benutzers nicht ausreichend, wird kein Dialog angezeigt und der globale Fehlerwert auf _rNoRec gesetzt.

Zur Darstellung des Fensters werden die Eigenschaften verwendet, die zum Entwurfszeitpunkt angegeben wurden. Kann aufgrund der Area-Eigenschaften das Fenster nicht im sichtbaren Bereich dargestellt werden, wird es automatisch in die Mitte des primären Bildschirms verschoben.

Je nach verwendetem Betriebssystem stehen unterschiedlich viele Ressourcen (Benutzer- und GDI-Objekte) zur Darstellung des Fensters zur Verfügung. Nach dem Laden des Fensters werden die verbleibenden Ressourcen überprüft. Stehen weniger als 10% zur Verfügung, wird der Dialog nicht geladen. Die Anweisung gibt 0 als Deskriptor zurück und der globale Fehlerwert wird auf _ErrOutOfMemory gesetzt. Die zur Verfügung stehenden Benutzer- und GDI-Objekte können über den Info-Dialog und die Eigenschaften ObjectsUserLimit und ObjectsGDILimit ermittelt werden.



Weiterführende Informationen unter Limitationen des Anwendungsprozesses

Folgende Optionen (int2) können angegeben werden:

- _WinDialogAlwaysOnTop

Das Fenster (alpha1) wird immer im Vordergrund angezeigt.

- _WinDialogApp

Der CONZEPT 16-Client wird versteckt.

- _WinDialogCenter

Das Fenster (alpha1) wird zum Elternfenster (handle3) zentriert.

- WinDialogCenterX

Das Fenster (alpha1) wird zum Elternfenster (handle3) horizontal zentriert.

- WinDialogCenterY

Das Fenster (alpha1) wird zum Elternfenster (handle3) vertikal zentriert.

- WinDialogCenterScreen

Das Fenster (alpha1) wird zum Bildschirm zentriert.

- WinDialogCenterScreenX

Das Fenster (alpha1) wird zum Bildschirm horizontal zentriert.

- WinDialogCenterScreenY

Das Fenster (alpha1) wird zum Bildschirm vertikal zentriert.

- WinDialogCreateHidden

Obsolet. Sollte nicht mehr verwendet werden.

- WinDialogMaximized

Das Fenster (alpha1) wird maximiert dargestellt.

- WinDialogMinimized

Das Fenster (alpha1) wird minimiert dargestellt.

- WinDialogNormal

Das Fenster (alpha1) wird weder minimiert noch maximiert dargestellt.

- WinDialogNoActivate

Das Fenster (alpha1) wird inaktiv gestartet.

Die Optionen WinDialogAlwaysOnTop, WinDialogApp und WinDialogNoActivate können mit einer der anderen Optionen kombiniert werden.

Wird (handle3) nicht angegeben, ist der CONZEPT 16-Client das Eltern-Objekt des Dialogs. Andernfalls ist das Objekt, dessen Objektdeskriptor in (handle3) angegeben ist, das Elternfenster. Als Elternfenster kann nur ein Fenster/MDI-Fenster-Objekt angegeben werden.

Erfolgt der Aufruf eines Fensters mit einem Elternfenster (int3), erscheint das Fenster nicht als separater Task in der Taskleiste.

Ein Wechsel des Fokus in das Elternfenster ist nicht möglich. Der Eingabefokus kann erst dann wieder in das Elternfenster zurückgelangen, wenn der Fenster geschlossen wurde.

Das Fenster wird mit dem Drücken einer Schaltfläche, deren Eigenschaft TypeButton auf WinBtnClose gesetzt ist, verlassen. Das Resultat ist der Wert der Eigenschaft ID der gedrückten Schaltfläche.



MDI-Fenster-Objekte müssen mit dem Befehl WinAdd() bzw. WinAddByName() geladen werden.

Beispiele:

```
// Fenster laden und anzeigenWinDialog('FrmKunden');if (ErrGet() != _ErrOk)...// Fenster 'Message
```

Mögliche Laufzeitfehler:

_ErrStringOverflow Fenstername (alpha1) länger als 40 Zeichen

Kontakt

obj -> WinDialogBox(alpha1,
alpha2, int3, int4, int5) : int
Meldungsfenster aufrufen



obj Elternfenster

alpha1 Fenstertitel

alpha2 Meldungstext

Fenstersymbol

_WinIcoApplication Anwendungssymbol

int3 _WinIcoInformation Informationssymbol

_WinIcoError Fehlersymbol

_WinIcoWarning Warnungssymbol

_WinIcoQuestion Fragesymbol

Optionen

_WinDialogOK OK-Schaltfläche
anzeigen

_WinDialogOKCancel OK- und
Abbrechen-Schaltfläche
anzeigen

int4 _WinDialogYesNo Ja- und
Nein-Schaltfläche
anzeigen

_WinDialogYesNoCancel Ja-, Nein- und
Abbrechen-Schaltfläche
anzeigen

_WinDialogAlwaysOnTop Meldungsfenster immer
im Vordergrund
darstellen

_WinDialogUtf8 UTF-8-Zeichensatz

int5 Standard-Schaltflächennummer

Schaltflächen-ID

_WinIdClose Schließen-Schaltfläche
gedrückt

Resultat int _WinIdOk OK-Schaltfläche gedrückt

_WinIdCancel Abbrechen-Schaltfläche
gedrückt

_WinIdYes Ja-Schaltfläche gedrückt

_WinIdNo Nein-Schaltfläche gedrückt

Siehe Verwandte Befehle

Mit diesem Befehl wird ein Meldungsfenster aufgerufen. Bei Angabe eines Elternfensters in (obj) verhält sich das Meldungsfenster modal zu diesem Fenster.



Sofern das Meldungsfenster nicht modal aufgerufen werden soll, wird in (obj) 0 angegeben, und dieses vor den Befehlsargumenten aufgeführt:

```
WinDialogBox(0, alpha1, alpha2, int3, int4, int5);
```

Standardmäßig wird die Positionierung der DialogBox vom Betriebssystem

Kontakt

übernommen (zentriert zum Bildschirm). Über die Eigenschaften DialogBoxTop und DialogBoxLeft des _App-Objekts kann die Positionierung auch individuell vorgenommen werden.


In der Titelzeile wird der Text in (alpha1) ausgegeben. Der Meldungstext wird in (alpha2) übergeben. Dieser Text wird in einer Zeile innerhalb des Fensters angezeigt. Sind in der übergebenen Zeichenkette Zeilenumbrüche angegeben (StrChar(13)), können auch mehrzeilige Texte ausgegeben werden. Mit Tab (StrChar(9)) kann eine Formatierung der Ausgabe erfolgen. (int3) bestimmt das Symbol im Meldungsfenster. In (int4) werden die Schaltflächen definiert und in (int5) wird die Nummer der Standardschaltfläche angegeben. Die Schaltflächen werden von links (mit 1) nach rechts durchnummeriert. Sollen Zeichenketten im UTF-8-Zeichensatz ausgegeben werden, muss die Angabe der Schaltflächen in (int4) mit der Konstante _WinDialogUtf8 kombiniert werden.

Die Option _WinDialogAlwaysOnTop kann mit den Schaltflächen-Optionen kombiniert werden.

Beispiele:

Der Befehl kann zur einfachen Fehlersuche im Prozedurtext verwendet werden. Es ist dabei zu beachten, dass durch den Aufruf einer Dialogbox der Eingabefokus in die Dialogbox wechselt und unter Umständen Ereignisse aufgerufen werden.

```
// Speichern-Abfrage durchführen if (WinDialogBox(0, 'Application', 'Save record?',
```

obj -> WinDialogResult([int1]) : 

int

Fenster-Resultat setzen/ermitteln

obj Fenster

int1 Neue Schaltflächen-ID
(optional)

Resultat int Aktuelle Schaltflächen-ID

Siehe Verwandte Befehle,
WinDialog(), WinDialogRun()


Dieser Befehl gibt die ID der gedrückten Schaltfläche zurück. Wenn (int1) übergeben wird, wird diese Zahl als ID gesetzt.

Beispiel:

```
// Durchführung einer Schleife bis zum Drücken der Abbrechen-SchaltflächeWinDialog('Message', _W
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Fenster (obj) ungültig

obj -> WinDialogRun([int1[,  handle2]]) : int

Geladenes Fenster anzeigen

obj Fenster

int1 Optionen (optional;
siehe Text)

handle2 Elternfenster
(optional)

Resultat int Schaltflächen-ID

Verwandte Befehle,

Siehe WinOpen(),
WinOpen(), Beispiel

Mit diesem Befehl wird ein geladenes Fenster-Objekt angezeigt. Das Objekt muss zuvor mit dem Befehl WinOpen() geladen worden sein. Als (obj) wird der zurückgegebene Deskriptor oder der Name des Fenster-Objektes mit einem vorangestellten \$ (zum Beispiel \$Meldung) verwendet.

Folgende Optionen (int1) können angegeben werden:

- WinDialogAlwaysOnTop

Das Fenster (obj) wird immer im Vordergrund angezeigt.

- WinDialogApp

Der CONZEPT 16-Client wird versteckt.

- WinDialogAsync

Das Fenster (obj) wird asynchron zur laufenden Prozedur angezeigt.

- WinDialogCenter

Das Fenster (obj) wird zum Elternfenster (handle2) zentriert.

- WinDialogCenterX

Das Fenster (obj) wird zum Elternfenster (handle2) horizontal zentriert.

- WinDialogCenterY

Das Fenster (obj) wird zum Elternfenster (handle2) vertikal zentriert.

- WinDialogCenterScreen

Das Fenster (obj) wird zum Bildschirm zentriert.

- WinDialogCenterScreenX

Das Fenster (obj) wird zum Bildschirm horizontal zentriert.

- WinDialogCenterScreenY

Das Fenster (obj) wird zum Bildschirm vertikal zentriert.

- WinDialogCreateHidden

Obsolet. Sollte nicht mehr verwendet werden.

- WinDialogMaximized

Kontakt

Das Fenster (obj) wird maximiert dargestellt.

- WinDialogMinimized

Das Fenster (obj) wird minimiert dargestellt.

- WinDialogNormal

Das Fenster (obj) wird weder minimiert noch maximiert dargestellt.

- WinDialogNoActivate

Das Fenster (obj) wird inaktiv gestartet.

Die Optionen WinDialogAlwaysOnTop, WinDialogApp, WinDialogAsync und WinDialogNoActivate können mit einer der anderen Optionen kombiniert werden.

Wird (handle2) nicht angegeben, ist der CONZEPT 16-Client das Eltern-Objekt des Dialoges. Andernfalls ist das Objekt, dessen Objektdeskriptor in (handle2) angegeben ist, das Elternfenster. Als Elternfenster kann nur ein Fenster/MDI-Fenster-Objekt angegeben werden.

Der Wert der Eigenschaft ID der gedrückten Schaltfläche im Fenster wird als Resultat zurückgegeben.

Über die Funktion WinDialogResult() kann der Rückgabewert von WinDialogRun() individuell gesetzt werden.

Nach dem Schließen (entweder durch das Drücken einer Schließen-Schaltfläche oder durch die Ausführung der Anweisung WinClose()) befindet sich das Fenster noch im Speicher. Es kann also auch nach der Anweisung WinDialogRun() auf die Eigenschaften und Objekte des Dialoges zugegriffen werden. Der Dialog wird erst nach einem erneuten Aufruf von WinClose() aus dem Speicher entfernt und kann auch erst dann wieder geladen und angezeigt werden.

Beispiel:


```
// Fenster ladenWinOpen('Message');// Fenster zentriert anzeigenRes # $Message->WinDialogRun(_Wi
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Fenster (obj) ungültig



WinOpen(alpha1[, int2[, int3]]) : int
System-Fenster/Fenster-Objekt laden

	Fenstername	
	<u>WinComFileOpen</u>	System-Fenster "Datei öffnen" laden
	<u>WinComFileSave</u>	System-Fenster "Datei speichern" laden
	<u>WinComPath</u>	System-Fenster "Pfadauswahl" laden
alpha1	<u>WinComPrint</u>	System-Fenster "Drucken" laden
	<u>WinComPrintSetup</u>	System-Fenster "Druckeinrichtung" laden
	<u>WinComFont</u>	System-Fenster "Schriftart" laden
	<u>WinComOdbc</u>	ODBC-Auswahl-Dialog laden
	<u>WinC16Info</u>	CONZEPT 16-Fenster "Info" laden
	Optionen (optional)	
	<u>WinOpenDialog</u>	Fenster-Objekt laden
	<u>WinOpenLock</u>	Fenster-Objekt laden und sperren
	<u>WinOpenEventsOff</u>	Fenster-Objekt ohne Ereignis-Verarbeitung laden
int2	<u>WinOpenUnicode</u>	Öffnen- (<u>WinComFileOpen</u>) oder Speichern-Dialog (<u>WinComFileSave</u>) im Unicode-Unterstützung öffnen
int3	Druckerausgabe-Objekt (optional)	
Resultat	<u>handle</u>	Fenster-Objekt 
Siehe	<u>Verwandte Befehle</u> , <u>WinDialogRun()</u> , <u>WinClose()</u> , <u>WinSave()</u>	

Mit diesem Befehl wird ein System-Fenster oder ein Fenster-Objekt in den Hauptspeicher geladen, aber noch nicht angezeigt. Mit dem Befehl WinDialogRun() erfolgt die Anzeige auf dem Bildschirm.

Die Funktion gibt den Deskriptor des System-Fenster/Fenster-Objekts zurück. Konnte das Fenster nicht geladen werden oder besitzt der angemeldete Benutzer keine ausreichende Berechtigung, ist das Ergebnis 0 und der globale Fehlerwert ist auf rNoRec gesetzt.

Je nach verwendetem Betriebssystem stehen unterschiedlich viele Ressourcen (Benutzer- und GDI-Objekte) zur Darstellung des Fensters zur Verfügung. Nach dem Laden des Fensters werden die verbleibenden Ressourcen überprüft. Stehen weniger als 10% zur Verfügung, wird der Dialog nicht geladen. Die Anweisung gibt 0 als Deskriptor zurück und der globale Fehlerwert wird auf _ErrOutOfMemory gesetzt. Die zur Verfügung stehenden Benutzer- und GDI-Objekte können über den Info-Dialog und die Funktionen ObjectsUserLimit und ObjectsGDILimit ermittelt werden.



Weiterführende Informationen unter Limitationen des Anwendungsprozesses

Laden eines System-Fensters

Je nach Art des System-Fensters wird in (alpha1) eine der Stringkonstanten übergeben. (int2) wird nicht angegeben. Bei den Dialogen "Drucken" und "Druckeinrichtung" muss in (obj3) der Deskriptor eines Druckerausgabe-Objektes übergeben werden.

Folgende System-Fenster (int1) stehen zur Verfügung:

- _WinComFileOpen

Ein Fenster zum Öffnen einer Datei wird geladen.

- _WinComFileSave

Ein Fenster zum Speichern einer Datei wird geladen.

- _WinComPath

Ein Fenster zum Auswählen eines Pfades wird geladen.

- _WinComPrint

Ein Fenster zum Drucken wird geladen.

- _WinComPrintSetup

Ein Fenster zum Anpassen der Drucker-Einstellungen wird geladen.

- _WinComFont

Ein Fenster zum Auswählen der Schriftart wird geladen.

- _WinComOdbc

Ein Dialog zur Angabe einer ODBC-Datenquelle wird geladen.

- _WinC16Info

Der CONZEPT 16-Info-Dialog wird geladen.

Die System-Fenster geben den Wert 0 zurück, wenn die OK-Schaltfläche gedrückt wurde. Wird das Fenster geschlossen, wird -1 zurückgegeben.

Die Eigenschaften, die bei System-Fenstern gesetzt/ermittelt werden können, sind bei den entsprechenden Konstanten erläutert.

Laden eines Fenster-Objekts

Der Name des Fenster-Objektes wird in (alpha1) und in (int2) die Option WinOpenDialog angegeben. Nach dem Öffnen eines Objekts ist der Suchpfad automatisch auf dieses Objekt gesetzt. Es werden also keine Namen von zuvor geladenen Objekten gefunden. Der Suchpfad kann mit dem Befehl WinSearchPath() gesetzt werden.

Beispiel 1:

```
// Fenster-Objekt laden und anzeigen// Fenster 'Message' in tHdl ladentHdl # WinOpen('Message', _
```

Beispiel 2:

```
// System-Fenster zum Speichern einer Datei aufrufen// System-Fenster "Datei speichern" in tHdl l
```

Beispiel 3:

```
// System-Fenster "Druckeinrichtung" laden und aufrufen// Druckausgabe-Objekt ladentPrintDev # Pr
```

Laden eines Fensters zum Bearbeiten

Wird ein Fenster-Objekt mit der Anweisung WinOpen(..., _WinOpenDialog | _WinOpenLock) geöffnet, kann es verändert und wieder in der Datenbank gespeichert werden. Die Speicherung erfolgt mit der Anweisung WinSave(). Das Objekt kann nur dann geladen und gesperrt werden, wenn es zu diesem Zeitpunkt nicht durch den eigenen oder einen anderen Benutzer geändert wird. Die Sperre bleibt bis zum Schließen des Objekts mit der Anweisung WinClose() erhalten. Wird in dieser Zeit der Dialog im Designer geöffnet erscheint die Meldung "Frame mit Name ... wird bereits editiert."

Ist das Fenster bereits zum Ändern geöffnet, wird von der WinOpen()-Anweisung 0 zurückgegeben und der globale Fehlerwert auf ErrLocked gesetzt. Besitzt der Benutzer keine Rechte zum Ändern des Fensters, ist der Fehlerwert ErrRights.

Beim Öffnen zum Ändern wird das Objekt geladen, ohne das Ereignisse ausgeführt werden. Das Objekt kann anschließend auch mit den Anweisungen WinDialogRun() bzw. WinAdd() angezeigt werden, ohne das Ereignisse aufgerufen werden.

Laden eines Fensters ohne Ereignis-Verarbeitung

Wird ein Fenster-Objekt mit der Anweisung WinOpen(..., _WinOpenDialog | _WinOpenEventsOff) geöffnet, ist die Ereignisverarbeitung des Dialogs deaktiviert. In allen anderen Dialogen werden die Ereignisse weiterhin ausgeführt.


Laden eines Systemfensters mit Unicode-Unterstützung

Wird ein Öffnen- (WinComFileOpen) bzw. Speichern-Dialog (WinComFileSave) mit der Option _WinOpenUnicode gestartet, werden die Eigenschaften Caption, FileName, PathName, FileNameExt und FileFilter als UTF8-kodierte Zeichenketten erwartet und zurückgegeben. Zur Umwandlung zwischen UTF8 und CONZEPT 16-Zeichensatz können folgende Makros verwendet werden:

```
define{ // Konvertierung von Zeichen: CONZEPT 16-Zeichensatz UTF8 mC16toUtf8(aText) : StrC
```

Mögliche Laufzeitfehler:

ErrStringOverflow Fenstername (alpha1) länger als 40 Zeichen

obj -> WinSave(int1[,  alpha2]) : int;

Fenster-Objekt speichern

obj Deskriptor des Fenster-Objekts
 Optionen
 _WinSaveDefault bestehendes
 Objekt nicht
 überschreiben
 _WinSaveOverwrite bestehendes
 Objekt
 überschreiben

alpha2 Name des Fenster-Objekts
 (optional)

Resultat int Fehlerwert

Siehe WinOpen(), WinCreate()

Mit dieser Anweisung kann ein mit WinOpen() geöffnetes oder mit WinCreate() erstelltes Fenster-Objekt in der Datenbank gespeichert werden. Wurde das Objekt mit WinOpen() geöffnet, muss dies mit der Option _WinOpenLock gesperrt worden sein.

Als (obj) wird der Deskriptor des Fenster-Objekts übergeben, der von WinOpen() bzw. WinCreate() zurückgegeben wurde. Als Optionen stehen folgende Konstanten zur Verfügung:

- _WinSaveDefault (0) - bestehendes Objekt nicht überschreiben

Das Objekt wird unter dem in (alpha2) übergebenem Namen gespeichert. Existiert das Objekt bereits, wird der Fehler _ErrExists zurückgegeben.

- _WinSaveOverwrite (1) - bestehendes Objekt überschreiben

Das Objekt wird unter dem in (alpha2) angegebenen Namen gespeichert, auch dann, wenn ein gleichnamiges Objekt bereits existiert. Das entsprechende Objekt wird überschrieben.

Wird kein Name angegeben, wird der Name des Objekts (Eigenschaft Name) verwendet. Es wird nicht zwischen Groß- und Kleinschreibung unterschieden.

Der Rückgabewert der Anweisung kann mit folgenden Konstanten verglichen werden:

<u>_ErrOk</u>	Das Objekt wurde gespeichert (kein Fehler).
<u>_ErrNameInvalid</u>	Der Name für das Objekt ist ungültig (Argument (alpha2)).
<u>_ErrLocked</u>	Das Objekt wird vom aktuellen Client im Designer editiert oder ist bereits durch <u>_WinOpenLock</u> geöffnet.
<u>_ErrExists</u>	Ein Objekt unter diesem Namen existiert bereits in der Datenbank und <u>_WinSaveOverwrite</u> ist nicht angegeben.
<u>_ErrRights</u>	Benutzerrechte nicht ausreichend.
<u>_rDeadlock</u>	Verklemmung aufgetreten

Mögliche Laufzeitfehler:

Kontakt

<u>ErrHdlInvalid</u>	Der in (obj) übergebene Deskriptor ist ungültig, ist kein Fenster-Objekt oder nicht gesperrt.
<u>ErrValueInvalid</u>	In (int1) ist ein ungültiger Wert übergeben worden.
<u>ErrStringOverflow</u>	In (alpha2) wurde ein Name mit mehr als 40 Zeichen übergeben.

obj -> WinUrmDialog(int1)



Fenster der Benutzerverwaltung laden und ausführen

obj Elternfenster

Optionen

int1 WinUrmFlagHelpTipOnSysProps Anzeige der
Eigenschaftsnamen
im Helptipp

Siehe Verwandte Befehle, WinDialog(), WinDialogRun()

Mit dieser Anweisung wird die CONZEPT 16-Benutzerverwaltung aufgerufen. Sie entspricht der Benutzerpflege, die in der Entwicklungsumgebung aufgerufen werden kann. Der Dialog ist in verschiedenen Sprachen verfügbar. Die Sprache kann über die Eigenschaft LangDisplay des App-Objekts gesteuert werden.

Als Objekt kann ein Fenster-Objekt übergeben werden. Das Fenster ist dann das Eltern-Objekt zum Benutzerpflege-Dialog. Wird kein Objekt angegeben, muss der Benutzerpflege-Dialog wie folgt aufgerufen werden:

```
WinUrmDialog(0, <Optionen>);
```

In (int1) kann die Option WinUrmFlagHelpTipOnSysProps angegeben werden. Sie bewirkt, dass bei Einstellungen, die einer Eigenschaft entsprechen, der Name der Eigenschaft im Helptip angezeigt wird.

Beispiel:

```
tHdlFrame->WinUrmDialog(_WinUrmFlagHelpTipOnSysProps);
```

Konstanten für Dialog-Befehle
Konstanten für Dialog-Befehle
Siehe Dialog-Befehle

Konstanten

- WinAddHidden
- WinDialogAlwaysOnTop
- WinDialogApp
- WinDialogAsync
- WinDialogBoxDefault
- WinDialogBoxUseFont
- WinDialogBoxUseFontButton
- WinDialogBoxUseTextButton
- WinDialogCenter
- WinDialogCenterScreen
- WinDialogCenterScreenX
- WinDialogCenterScreenY
- WinDialogCenterX
- WinDialogCenterY
- WinDialogCreateHidden
- WinDialogMaximized
- WinDialogMinimized
- WinDialogNoActivate
- WinDialogOK
- WinDialogOKCancel
- WinDialogUtf8
- WinDialogYesNo
- WinDialogYesNoCancel
- WinIcoApplication
- WinIcoError
- WinIcoExternFile
- WinIcoInformation
- WinIcoQuestion
- WinIcoWarning
- WinOpenDialog
- WinOpenEventsOff
- WinOpenLock
- WinOpenUnicode
- WinUpdScrollPos
- WinUrmFlagHelpTipOnSysProps

_WinAddHidden
MDI-Fenster unsichtbar laden
Wert 1 / 0x00000001

Verwandte

Siehe Befehle, WinAdd(),
WinAddByName()

Option bei WinAdd() und WinAddByName() durch die ein MDI-Fenster unsichtbar geladen werden kann.

Nach dem Laden des MDI-Fensters stehen alle Eigenschaften und Objekte des Fensters zur Verfügung. Eventuell definierte Ereignisse werden verarbeitet. Mit WinUpdate() und dem Parameter _WinUpdOn wird das Fenster gezeichnet.

WinCopyDefault
Objekte mit allen Eigenschaften und Ereignissen kopieren
Wert 0 /
0x00000000

Verwandte

Siehe Befehle,

WinCopy()

Option bei WinCopy() durch die alle Eigenschaften und Ereignisse des Objektes und aller Unterobjekte kopiert werden.

_WinCopyNoEvents
Objekte ohne Ereignisse kopieren
Wert 1 /
0x00000001

Verwandte

Siehe Befehle,

WinCopy()

Option bei WinCopy() durch die keine Ereignisse des Objektes und aller Unterobjekte kopiert werden.

WinDialogAlwaysOnTop
Fenster ist immer im Vordergrund

Wert -2.147.483.648 /
0x80000000

Verwandte

Befehle,

Siehe WinDialog(),
WinDialogRun(),
WinDialogBox()

Option bei WinDialog(), WinDialogRun() und WinDialogBox().

Wird diese Konstante angegeben, wird das Fenster immer im Vordergrund angezeigt.
Die Option kann nicht bei Objekten AppFrame, TrayFrame und den Systemdialogen
verwendet werden.

WinDialogApp
CONZEPT 16-Client unsichtbar setzen
Wert 536.870.912 /
0x20000000

Verwandte

Siehe Befehle,
WinDialog(),
WinDialogRun()

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der CONZEPT 16-Client nicht mehr auf dem Desktop und in der Taskbar angezeigt.

WinDialogAsync

Fenster wird asynchron zur laufenden Prozedur aufgerufen


Wert 268.435.456 /
0x10000000

Verwandte

Siehe Befehle,

WinDialogRun()


Ist diese Konstante bei WinDialogRun() angegeben, wird die Prozedur nach dem Aufrufen des Dialoges fortgesetzt.

 Es darf immer nur ein asynchroner Dialog gestartet sein. Wird ein weiterer asynchroner Dialog gestartet, wird der Laufzeitfehler ErrValueInvalid erzeugt. In der Regel erfolgt im weiteren Programmverlauf eine Schleife, die in regelmäßigen Abständen überprüft, ob der asynchrone Dialog geschlossen wurde (siehe WinDialogResult()).


Damit die Prozessorzeit nicht (fast) ausschließlich zur Verarbeitung dieser Schleife verwendet wird, kann mit dem Befehl WinSleep() eine Wartezeit implementiert werden, die von CONZEPT 16 genutzt wird, um die Windows-Nachrichten-Schleife abzufragen und ein eingetroffenes Ereignis zu verarbeiten. Die Dauer der Wartezeit sollte von der Verarbeitung innerhalb der Schleife abhängig gemacht werden. Es kann ebenfalls eine Dauer von 0 Millisekunden angegeben werden.

Beispiel:

```
// Frame ladentFrame # WinOpen('Frame', _WinOpenDialog);// Frame asynchron startentFrame->WinDial
```

 Damit das Fenster über die Schaltfläche geschlossen werden kann, muss die Eigenschaft ID auf den Wert gesetzt werden, der in der Schleife überprüft wird. In diesem Fall _WinIdClose, es kann aber auch jede andere ID verwendet werden.

Werden in dem Dialog vordefinierte Schaltflächen verwendet (siehe TypeButton), werden automatisch den Schaltflächen entsprechende IDs gesetzt (siehe ID-Ausprägungen).

Um zu verhindern, dass der asynchrone Dialog mit  geschlossen werden kann, muss die Eigenschaft StyleCloseBox auf false gesetzt werden. Prozedural kann das Fenster über die Anweisung tFrame->WinDialogResult(_WinIdClose) geschlossen werden.

_WinDialogBoxDefault

Standard-Einstellung für Nachrichtenfenster.

Wert 0 / 0x00000000

Siehe WinDialogBox()

Option bei der Eigenschaft DialogBoxFlags des _App-Objektes.

Wird dieser Wert angegeben, wird in Nachrichtenfenstern, die mit WinDialogBox() angezeigt werden, die Standard-Schriftart und die Standard-Schaltflächenbeschriftung verwendet.

_WinDialogBoxUseFont

Schriftart für Nachrichtenfenster.

Wert 1 / 0x00000001

Siehe WinDialogBox()

Option bei der Eigenschaft DialogBoxFlags des _App-Objektes.

Nachfolgende Aufrufe von WinDialogBox() verwenden für die Textanzeige die Schriftart, die in der Eigenschaft DialogBoxFont hinterlegt ist. Die Schriftart wird auch für die Anzeige der Schaltflächen verwendet, sofern die Option _WinDialogBoxUseFontButton nicht gesetzt ist. Die Größe des Nachrichtenfensters wird entsprechend der Schriftart angepasst.



Diese Option wirkt sich nicht aus, wenn das Elternfenster der Dialogbox im Modern Theme Style dargestellt (Eigenschaft StyleTheme = _WinStyleThemeModern) wird.

_WinDialogBoxUseFontButton

Schriftart für Schaltflächen in Nachrichtenfentern.

Wert 2 / 0x00000002

Siehe WinDialogBox()

Option bei der Eigenschaft DialogBoxFlags des _App-Objektes.

Nachfolgende Aufrufe von WinDialogBox() verwenden für die Textanzeige die Schriftart, die in der Eigenschaft DialogBoxFontButton hinterlegt ist. Die Größe der Schaltflächen wird entsprechend der Schriftgröße angepasst - ebenso die daraus resultierende Größe des Nachrichtenfenters.



Diese Option wirkt sich nicht aus, wenn das Elternfenster der Dialogbox im Modern Theme Style dargestellt (Eigenschaft StyleTheme = WinStyleThemeModern) wird.

Kontakt

WinDialogBoxUseTextButton

Beschriftung für Schaltflächen in Nachrichtenfestern.

Wert 4 / 0x00000004

Siehe WinDialogBox()

Option bei der Eigenschaft DialogBoxFlags des _App-Objektes.

Nachfolgende Aufrufe von WinDialogBox() verwenden für die Textanzeige der Schaltflächen die Beschriftung, die in der Eigenschaft DialogBoxTextButton hinterlegt ist.

_WinDialogCenter

Fenster wird zum Elternobjekt zentriert

Wert 3 / 0x00000003

Verwandte Befehle,

Siehe WinDialog(),

WinDialogRun(),

_WinDialogCenterScreen

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der Dialog zum Elternobjekt zentriert.

Wird die Konstante bei der Eigenschaft DialogBoxArrange des _App-Objektes angegeben, dann werden die folgenden mit WinDialogBox() erzeugten Dialog-Boxen zum Elternobjekt zentriert. Damit sich die Eigenschaft DialogBoxArrange auswirkt, müssen die Eigenschaften DialogBoxLeft und DialogBoxTop auf den Wert -1 gesetzt sein.

Ist durch die Darstellung das Fenster in keinem Bildschirm sichtbar, wird es auf dem nächsten Bildschirm zentriert dargestellt.

_WinDialogCenterScreen
Fenster wird zum Bildschirm zentriert
Wert 12 / 0x0000000C

Verwandte Befehle,
Siehe WinDialog(),
WinDialogRun(),
_WinDialogCenter

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der Dialog zum Bildschirm zentriert.

Wird die Konstante bei der Eigenschaft DialogBoxArrange des _App-Objektes angegeben, dann werden die folgenden mit WinDialogBox() erzeugten Dialog-Boxen zum Bildschirm zentriert. Damit sich die Eigenschaft DialogBoxArrange auswirkt, müssen die Eigenschaften DialogBoxLeft und DialogBoxTop auf den Wert -1 gesetzt sein.

Ist durch die Darstellung das Fenster in keinem Bildschirm sichtbar, wird es im primären Bildschirm zentriert dargestellt.

_WinDialogCenterScreenX

Fenster wird horizontal zum Bildschirm zentriert

Wert 4 / 0x00000004

Verwandte Befehle,

Siehe WinDialog(),

WinDialogRun(),

_WinDialogCenterX

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der Dialog horizontal zum Bildschirm zentriert.

Wird die Konstante bei der Eigenschaft DialogBoxArrange des _App-Objektes angegeben, dann werden die folgenden mit WinDialogBox() erzeugten Dialog-Boxen horizontal zum Bildschirm zentriert. Damit sich die Eigenschaft DialogBoxArrange auswirkt, müssen die Eigenschaften DialogBoxLeft und DialogBoxTop auf den Wert -1 gesetzt sein.

Ist durch die Darstellung das Fenster in keinem Bildschirm sichtbar, wird es im primären Bildschirm zentriert dargestellt.

_WinDialogCenterScreenY

Fenster wird vertikal zum Bildschirm zentriert

Wert 8 / 0x00000008

Verwandte Befehle,

Siehe WinDialog(),

WinDialogRun(),

_WinDialogCenterY

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der Dialog vertikal zum Bildschirm zentriert.

Wird die Konstante bei der Eigenschaft DialogBoxArrange des _App-Objektes angegeben, dann werden die folgenden mit WinDialogBox() erzeugten Dialog-Boxen vertikal zum Bildschirm zentriert. Damit sich die Eigenschaft DialogBoxArrange auswirkt, müssen die Eigenschaften DialogBoxLeft und DialogBoxTop auf den Wert -1 gesetzt sein.

Ist durch die Darstellung das Fenster in keinem Bildschirm sichtbar, wird es im primären Bildschirm zentriert dargestellt.

_WinDialogCenterX

Fenster wird horizontal zum Elternobjekt zentriert

Wert 1 / 0x00000001

Verwandte Befehle,

Siehe WinDialog(),

WinDialogRun(),

_WinDialogCenterY

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der Dialog horizontal zum Elternobjekt zentriert.

Wird die Konstante bei der Eigenschaft DialogBoxArrange des _App-Objektes angegeben, dann werden die folgenden mit WinDialogBox() erzeugten Dialog-Boxen horizontal zum Elternobjekt zentriert. Damit sich die Eigenschaft DialogBoxArrange auswirkt, müssen die Eigenschaften DialogBoxLeft und DialogBoxTop auf den Wert -1 gesetzt sein.

Ist durch die Darstellung das Fenster in keinem Bildschirm sichtbar, wird es im primären Bildschirm zentriert dargestellt.

_WinDialogCenterY

Fenster wird vertikal zum Elternobjekt zentriert

Wert 2 / 0x00000002

Verwandte Befehle,

Siehe WinDialog(),

WinDialogRun(),

_WinDialogCenterX

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der Dialog vertikal zum Elternobjekt zentriert.

Wird die Konstante bei der Eigenschaft DialogBoxArrange des _App-Objektes angegeben, dann werden die folgenden mit WinDialogBox() erzeugten Dialog-Boxen vertikal zum Elternobjekt zentriert. Damit sich die Eigenschaft DialogBoxArrange auswirkt, müssen die Eigenschaften DialogBoxLeft und DialogBoxTop auf den Wert -1 gesetzt sein.

Ist durch die Darstellung das Fenster in keinem Bildschirm sichtbar, wird es im primären Bildschirm zentriert dargestellt.

WinDialogCreateHidden
Fenster erst nach Erzeugung anzeigen
Wert 1.073.741.824 /
0x40000000

Verwandte

Siehe Befehle,
WinDialog(),
WinDialogRun()



Die Konstante ist obsolet und sollte nicht mehr verwendet werden.

_WinDialogMaximized
Fenster wird maximiert dargestellt
Wert 16 / 0x00000010

Verwandte

Befehle,

WinDialog(),

Siehe WinDialogRun(),

WinAdd(),

WinAddByName(),

WinUpdate(),

WinInfo()

Je nach Befehl hat _WinDialogMaximized folgende Bedeutung:

- WinDialog(), WinDialogRun(), WinAdd() und WinAddByName()

Dialog wird maximiert geladen.

- WinUpdate()

Dialog wird zur Laufzeit maximiert. Die Option kann nur angegeben werden, wenn _WinUpdState als ersten Parameter übergeben wird.

- WinInfo()

Liefert WinInfo(_WinState) den Wert _WinDialogMaximized zurück, ist der Dialog maximiert.

_WinDialogMinimized
Fenster wird minimiert dargestellt
Wert 32 / 0x00000020

Verwandte
Befehle,
WinDialog(),
Siehe WinDialogRun(),
WinAdd(),
WinAddByName(),
WinUpdate(),
WinInfo()

Je nach Befehl hat _WinDialogMinimized folgende Bedeutung:

- WinDialog(), WinDialogRun(), WinAdd() und WinAddByName()

Dialog wird minimiert geladen.

- WinUpdate()

Dialog wird zur Laufzeit minimiert. Die Option kann nur angegeben werden, wenn _WinUpdState als ersten Parameter übergeben wird.

- WinInfo()

Liefert WinInfo(_WinState) den Wert _WinDialogMinimized zurück ist der Dialog minimiert.

_WinDialogNoActivate
Fenster wird inaktiv gestartet
Wert 16.777.216 / 0x01000000

Verwandte Befehle,

Siehe WinDialog(),
WinDialogRun(),
_WinDialogCenterScreen

Ist diese Konstante bei WinDialog() bzw. WinDialogRun() angegeben, wird der Dialog inaktiv gestartet, das heißt er bekommt solange nicht den Fokus, bis der Benutzer ihn anwählt oder die Aktivierung prozedural durchgeführt wird (Beispielweise über WinFocusSet()).

Die Option kann nicht bei Objekten AppFrame und TrayFrame verwendet werden.

WinDialogOk
Fenster hat eine OK-Schaltfläche
Wert 0

Verwandte

Siehe Befehle,

WinDialogBox()

Wird diese Konstante bei dem Befehl WinDialogBox() angegeben, besitzt das Fenster eine OK-Schaltfläche.

_WinDialogOkCancel

Fenster hat eine OK- und eine Abbrechen-Schaltfläche

Wert 2

Verwandte

Siehe Befehle,

WinDialogBox()

Wird diese Konstante bei dem Befehl WinDialogBox() angegeben, besitzt das Fenster eine OK- und eine Abbrechen-Schaltfläche.

WinDialogUtf8
Text in UTF-8 darstellen
Wert 268.435.456 /
0x10000000

Verwandte

Siehe Befehle,

WinDialogBox()

Wird diese Konstante bei dem Befehl WinDialogBox() angegeben, können der Dialog-Box Zeichenketten im UTF-8-Zeichensatz übergeben werden.

_WinDialogYesNo

Fenster hat eine Ja- und eine Nein-Schaltfläche

Wert 3

Verwandte

Siehe Befehle,

WinDialogBox()

Wird diese Konstante bei dem Befehl WinDialogBox() angegeben, besitzt das Fenster eine Ja- und eine Nein-Schaltfläche.

WinDialogYesNoCancel

Fenster hat eine Ja-, eine Nein und eine Abbrechen-Schaltfläche

Wert 4

Verwandte

Siehe Befehle,

WinDialogBox()

Wird diese Konstante bei dem Befehl WinDialogBox() angegeben, besitzt das Fenster eine Ja-, eine Nein- und eine Abbrechen-Schaltfläche.

_WinOpenDialog

Als Dialog öffnen

Wert 1 / 0x01

Siehe WinOpen()

Option bei WinOpen(). Lädt das Fenster als Dialog.

_WinUrmFlagHelpTipOnSysProps

Anzeige der Eigenschaftsnamen im Helptip

Wert 1

Siehe WinUrmDialog()

Option bei WinUrmDialog(). Ist diese Option angegeben werden bei Einstellungen, die eine Eigenschaft repräsentieren, der Name der Eigenschaften im Helptip angezeigt.

GanttGraph-Befehle

Befehle zum Bearbeiten eines GanttGraph-Objekts

Verwandte

Befehle,

Siehe GanttGraph,

Befehlsgruppen,

Befehlsliste

Befehle

- WinGanttBoxAdd
- WinGanttCellInfo
- WinGanttIvlAdd
- WinGanttIvlRemove
- WinGanttLineAdd

obj -> WinGanttIvlAdd(int1, int2,
int3[, alpha4[, alpha5]]) : handle
Intervall zu GanttGraph hinzufügen



obj GanttGraph
int1 Horizontale Position
int2 Vertikale Position
int3 Intervalllänge
alpha4 Intervallname
 (optional)
alpha5 Intervalltitel (optional)
Resultat handle Intervall-Objekt

Siehe Verwandte Befehle,
Interval, GanttGraph,
WinGanttIvlRemove()

Dieser Befehl fügt einem GanttGraph-Objekt (obj) ein Interval-Objekt hinzu. Die Koordinaten werden ab Zelle (0,0) gezählt. Falls ein Intervall hinzugefügt wird, das außerhalb des durch CellCountHorz und CellCountVert definierten Bereichs liegt, wird CellCountHorz bzw. CellCountVert entsprechend erweitert. Optional können Name (siehe Name) und Titel (siehe Caption) übergeben werden.

Im erfolgreichen Fall ist das Resultat ein Deskriptor auf das Interval-Objekt. Im Fehlerfall ist das Ergebnis 0.

Die Hintergrundfarbe der Interval-Objekte kann Transparenzen enthalten. Ein Farbwert mit Transparenzen wird mit dem Befehl WinColorOpacitySet() erzeugt.

Beispiel:

```
// Intervall in der 1. Spalte und der 3. Zeile, mit einer Länge// von 4, dem Namen 'ivl1' und dem
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid GanttGraph (obj) ungültig

Kontakt

```
obj -> WinGanttCellInfo(point1, var point2) : logic
```

Fensterkoordinaten in Zellkoordinaten umrechnen

obj GanttGraph

point1 Fensterkoordinaten relativ zur linken oberen

Objektecke

```
var point2 Zellkoordinaten
```

Resultat logic Fensterkoordinaten liegen auf einer Zelle

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

PrtGanttGraph

Mit dem Befehl kann im GanttGraph- und im PrtGanttGraph ermittelt werden, welche Zellkoordinaten an einer bestimmten Koordinate (point1) relativ zur linken oberen Objektecke liegen. Die Zellkoordinaten werden nach erfolgreicher Durchführung in (point2) geschrieben. Die Zellkoordinaten beginnen an der Position 0 / 0.


Befindet sich an den Fensterkoordinaten eine Zelle, ist das Resultat true, ansonsten false.

Beispiel:

```
sub EvtMouseEvent( aEvt      : event;    // Ereignis aButton      : int;
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Das Objekt (obj) ist weder ein GanttGraph- noch ein PrtGanttGraph-Objekt.

obj -> WinGanttLineAdd(int1[, int2, , logic3, alpha4]) : handle

Hilfslinie zu GanttGraph hinzufügen

obj GanttGraph

int1 Linienanfangspunkt

int2 Linienfarbe (optional)

Linienausrichtung

(optional)

logic3 true Horizontal

false Vertikal

alpha4 Linienname

Resultat handle Linien-Objekt

Verwandte Befehl,

Siehe Line, GanttGraph,

WinGanttIvlRemove()

Dieser Befehl fügt einem GanttGraph-Objekt (obj) ein Line-Objekt hinzu. Der Parameter (int1) gibt die Nummer der Zeile/Splte an, ab der die Hilfslinie gezeichnet wird. Falls (logic3) fehlt oder den Wert true hat, wird eine horizontale Hilfslinie gezeichnet, ansonsten eine vertikale. Fehlt (int2) wird WinColHighLight als Linienfarbe benutzt.

Im erfolgreichen Fall ist das Resultat ein Deskriptor auf das Line-Objekt. Im Fehlerfall ist das Ergebnis 0.

Beispiel:

```
// Hilfslinie in der 2. Spalte in hellrot und mit vertikaler Ausrichtung erzeugen$GanttGraph->WinGanttLineAdd(2, 1, 0, false, "Hilfslinie")
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid GanttGraph (obj) ungültig

obj -> WinGanttBoxAdd(rect1[,
int2[, alpha3[, alpha4]]) :
handle



Box zu GanttGraph hinzufügen

obj GanttGraph
rect1 Box-Koordinaten
int2 Box-Füllfarbe
 (optional)
alpha3 Name der Box
 (optional)
alpha4 Caption der Box
 (optional)

Resultat handle Box-Objekt

Verwandte Befehle,

Siehe WinGanttIvlRemove(),
 GanttGraph, Box

Dieser Befehl fügt einem GanttGraph-Objekt (obj) ein Box-Objekt hinzu. In (rect1) wird die linke obere und rechte untere Ecke des Rechtecks übergeben. Die Angabe erfolgt in Zellkoordinaten. Fehlt (int2), wird WinColHighLight als Füllfarbe verwendet.

Über die Parameter (alpha3) und (alpha4) kann der Name und die Caption des Box-Objekts angegeben werden. Werden die Parameter nicht übergeben, bleiben die Eigenschaften leer.



Damit die Caption (alpha4) angezeigt wird, muss beim GanttGraph in der Eigenschaft GanttFlags der Wert WinGanttIvlBoxShowText gesetzt sein.

Im erfolgreichen Fall ist das Resultat ein Deskriptor auf das Box-Objekt. Im Fehlerfall ist das Ergebnis 0.

Die Hintergrundfarbe der Box-Objekte kann Transparenzen enthalten. Ein Farbwert mit Transparenzen wird mit dem Befehl WinColorOpacitySet() erzeugt.

Beispiel:

```
// Box in erste Zeile und erster Spalte mit drei Zellen Breite// und 2 Zellen Höhe und schwarzer
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid GanttGraph (obj) ungültig

obj -> WinGanttIvlRemove()



Box/Intervall/Linie aus GanttGraph entfernen

obj Box/Interval/Line

Verwandte

Siehe Befehl, Box,
Interval, Line,
GanttGraph

Dieser Befehl entfernt ein Box-/Interval-/Line-Objekt (obj) aus einem GanttGraph-Objekt.

Beispiel:

```
// Alle Box-, Intervall- oder Linien-Objekte aus einem GanttGraph entfernen  
sub ObjectsRemove( aC
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid GanttGraph (obj) ungültig

Menu-Befehle

Befehle zum Bearbeiten eines Menu-Objekts

Verwandte

Befehle,


Siehe MenuItem,

Befehlsgruppen,

Befehlsliste

Befehle

- WinMenuContext
- WinMenuItemAdd
- WinMenuItemRemove

obj -> WinMenuItemAdd([alpha1[,  alpha2[, int3]]) : handle

Menüeintrag zu Menü hinzufügen

obj Menü

alpha1 Menüeintragsname (optional)

alpha2 Menüeintragstitel (optional)

int3 Menüeintragsposition (optional)

Resultat handle Deskriptor des neuen Menüeintrags

Siehe Verwandte Befehle

Dieser Befehl fügt einen Menüeintrag zu einem Menü-Objekt (obj) hinzu. Wird in (obj) ein Menü oder ein Kontextmenü übergeben, wird der Menüeintrag in der ersten Ebene angelegt. Handelt es sich um ein Menü, wird der Eintrag in der Menüzeile angelegt.



Bei einem Kontextmenü muss das Anlegen eines Menüeintrags in dem Ereignis EvtMenuInitPopup vorgenommen werden.

Bei der Übergabe eines bestehenden Menüeintrags in (obj) wird in dessen Untermenü der neue Eintrag hinzugefügt. Existiert noch kein Untermenü, wird ein neues angelegt. In (int3) wird die Position des Menüeintrags angegeben. Wird hier 0 angegeben oder kein Parameter angegeben, wird der Eintrag immer am Ende hinzugefügt. Name (alpha1) (siehe Name) und Titel (alpha2) (siehe Caption) können direkt beim Aufruf bestimmt werden, alle weiteren Eigenschaften müssen mit Hilfe des Resultats gesetzt werden.

Als Resultat wird der Deskriptor des neuen Menüeintrags zurückgeliefert. Im Fehlerfall ist das Ergebnis 0.

Beispiel:

```
// Menüeintrag 'Kopieren' zu Menü tMenuItemEdit hinzufügendMenuItem # tMenuItemEdit->WinMenuItemA
```

obj ->

WinMenuItemRemove([logic1])

Menüeintrag aus Menü entfernen

obj Menüeintrag

Menüeintrag

logic1 leeren

(optional)

Siehe [Verwande](#)
[Befehle](#)

Dieser Befehl entfernt ein Menüeintrag (obj) inklusive aller Untermenüeinträge aus einem Menü. Wird der optionalen Parameter (logic1) auf true gesetzt, werden nur die Untermenüeinträge des Menüeintrags (obj) entfernt.




Bei einem Kontextmenü muss das Entfernen eines Menüeintrags in dem Ereignis EvtMenuInitPopup vorgenommen werden.

Beispiel:

```
// Wenn Menüeintrag vorhanden if (tMenuItem > 0){ // Menüeintrag entfernen tMenuItem->WinMenuIt
```



Kontakt

obj -> WinMenuContext(alpha1, , int2, int3[, int4]) : int
Anzeige eines Kontextmenüs
obj Deskriptor des Eltern-Objekts
alpha1 Name des Kontextmenüs
int2 horizontale Position
int3 vertikale Position
Optionen (optional)
int4 WinMenuContextDefault Verarbeitung im EvtMenuCommand
 WinMenuContextReturnID Rückgabe der MenuId
Resultat int Fehlerwert oder MenuId des
 gewählten Menüpunkts

Siehe Verwandte Befehle

Mit dieser Anweisung kann prozedural ein Kontextmenü angezeigt werden. Als Eltern-Objekt (obj) muss der Deskriptor eines Oberflächen-Objekts angegeben werden.

In (alpha1) wird der Name des Kontextmenüs angegeben. Wird hier eine leere Zeichenkette übergeben, wird das Kontextmenü des übergebenen Objekts (siehe MenuNameCntxt) verwendet. Ist diese Eigenschaft nicht gesetzt oder bei dem Objekt nicht vorhanden, wird der Laufzeitfehler ErrValueInvalid erzeugt.

Die Argumente (int2) und (int3) definieren die Position an der das Menü dargestellt wird. Enthalten beide Werte -1, dann wird das Menü an der aktuellen Position des Mauszeigers angezeigt. Im anderen Fall wird hier eine Position relativ zur linken oberen Ecke des übergebenen Eltern-Objektes angegeben. Handelt es sich bei dem Objekt jedoch um einen TrayFrame, dann geben die Argumente die Position relativ zum Ursprung des primären Bildschirms an.

Das Kontextmenü kann in zwei Modi betrieben werden. Der Modus wird im Parameter (int4) übergeben. Folgende Konstanten können übergeben werden:

- **WinMenuContextDefault (0)** (default)

Wählt der Anwender einen Menüpunkt aus, dann wird das Ereignis EvtMenuCommand durchgeführt. In diesem Fall wird immer ErrOk als Resultat zurückgegeben.

- **WinMenuContextReturnID (1)**

Wählt der Anwender einen Menüpunkt aus, wird lediglich der Wert der Eigenschaft MenuId des ausgewählten MenuItem-Objektes zurückgeliefert. Ist die MenuId < 0 wird der absolute Betrag geliefert, damit kein Konflikt mit Fehlerwerten (< 0) resultiert.

Verlässt der Anwender das Menü ohne einen Menüpunkt auszuwählen, wird ErrOk zurückgeliefert. Es muss darauf geachtet werden, dass für MenuId nur Werte größer 0 definiert werden, wenn das Menü in diesem Modus betrieben werden soll.

In beiden Modi werden die Ereignisse EvtMenuContext und EvtMenuInitPopup des übergebenen Objektes durchgeführt, sofern sie dort eingetragen sind.

Resultate:

Findet die Verarbeitung des ausgewählten Menüpunkts in der EvtMenuCommand statt, wird immer ErrOk zurückgegeben.

Wird der Modus WinMenuContextReturnID verwendet, wird nach dem Anklicken des Menüeintrags dessen MenuId zurückgegeben. Ist die MenuId negativ, wird der Wert in eine positive Zahl gewandelt, um Konflikte mit Fehlerwerten zu vermeiden.

Bei der Rückgabe von ErrUnavailable konnte das in (alpha1) angegebene Menü nicht gefunden werden. Verfügt der Benutzer nicht über ausreichende Rechte, um das Menü auszuführen, wird ErrRights zurückgegeben.

Mögliche Laufzeitfehler:

ErrHdlInvalid In (obj) wurde kein Oberflächen-Objekt übergeben.
Bei dem übergebenen Objekt ist die Eigenschaft MenuNameCntxt
ErrValueInvalid leer oder nicht vorhanden und in (alpha1) wurde kein Menü übergeben.

RecList-Befehle

Befehle zum Bearbeiten eines RecList-Objekts

Verwandte

Befehle,


Siehe RecList,

Befehlsgruppen,

Befehlsliste

Befehle

- WinLstEdit

obj -> WinLstEdit(int1[, int2]) 

: handle

Feld in einer Liste editieren

obj Deskriptor der Liste

int1 Deskriptor der Spalte

Optionen

WinLstEditLst Edit-Objekt
mit Liste

int2 WinLstEditLstAlpha Edit-Objekt
mit
zweispaltiger
Liste

WinLstEditClearChanged Changed-Flag
zurücksetzen

Resultat handle Objekt

Verwandte Befehle, EvtLstEditStart,

Siehe EvtLstEditCommit,

EvtLstEditFinished, Ereignisabläufe

WinLstEdit(), Beispiel

Mit diesem Befehl kann ein Feld innerhalb einer RecList oder DataList editiert werden.

In (obj) und (int1) muss der Deskriptor der Liste und der Spalte übergeben werden. Als Resultat wird der Deskriptor auf das Edit-Objekt zurückgegeben, der von diesem Befehl erzeugt wurde. Das Objekt ist vom gleichen Typ wie die Spalte.

Innerhalb des Listen-Objekts muss eine Zeile selektiert sein. Bei einem DataList-Objekt muss also die Eigenschaft CurrentInt auf eine Zeilennummer gesetzt sein.

In Abhängigkeit von den angegebenen Optionen in (int2) werden neben dem Eingabeobjekt noch weitere Objekte erzeugt:

- 0

Es werden keine weiteren Objekte erzeugt. Das Feld kann editiert werden.

- WinLstEditLst


Es wird ein DataListPopup-Objekt mit einer Spalte erzeugt. Die Spalte hat den gleichen Typ, wie das zu editierende Feld.

- WinLstEditLstAlpha

Es wird ein DataListPopup-Objekt mit zwei Spalten erzeugt. Die erste Spalte hat den gleichen Typ, wie das zu editierende Feld. Die zweite Spalte ist vom Typ alpha und kann mit Informationen zum Eintrag in der ersten Spalte genutzt werden. Der Wert der ersten Spalte wird in das Feld übertragen.

Die angegebenen Optionen können mit der Option WinLstEditClearChanged kombiniert werden. Diese Option setzt das Changed-Flag zurück. Das Flag wird nach dem Ereignis EvtLstEditCommit gesetzt, wenn die Funktion true zurückgegeben hat.

Das Flag wird in dem Ereignis EvtLstEditFinished mit übergeben, um kenntlich zu machen, dass Änderungen an den Daten vorgenommen wurden.

 Das Flag bleibt so lange gesetzt, bis es durch einen Aufruf von `WinLstEdit()` mit der Option WinLstEditClearChanged zurückgesetzt wird.

Bei Verwendung einer DataList ist folgendes zu beachten:

Standardmäßig wird nach Übernahme eines Eintrages aus der Liste im Edit-Objekt verblieben. Soll das Edit-Objekt verlassen und damit die Ereignisse EvtLstEditCommit und EvtLstEditFinished ausgelöst werden, muss in dem Ereignis EvtLstEditStart die Eigenschaft LstFlags des DataListPopup-Objektes auf WinLstEditClose gesetzt werden.

Beispiel:

```
sub EvtMouseEvent( aEvt      : event;      // Ereignis  aButton    : int;          // Maustaste  aHitTes
```


RecView-Befehle

Befehle eines RecView-Objekts

Verwandte

Befehle,

Siehe RecView,

Befehlsgruppen,

Befehlsliste

Befehle

- WinRvwColumn
- WinRvwEdit
- WinRvwUpdate

Kontakt

```
obj -> WinRvwColumn(point1[, int2]) : int;
```



Spalte eines RecView von einer Koordinate ermitteln

obj RecView-Objekt

point1 Position relativ zur linken oberen Ecke
des RecView-Objektes

Optionen (optional)

```
int2      WinRvwColumnHitHeader Spaltenköpfe
                                beachten
```

Resultat int Spaltendeskriptor oder 0



Siehe Verwandte Befehle

Mit diesem Befehl ist es möglich, ausgehend von einer gegebenen Koordinate (point1), den Deskriptor der Spalte zu ermitteln. Dies ist insbesondere im Ereignis EvtMouseEvent hilfreich, wenn dort der Spaltendeskriptor benötigt wird.

Der Rückgabewert liefert den Deskriptor der Spalte an der gegebenen Position oder 0. Ein Deskriptor wird zurückgegeben, wenn die Position (point1) innerhalb des Anzeigebereiches des RecView liegt (ohne Spaltenköpfe und ohne Scrollbars). Sollen die Spaltenköpfe berücksichtigt werden, dann kann die Option (int2) WinRvwColumnHitHeader angegeben werden.


Beispiel:

```
sub EvtMouseEvent( aEvt      : event;    // Ereignis aButton      : int;
```

Mögliche Laufzeitfehler:

ErrHdlInvalid In (obj) wurde kein gültiger Deskriptor eines RecView-Objektes angegeben.

ErrValueInvalid In (int2) wurde eine ungültige Option angegeben.

obj -> WinRvwEdit(int1[, int2[, int3[, int4]]) : 

(Sub-)Item eines RecView-Objektes bearbeiten


obj Objekt

int1 Nummer des
 Items

int2 Nummer des
 Subitems


int3 Nummer des
 Views

int4 Optionen

Resultat int Fehlerwert 

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

Dieser Befehl startet die Bearbeitung eines Items (int1) oder SubItems (int2) in dem View (int3) des RecView-Objektes (obj) durch. Wird kein SubItem (int2) angegeben, oder 0 übergeben, wird das übergeordnete Item bearbeitet.

 Die Gruppe des zu bearbeitenden Items muss selektiert und sichtbar sein. Daher empfiehlt sich vorher die Verwendung von WinRvwUpdate() mit den Update-Modi _WinRvwUpdateFromSelected und _WinRvwUpdateDoKeepSelect. Durch den Befehl wird der Ablauf, wie in Ereignisabläufe des RecViews beschrieben, durchgeführt. Wird daher im Ereignis EvtLstEditStartItem das Resultat _WinRvwGroupEditSkipItem gesetzt, wird das nächste zu bearbeitende Item gesucht.

Die Nummer des Views (int3) gibt an, in welchem View die Gruppe bearbeitet werden soll. Mögliche Werte sind 0 bis 4. Wird 0 angegeben, oder das Argument weggelassen, wird im aktiven View, welches den Eingabe-Fokus besitzt, bearbeitet. Eine Angabe von 0 und 1 ist immer möglich. Die Views 2 bis 4 existieren nur, wenn eine entsprechende Splittung des RecView durch den Anwender vorgenommen wurde. Existiert das angegebene View nicht wird der Fehlerwert _ErrUnknown zurückgegeben.

Als Option (int4) kann _WinRvwEditAbortEditing angegeben werden, um eine aktuell aktive Bearbeitung abubrechen. Diese Option ist standardmäßig aktiv. Soll die Bearbeitung nicht abgebrochen werden, muss 0 übergeben werden. Ist in diesem Fall bereits ein Datensatz in Bearbeitung, wird der Fehlerwert _ErrInUse zurückgegeben.


Als Rückgabewert wird der Fehlerwert _ErrUnknown zurückgegeben, wenn das angegebene View (int3) nicht existiert oder noch nicht initialisiert (siehe EvtLstViewInit) wurde. Wurde als Option (int4) 0 übergeben und es ist bereits ein Datensatz in Bearbeitung wird _ErrInUse zurückgegeben. Ist das Item (int1) oder das SubItem (int2) nicht vorhanden, kein Datensatz markiert, oder der aktive Datensatz nicht im sichtbaren Bereich, gibt der Befehl den Fehlerwert _ErrUnavailable zurück. Ist keines der EvtLstEdit...-Ereignisse bei dem RecView (obj) eingetragen, kommt der Fehlerwert _ErrIllegalOp. Sonst wird _ErrOk zurückgegeben.

Mögliche Laufzeitfehler

_ErrHdlInvalid In (obj) wurde kein gültiger Deskriptor angegeben.

Kontakt

__ErrValueInvalid In den Optionen (int4) wurde weder 0 noch
__WinRvwEditAbortEditing angegeben.

obj -> WinRvwUpdate([int1[, int2[, int3]]) : 

int

View eines RecView-Objektes aktualisieren

obj Objekt

int1 Update-Modus
(siehe Text)

int2 Nummer des
Views
Datensatzpuffer

int3 oder
Dateinummer

Resultat int Fehlerwert 

Verwandte


Siehe Befehle,
WinUpdate()

Dieser Befehl führt eine Positionierung in dem View (int2) des RecView-Objektes (obj) durch.

Folgende Parameter können als Update-Modus (int1) übergeben werden:

<u>_WinRvwUpdateFromFirst</u>	Neuaufbau des RecViews ab dem ersten Datensatz
<u>_WinRvwUpdateFromLast</u>	Neuaufbau des RecViews ab dem letzten Datensatz
<u>_WinRvwUpdateFromRecBuf</u>	Neuaufbau des RecViews ausgehend von den Werten des Feldpuffers (int3)
<u>_WinRvwUpdateFromTop</u>	Neuaufbau des RecViews ab dem ersten sichtbaren Datensatz
<u>_WinRvwUpdateFromTopLocation</u>	Neuaufbau der angezeigten Datensätze des RecViews. Die Position der ersten angezeigten Gruppe wird beibehalten.
<u>_WinRvwUpdateFromSelected</u>	Neuaufbau des RecViews ab dem selektieren Datensatz
<u>_WinRvwUpdateDoSelect</u>	In dem RecView wird der aktuelle Datensatz selektiert
<u>_WinRvwUpdateDoKeepSelect</u>	In dem RecView wird der aktuell selektierte Datensatz weiterhin selektiert
<u>_WinRvwUpdateOptClearCache</u>	Alle Gruppen werden aus dem Cache des RecView-Objektes entfernt
<u>_WinRvwUpdateOptClearSelected</u>	Die selektierte Gruppe wird aus dem Cache des RecView-Objektes entfernt

Konstanten aus den Bereichen _WinRvwUpdateFrom*, _WinRvwUpdateDo* und _WinRvwUpdateOpt* können miteinander kombiniert werden. Konstanten aus gleichen Bereichen können nicht kombiniert werden.

 Wird zwar ein Update-Modus (int1), aber keine Konstante aus dem Bereich _WinRvwUpdateFrom* angegeben, wird der Laufzeitfehler _ErrValueInvalid ausgelöst.

Wird als Update-Modus (int1) _WinRvwUpdateFromRecBuf übergeben, muss in int3 ein Datensatzpuffer oder eine Datei übergeben werden. Diese muss mit der

Eigenschaft DbFileNo oder DbLinkFileNo übereinstimmen. Wird in (int1) nicht WinRvwUpdateFromRecBuf angegeben, so führt eine Angabe von einem Datensatzpuffer oder einer Datei zu dem Laufzeitfehler ErrValueInvalid. Dieser Modus kann auch verwendet werden, um verknüpfte Datensätze anzuzeigen. Dazu wird der Quelldatensatz gelesen und die Quelldatei in int3 übergeben. Um in einer Verknüpfung auf einen bestimmten Datensatz zu positionieren, muss bei dem Verknüpfungsfeld "nur Zugriffspositionierung" aktiviert sein.

Fehlt die Angabe des Update-Modus, so wird WinRvwUpdateFromTop verwendet.

In der Nummer des Views (int2) wird definiert, welche Anzeige aktualisiert werden soll. Mögliche Werte sind 0 bis 4. Wird 0 angegeben, oder das Argument weggelassen, wird das aktive View, welches den Eingabe-Fokus besitzt, aktualisiert. Eine Angabe von 0 und 1 ist immer möglich. Die Views 2 bis 4 existieren nur, wenn eine entsprechende Splittung des RecView durch den Anwender vorgenommen wurde. Existiert das angegebene View nicht wird der Fehlerwert ErrUnknown zurückgegeben.

Der Befehl hat zur Folge, dass die Anzeige des entsprechenden Views neu aufgebaut wird. Daher werden anschließend EvtLstGroupInit-Ereignisse durchgeführt.

Als Rückgabewert wird der Fehlerwert ErrUnknown zurückgegeben, wenn das angegebene View (int2) nicht existiert oder noch nicht initialisiert (siehe EvtLstViewInit) wurde. Der Fehlerwert ErrUnavailable wird zurückgegeben, wenn der Update-Modus WinRvwUpdateFromTop angegeben wurde und kein Datensatz angezeigt wird, oder der Update-Modus WinRvwUpdateFromSelected verwendet wird und kein Datensatz selektiert ist. Sonst wird ErrOk zurückgegeben.



Wird der Befehl in einem der Ereignisse EvtLstViewInit, EvtLstGroupInit, EvtLstEditStartGroup, EvtLstEditStartItem, EvtLstEditActivate oder EvtLstEditEndItem aufgerufen, wird der Laufzeitfehler ErrHdlInvalid ausgelöst.

Mögliche Laufzeitfehler:

ErrHdlInvalid

In (obj) wurde kein gültiger Deskriptor angegeben. Wird beim Update-Modus (int1) WinRvwUpdateFromRecBuf kein Datensatzpuffer oder die Nummer oder ein Datensatzpuffer einer anderen Datei angegeben, wird der Laufzeitfehler ebenfalls ausgelöst. Weiterhin wird der Laufzeitfehler ausgelöst, wenn der Befehl in einem der oben genannten Ereignisse aufgerufen wird.

ErrValueInvalid

In (int3) wurde ein Wert angegeben und der Update-Modus in (int1) ist nicht WinRvwUpdateFromRecBuf oder es wurde ein ungültiger Update-Modus angegeben. Wird in (int1) keine Konstante aus dem Bereich WinRvwUpdateFrom* angegeben, wird der Laufzeitfehler ebenfalls ausgelöst.

RtfEdit-Befehle

Befehle und Konstanten für RtfEdit-Objekt

Verwandte

Befehle,

Siehe RtfEdit,

Befehlsgruppen,

Befehlsliste


Befehle

- RtfTabMake
- WinEmfProcess
- WinRtfLoad
- WinRtfLoadBin
- WinRtfLoadName
- WinRtfPicInsertMem
- WinRtfPicInsertName
- WinRtfSave
- WinRtfSaveBin
- WinRtfSaveName
- WinRtfSearch
- WinRtfTabGet
- WinRtfTabSet

Konstanten

- WinRtfLoadAscii
- WinRtfLoadAuto
- WinRtfLoadInsert
- WinRtfLoadMix
- WinRtfLoadOem
- WinRtfLoadRtf
- WinRtfPicModeAuto
- WinRtfPicModeQuality
- WinRtfPicModeSpeed
- WinRtfSaveAscii
- WinRtfSaveAuto
- WinRtfSaveMark
- WinRtfSaveMix
- WinRtfSaveOem
- WinRtfSaveRtf
- WinRtfSearchCase
- WinRtfSearchDelete
- WinRtfSearchReplace
- WinRtfSearchUp
- WinRtfSearchWord
- WinRtfTabCenter
- WinRtfTabDecimal
- WinRtfTabLeft
- WinRtfTabNone
- WinRtfTabRight
- WinStreamBufField

- WinStreamBufText
- WinStreamCaption
- WinStreamNameBin
- WinStreamNameFile
- WinStreamNameText

obj -> WinRtfLoad(int1[,  int2[, handle3]]) : int

Text in RtfEdit-Objekt laden

obj Objekt (RtfEdit-Objekt)

Quelle des Textes

WinStreamCaption Text aus der
Eigenschaft
Caption

int1 WinStreamBufField Text aus
Feldpuffer
laden

WinStreamBufText Text aus
Textpuffer
laden

Modus (optional)

WinRtfLoadAuto Textformat
automatisch
erkennen

WinRtfLoadRtf RTF-Text
laden

int2 WinRtfLoadAscii ASCII-Text
laden

WinRtfLoadInsert Text in den
bestehenden
Text einfügen

WinRtfLoadMix Text mit
Daten
mischen

int3 Deskriptor des Textpuffers
(optional)

Resultat int Fehlerwert

Verwandte Befehle,

Siehe WinRtfLoadName(),

WinRtfLoadBin(), WinRtfSave()

Mit diesem Befehl wird ein Text in ein RtfEdit-Objekt geladen. Der Deskriptor des Objektes wird in (obj) übergeben.

In (int1) wird die Quelle des zu ladenden Textes definiert. Der Deskriptor der Quelle kann in (handle3) angegeben werden. Wird kein Deskriptor angegeben, wird der Inhalt der entsprechenden Eigenschaft verwendet.

Folgende Quellen können angegeben werden:

- **WinStreamCaption**

Werden prozedural Änderungen der Caption vorgenommen, kann mit dieser Option eine Aktualisierung des Objektes durchgeführt werden. Der Parameter (handle3) darf nicht gesetzt werden.

- **WinStreamBufField**

Der Inhalt des in der Eigenschaft DbFieldName angegebenen Feldpuffers wird geladen. der Parameter (handle3) darf nicht gesetzt sein.

- **WinStreamBufText**

Der Text steht in einem Textpuffer bereit. Der Deskriptor des Textes kann in (handle3) angegeben werden. Ist kein Textpuffer angegeben, wird der in der Eigenschaft DbTextBuf angegebene Textpuffer verwendet.

Der Parameter (int2) bestimmt das Format der Quelle. Folgende Konstanten können übergeben werden:

- **WinRtfLoadAuto**

Das Format der Quelle wird automatisch bestimmt.

- **WinRtfLoadRtf**

Die Textquelle ist im RTF-Format.

- **WinRtfLoadAscii**

Die Textquelle ist im ASCII-Format.

Die Parameter zum Quellenformat können mit WinRtfLoadInsert kombiniert werden, um in einen bestehenden Text den angegebenen Text einzufügen.

Der Text ersetzt den Text, der mit der Markierung (Range) selektiert ist. Ist keine Markierung vorhanden, wird der Text an der aktuellen Cursorposition eingefügt.

Bei der Kombination mit der Option WinRtfLoadMix werden beim Laden des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.

Als Rückgabewert können neben den Fehlerkonstanten aus dem Bereich der externe Dateien die Werte ErrData und ErrRtfSyntaxError zurückgegeben werden. Bei der Rückgabe von ErrOk ist kein Fehler aufgetreten.

obj -> WinRtfLoadBin(handle1, int2[,
alpha3]) : int



Binäres Objekt in RtfEdit-Objekt laden

obj Objekt (RtfEdit-Objekt)

handle1 Deskriptor des binären Objektes

Modus (optional)

WinRtfLoadAuto Textformat
automatisch
erkennen

WinRtfLoadRtf RTF-Text
laden

int2 WinRtfLoadAscii ASCII-Text
laden

WinRtfLoadInsert Text in den
bestehenden
Text einfügen

WinRtfLoadMix Text mit
Daten
mischen

alpha3 Verschlüsselungscode (optional)

Resultat int Fehlerwert

Verwandte Befehle,

Siehe WinRtfLoad(), WinRtfSaveBin(),
BinOpen()

Mit diesem Befehl wird der Inhalt eines binären Objekts in ein RtfEdit-Objekt geladen. Der Deskriptor des RtfEdit-Objektes wird in (obj), der Deskriptor des binären Objekts in (handle1) übergeben.

Der Parameter (int2) bestimmt das zu lesende Format. Folgende Konstanten können angegeben werden:

- **WinRtfLoadAuto**

Das Format der Quelle wird automatisch bestimmt.

- **WinRtfLoadRtf**

Die Textquelle ist im RTF-Format.

- **WinRtfLoadAscii**

Die Textquelle ist im ASCII-Format.

Die Parameter zum Quellenformat können mit WinRtfLoadInsert kombiniert werden, um in einen bestehenden Text den angegebenen Text einzufügen.

Der Text ersetzt den Text, der mit der Markierung (Range) selektiert ist. Ist keine Markierung vorhanden, wird der Text an der aktuellen Cursorposition eingefügt.

Bei der Kombination mit der Option WinRtfLoadMix werden beim Laden des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen

Kontakt

befinden sich im Abschnitt Text und Daten mischen.

Als Rückgabewert können neben den Fehlerkonstanten aus dem Bereich der externe Dateien die Werte _ErrData und _ErrRtfSyntaxError zurückgegeben werden. Bei der Rückgabe von _ErrOk ist kein Fehler aufgetreten.

obj ->

WinRtfLoadName(int1,
int2[, alpha3]) : int



Text in RtfEdit-Objekt laden

obj Objekt (RtfEdit-Objekt)

Quelle des Textes

_WinStreamNameText internen
Text
laden

int1

_WinStreamNameFile externen
Text
laden

Modus

_WinRtfLoadAuto Textformat
automatisch
erkennen

_WinRtfLoadRtf RTF-Text
laden

_WinRtfLoadAscii ASCII-Text
laden

int2

_WinRtfLoadOem OEM-Text
laden

_WinRtfLoadInsert Text in den
bestehenden
Text einfügen

_WinRtfLoadMix Text mit
Daten
mischen

alpha3 Name des Textes oder Feldpuffer
(optional)

Resultat int Fehlerwert

Verwandte Befehle,

Siehe WinRtfLoad(), WinRtfLoadBin(),
WinRtfSaveName()

Mit diesem Befehl wird ein Text in ein RtfEdit-Objekt geladen. Der Deskriptor des Objektes wird in (obj) übergeben.

In (int1) wird die Quelle des zu ladenden Textes definiert. Der Name der Quelle kann in (alpha3) angegeben werden. Wird kein Name angegeben, wird der Inhalt der entsprechenden Eigenschaft verwendet.

Folgende Quellen können angegeben werden:

- **_WinStreamNameText**

Der Text steht in einem internen Text zur Verfügung. Der Name des Textes kann entweder in (alpha3) übergeben werden oder wird der Eigenschaft FileName entnommen.

- **WinStreamNameFile**

Der Text steht in einer externen Datei zur Verfügung. Der Name der Datei kann entweder in (alpha3) übergeben werden oder wird der Eigenschaft FileName entnommen.

Der Parameter (int2) bestimmt das Format der Quelle. Folgende Konstanten können übergeben werden:

- **WinRtfLoadAuto**

Das Format der Quelle wird automatisch bestimmt.

- **WinRtfLoadRtf**

Die Textquelle ist im RTF-Format.

- **WinRtfLoadAscii**

Die Textquelle ist im ASCII-Format.

- **WinRtfLoadOem**

Die Textquelle ist im OEM-Format.

Die Parameter zum Quellenformat können mit WinRtfLoadInsert kombiniert werden, um in einen bestehenden Text den angegebenen Text einzufügen.

Der Text ersetzt den Text, der mit der Markierung (Range) selektiert ist. Ist keine Markierung vorhanden, wird der Text an der aktuellen Cursorposition eingefügt.

Bei der Kombination mit der Option WinRtfLoadMix werden beim Laden des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.

Als Rückgabewert können neben den Fehlerkonstanten aus dem Bereich der externe Dateien die Werte ErrData und ErrRtfSyntaxError zurückgegeben werden. Bei der Rückgabe von ErrOk ist kein Fehler aufgetreten.

obj -> WinRtfSave(int1[, int2[,
int3]]) : int



Text aus RtfEdit-Objekt speichern

obj Objekt (RtfEdit-Objekt)

Ziel des Textes

WinStreamCaption Text in die
Eigenschaft
Caption
schreiben

int1 WinStreamBufField Text in
einen
Feldpuffer
schreiben

WinStreamBufText Text in
einen
Textpuffer
schreiben

Modus (optional)

WinRtfSaveAuto Textformat
automatisch
erkennen

int2 WinRtfSaveRtf RTF-Text
sichern

WinRtfSaveAscii ASCII-Text
sichern

WinRtfSaveMark markierten
Text sichern

WinRtfSaveMix Text mit Daten
mischen

int3 Deskriptor des Textpuffers
(optional)

Resultat int Fehlerwert

Verwandte Befehle,

Siehe WinRtfSaveName(),

WinRtfSaveBin(), WinRtfLoad()

Mit diesem Befehl wird ein Text aus einem RtfEdit-Objekt gespeichert. Der Deskriptor des Objektes wird in (obj) übergeben.

In (int1) wird das Ziel des Textes definiert. Der Deskriptor des Ziels wird in (int3) angegeben werden.

Folgende Ziele können angegeben werden:

- **WinStreamCaption**

Werden prozedural Änderungen im Text vorgenommen, kann mit dieser Option eine Aktualisierung des Objekts durchgeführt werden. Der Parameter (int3) darf nicht gesetzt werden.

- **WinStreamBufField**

Der Text wird in den in der Eigenschaft DbFieldName angegebenen Feldpuffer gespeichert. Der Parameter (int3) darf nicht gesetzt sein.

- **WinStreamBufText**

Der Text wird in den in (int3) übergebenen Textpuffer geschrieben. Ist kein Textpuffer angegeben, wird der in der Eigenschaft DbTextBuf angegebene Textpuffer verwendet.

Der Parameter (int2) bestimmt das zu schreibende Format. Folgende Konstanten können übergeben werden:

- **WinRtfSaveAuto**

Das Format des Textes wird automatisch bestimmt.

- **WinRtfSaveRtf**

Der Text wird im RTF-Format geschrieben.


- **WinRtfSaveAscii**

Der Text wird im ASCII-Format geschrieben.

Die Parameter zum Zielformat können mit WinRtfSaveMark kombiniert werden, um einen markierten Textbereich innerhalb des Textes zu sichern.

Bei der Kombination mit der Option WinRtfSaveMix werden beim Speichern des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.

Als Rückgabewert können neben den Fehlerkonstanten aus dem Bereich der externe Dateien die Werte ErrData und ErrRtfSyntaxError zurückgegeben werden. Bei der Rückgabe von ErrOk ist kein Fehler aufgetreten.

obj -> WinRtfSaveBin(int1, int2[, int3[, alpha4]]) : int 

Text aus RtfEdit-Objekt in binärem Objekt speichern

obj Objekt (RtfEdit-Objekt)

int1 Deskriptor des binären
Objektes

Modus (optional)

_WinRtfSaveAuto Textformat
automatisch
erkennen

_WinRtfSaveRtf RTF-Text
sichern

int2 _WinRtfSaveAscii ASCII-Text
sichern

_WinRtfSaveMark markierten
Text sichern

_WinRtfSaveMix Text mit
Daten
mischen

int3 Kompressionsfaktor (optional)

alpha4 Verschlüsselungscode
(optional)

Resultat int Fehlerwert

Verwandte Befehle,

Siehe WinRtfSave(), WinRtfLoadBin(),
BinOpen()

Mit diesem Befehl wird ein Text aus einem RtfEdit-Objekt in einem binären Objekt gespeichert. Der Deskriptor des RtfEdit-Objektes wird in (obj), der Deskriptor des binären Objekts in (int1) übergeben.

Der Parameter (int2) bestimmt das zu schreibende Format.Folgende Konstanten können angegeben werden:

- **_WinRtfSaveAuto**

Das Format des Textes wird automatisch bestimmt.

- **_WinRtfSaveRtf**

Der Text wird im RTF-Format geschrieben.

- **_WinRtfSaveAscii**

Der Text wird im ASCII-Format geschrieben.

Die Parameter zum Zielformat können mit _WinRtfSaveMark kombiniert werden, um einen markierten Teil innerhalb des Textes zu sichern.

Bei der Kombination mit der Option _WinRtfSaveMix werden beim Speichern des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.

Kontakt

Als Rückgabewert können neben den Fehlerkonstanten aus dem Bereich der externe Dateien die Werte _ErrData und _ErrRtfSyntaxError zurückgegeben werden. Bei der Rückgabe von _ErrOk ist kein Fehler aufgetreten.

obj -> WinRtfSaveName(int1,
int2, alpha3) : int
Text aus RtfEdit-Objekt speichern



obj Objekt (RtfEdit-Objekt)
 Ziel des Textes
 WinStreamNameText Text in
 einen
 internen
 Text
int1 schreiben
 WinStreamNameFile Text in
 eine
 externe
 Datei
 schreiben

 Modus
 WinRtfSaveAuto Textformat
 automatisch
 erkennen
 WinRtfSaveRtf RTF-Text
 sichern
int2 WinRtfSaveAscii ASCII-Text
 sichern
 WinRtfSaveOem OEM-Text
 sichern
 WinRtfSaveMark markierten
 Text sichern
 WinRtfSaveMix Text mit Daten
 mischen

alpha3 Zielname

Resultat int Fehlerwert

Verwandte Befehle, WinRtfSave(),

Siehe WinRtfSaveBin(),

WinRtfLoadName()

Mit diesem Befehl wird ein Text aus einem RtfEdit-Objekt gespeichert. Der Deskriptor des Objektes wird in (obj) übergeben.

In (int1) wird das Ziel des Textes definiert. Der Name des Ziels wird in (alpha3) angegeben.

Folgende Ziele können angegeben werden:

- **WinStreamNameText**

Der Text wird in einem internen Text gespeichert. Der Name des Textes wird in (alpha3) übergeben.

- **WinStreamNameFile**

Der Text wird in einer externen Datei gespeichert. Der Name der Datei wird in (alpha3) übergeben.

Der Parameter (int2) bestimmt das zu schreibende Format. Folgende Konstanten können übergeben werden:

- **_WinRtfSaveAuto**

Der Text wird im RTF-Format geschrieben.

- **_WinRtfSaveRtf**

Der Text wird im RTF-Format geschrieben.

- **_WinRtfSaveAscii**

Der Text wird im ASCII-Format geschrieben.

- **_WinRtfSaveOem**

Der Text wird im OEM-Format geschrieben.

Die Parameter zum Zielformat können mit _WinRtfSaveMark kombiniert werden, um einen markierten Teil innerhalb des Textes zu sichern.

Bei der Kombination mit der Option _WinRtfSaveMix werden beim Speichern des Textes die Platzhalter durch die entsprechenden Daten ersetzt. Weitere Informationen befinden sich im Abschnitt Text und Daten mischen.

Als Rückgabewert können neben den Fehlerkonstanten aus dem Bereich der externe Dateien die Werte _ErrData und _ErrRtfSyntaxError zurückgegeben werden. Bei der Rückgabe von _ErrOk ist kein Fehler aufgetreten.

obj -> WinRtfSearch(alpha1[, int2[, range3[, alpha4[, var
alpha5]]]]) : int



Zeichenfolge in einem Text suchen / Suchen und Ersetzen

obj Deskriptor eines RtfEdit-Objekts

alpha1 Suchtext

Optionen (optional)

WinRtfSearchUp Suchbereich vom
Ende nach Vorne
durchsuchen

WinRtfSearchCase Groß-/Kleinschreibung
beachten

int2 WinRtfSearchWord Nur ganze Wörter
suchen

WinRtfSearchReplace Suchbegriff durch
(alpha4) ersetzen

WinRtfSearchDelete Bereich in (range3)
entfernen

range3 Suchbereich (optional)

alpha4 Ersetzungstext (optional)

var alpha5 Auf Suchtext folgende Zeichenkette
(optional)

Resultat int Position der gefundenen Zeichenfolge

Siehe Verwandte Befehle, PrtRtfSearch(),
TextSearch()

Diese Funktion durchsucht einen Text in einem RtfEdit-Objekt. Der Deskriptor des Objektes wird in (obj) angegeben.

Die zu suchende Zeichenkette wird in (alpha1) angegeben. Alle weiteren Parameter sind optional. Werden keine weiteren Parameter angegeben, wird der gesamte Text von vorne nach hinten nach der Zeichenkette durchsucht. Wird der Begriff gefunden, wird die Position des ersten Zeichens innerhalb des Textes zurückgegeben. Befindet sich die zu suchende Zeichenkette mehrfach im Text, wird nur das erste Vorkommen zurückgegeben. Ist die Zeichenkette nicht vorhanden wird der Wert -1 zurückgegeben.

Beispiel:

```
tPos # $RtfEdit->WinRtfSearch('suche');
```

Die Suche kann mit folgenden Optionen beeinflusst werden:

- **WinRtfSearchUp**

Der Suchbereich wird vom Ende zum Anfang durchsucht.

- **WinRtfSearchCase**

Die Groß- und Kleinschreibung des Suchbegriffes wird beachtet.

- **WinRtfSearchWord**

Es wird nur nach ganzen Wörtern gesucht.

- **WinRtfSearchReplace**

Der Suchtext wird durch den Ersetzungstext in (alpha4) ersetzt.

- **WinRtfSearchDelete**

Der Bereich in (range3) wird aus dem RTF-Text entfernt.

Die Optionen können miteinander kombiniert werden. Der zu durchsuchende Text kann durch die Angabe eines Bereiches in (range3) bestimmt werden. Standardmäßig wird der Bereich (0, -1) (Anfang bis Ende) durchsucht.

Wird die Option WinRtfSearchReplace angegeben, muss in (alpha4) ein entsprechender Ersetzungstext angegeben werden, da sonst der Suchbegriff aus dem Text entfernt wird. Wie beim Suchen wird nur die erste Fundstelle ersetzt.

Wird die Option WinRtfSearchDelete angegeben, kann zusätzlich mit der Option WinRtfSearchReplace der Bereich durch einen anderen Text ersetzt werden. Der entsprechende Ersetzungstext wird in (alpha4) angegeben.

Wird der optionale var-Parameter (alpha5) angegeben, dann wird in dieser Variable der Text hinterlegt, der hinter dem Suchtext (alpha1) steht. Wird der Suchtext nicht gefunden, ist die Variable nach dem Aufruf leer. Die Länge des Nachfolgenden Textes richtet sich nach der Dimension der Variable. Bei einem alpha(10) beispielsweise, werden maximal 10 Zeichen zurückgegeben.

Beispiele:

```
// Suchen nach ganzem Wort mit Groß-/KleinschreibungtPos # $RtfEdit->WinRtfSearch('Suche', _WinRt
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

ErrValueInvalid Optionen (int2) ungültig

obj -> WinRtfTabGet(int1[, var
rtftab2]) : int



vorhandene Tabulatoren ermitteln

obj Deskriptor eines RtfEdit-Objekts
Bereich

int1 _WinEditAll gesamtes Dokument
 _WinEditMark markierter Bereich

var rtftab2 Array mit Tabulatoren

Resultat int Anzahl der definierten
Tabulatoren

Siehe Verwandte Befehle, WinRtfTabSet()

Mit diesem Befehl werden alle Tabulatoren in einem RtfEdit-Objekt ermittelt.

In (int1) wird übergeben, ob die Tabulatoren für den gesamten Text (_WinEditAll) oder für den markierten Bereich (_WinEditMark) ermittelt werden.


Die Tabulatoren werden in einem Array vom Typ rtftab zurückgegeben.

Die Tabulatoren befinden sich in der Reihenfolge ihrer Position in dem Array. Die Anzahl der definierten Tabulatoren wird als Rückgabewert zurückgegeben.

Beispiele:

```
local{ tTab : rtftab[64];}...// Tabulatoren auslesenif ($rtfEdit->WinRtfTabGet(_WinEditAll, var
```

obj ->

WinRtfTabSet(int1[) : int

var rtftab2]) : int

Tabulatoren setzen

obj Deskriptor eines
 RtfEdit-Objekts

Bereich

int1 _WinEditAll gesamtes
 Dokument

_WinEditMark markierter
 Bereich

var Array mit Tabulatoren

rtftab2 (optional)

Resultat int _ErrOk

Siehe Verwandte Befehle,
 WinRtfTabGet()

Mit diesem Befehl können bis zu 64 Tabulatoren in einem RtfEdit-Objekt gesetzt werden.

In (int1) wird übergeben, ob die Tabulatoren für den gesamten Text (_WinEditAll) oder für den markierten Bereich (_WinEditMark) gesetzt werden.

Die Tabulatoren werden in einem Array vom Typ rtftab übergeben und können mit dem Befehl RtfTabMake() definiert werden. Werden keine Tabulatoren übergeben, werden alle bestehenden Tabulatoren gelöscht.

Die Tabulatoren müssen in der Reihenfolge ihrer Position in dem Array angegeben sein.

Beispiele:

```
local{ tTab : rtftab[64];}...// Tabulatoren definieren
tTab[1]:tabpos # PrtUnitLog(2.0, _PrtUnitLog(2.0, _PrtUnitLog(2.0, ...))
```

Mit dem Befehl WinRtfTabSet() werden immer alle Tabulatoren übergeben. Um zu den vorhandenen Tabulatoren einen weiteren Tabulator zu setzen, müssen zunächst mit dem Befehl WinRtfTabGet() die vorhandenen Tabulatoren ermittelt und anschließend um den neuen Tabulator erweitert werden.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.

_ErrFldType Das übergebene Array ist nicht vom Typ rtftab.

obj -> WinRtfPicInsertMem(handle1[,
int2[, int3]]) : int



Einfügen von Bildern in ein RtfEdit-Objekt

obj Deskriptor des RtfEdit-Objekts

handle1 Deskriptor des Memory-Objekts


int2 Seitennummer (optional)

Optionen (optional)

WinRtfPicModeSpeed Performantes
Einfügen

WinRtfPicModeQuality Qualitatives
Einfügen

int3 WinRtfPicModeAuto Modus
abhängig von
der Farbtiefe
und Größe
des Bildes
auswählen

Resultat int Einfügeresultat (siehe Text) 

Siehe Verwandte Befehle,
WinRtfPicInsertName()

Diese Funktion fügt ein Bild aus einem Memory-Objekt (handle1) an der aktuellen Cursorposition eines RtfEdit-Objekts (obj) ein. Ist eine Selektion vorhanden, wird diese durch das Bild ersetzt. Das Argument Seitennummer (int2) bestimmt bei einem Multipage-TIFF, die Seite, die das einzufügende Bild enthält. Die Seitenzählung beginnt mit 1. Wird (int2) nicht angegeben oder ist der Wert Null, dann wird immer die erste Seite gewählt. Bei Formaten außer TIFF wird das Argument ignoriert.

Optional können in (int3) folgende Optionen angegeben werden:

WinRtfPicModeSpeed Performantes Einfügen

WinRtfPicModeQuality Qualitatives Einfügen

WinRtfPicModeAuto Modus abhängig von der Farbtiefe und Größe des Bildes
auswählen

Wird keine der Optionen angegeben, wird automatisch WinRtfPicModeAuto verwendet.

Die anzeigbaren Formate sind GIF, TIFF, JPEG, PNG und BMP.



Transparente oder semitransparente Pixel bei PNG bzw. 32bpp BMP-Dateien werden zu weißer Farbe gemischt.

Wird das Bildformat nicht unterstützt, gibt die Funktion den Fehlerwert ErrGeneric zurück. Ist der Arbeitsspeicher nicht ausreichend, wird ErrOutOfMemory zurückgegeben.

Beispiel:

```
// Bild ladentMem # MemAllocate(_MemAutoSize);tFsi # FsiOpen(_Sys->spPathMyPictures + '\MyPicture
```

Mögliche Laufzeitfehler:

Kontakt

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein RtfEdit-Objekt

obj -> WinRtfPicInsertName(alpha1[,
alpha2[, int3[, int4]]]) : int



Einfügen von Bildern in ein RtfEdit-Objekt

obj Deskriptor des RtfEdit-Objekts
alpha1 Name der Bilddatei
alpha2 Verschlüsselungs-Code (optional)
int3 Seitennummer (optional)
 Optionen (optional)
 WinRtfPicModeSpeed Performantes
 Einfügen
 WinRtfPicModeQuality Qualitatives
 Einfügen
int4 WinRtfPicModeAuto Modus
 abhängig von
 der Farbtiefe
 und Größe
 des Bildes
 auswählen

Resultat int Einfügeresultat (siehe Text)

Siehe Verwandte Befehle,
WinRtfPicInsertMem()

Diese Funktion fügt ein Bild an der aktuellen Cursorposition eines RtfEdit-Objekts ein. Ist eine Selektion vorhanden, wird diese durch das Bild ersetzt.

Durch die Voranstellung eines Präfixes (alpha1) bezeichnet der Name entweder eine externe Datei, ein binäres Objekt oder ein in der Datenbank gespeichertes Bild.

Der Verschlüsselungs-Code (alpha2) wird verwendet, wenn das Bild in der Datenbank verschlüsselt vorliegt. Das Argument Seitennummer (int3) bestimmt bei einem Multipage-Tiff, die Seite, die das einzufügende Bild enthält. Die Seitenzählung beginnt mit 1. Wird (int3) nicht angegeben oder ist der Wert Null, dann wird immer die erste Seite gewählt. Bei Formaten außer TIFF wird das Argument ignoriert.

Optional können in (int4) folgende Optionen angegeben werden:

WinRtfPicModeSpeed Performantes Einfügen
WinRtfPicModeQuality Qualitatives Einfügen
WinRtfPicModeAuto Modus abhängig von der Farbtiefe und Größe des Bildes
 auswählen

Wird keine der Optionen angegeben, wird automatisch WinRtfPicModeAuto verwendet.

Die anzeigbaren Formate sind GIF, TIFF, JPEG, PNG und BMP.



Transparente oder semitransparente Pixel bei PNG bzw. 32bpp BMP-Dateien werden zu weißer Farbe gemischt.

Als Fehlerwerte können Fehlerwerte aus dem Bereich der Befehle für externe Dateien auftreten, wenn das Bild extern gelesen wird.

Zusätzlich sind folgende Fehlerwerte möglich:

<u>ErrGeneric</u>	BLOB oder externe Datei enthält kein Bild oder ein unbekanntes Bildformat
<u>ErrNameInvalid</u>	Der Parameter (alpha1) ist leer
<u>ErrUnavailable</u>	Das Bild (alpha1) existiert nicht in der Datenbank
<u>ErrOutOfMemory</u>	Arbeitsspeicher nicht ausreichend

Beispiele:

```
// Externe Datei laden $RtfEdit->WinRtfPicInsertName('*' + _Sys->spPathMyPictures + '\MyPicture.jpg')
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein RtfEdit-Objekt

RtfTabMake(int1, int2) :

rtftab



Erzeugung eines Tabulators

int1 Tabulatorposition

Tabulatortyp

_WinRtfTabNone Tabulator löschen

_WinRtfTabLeft linksbündigen
Tabulator setzen

int2 _WinRtfTabCenter zentrierten
Tabulator setzen

_WinRtfTabRight rechtsbündigen
Tabulator setzen


_WinRtfTabDecimal Dezimal-Tabulator
setzen

Resultat rtftab Erzeugter Tabulator

Siehe Verwandte Befehle, rtftab

Dieser Befehl erzeugt aus den Angaben des Tabulatortyps und der Tabulatorposition eine rtftab-Struktur.

Die Tabulatorposition wird in logischen Einheiten angegeben.

obj -> WinEmfProcess(int1, alpha2[, int3[,  int4[, int5]]) : int

Text aus RtfEdit-Objekt als Bild speichern

obj RtfEdit-Objekt

Funktion

int1 _WinEmfProcessCreate Bild erzeugen

_WinEmfProcessDelete Bild löschen

alpha2 Name des Bildes oder der externen Datei

Optionen (optional)

int3 _WinEmfCreateOverwrite Zielfile oder -objekt
überschreiben

_WinEmfCreateDefault Zielfile oder -objekt
nicht überschreiben

int4 Position des Startzeichens (optional)

int5 Höhe des Bildes in Twips (optional)

Resultat int Position des ersten, nicht geschriebenen
Zeichens, 0 oder Fehlerwert 

Siehe Verwandte Befehle

Der Befehl erstellt aus dem Inhalt eines RtfEdit-Objekts ein EMF-Bild. Das Bild kann als Storage-Objekt in der Datenbank oder als externe Datei gespeichert werden.

In (obj) wird der Deskriptor des RtfEdit-Objekts übergeben. Die durchzuführende Funktion ist in (int1) angegeben. Folgende Funktionen können durchgeführt werden:

• _WinEmfProcessCreate (0)

Diese Funktion erzeugt das Bild. Der Name des erzeugten Objekts wird in (alpha2) angegeben. Wird dem Namen ein '*' vorangestellt, erfolgt die Speicherung als externe Datei, sonst erfolgt die Speicherung im Storage-Verzeichnis MetaPicture. Wenn ein Storage-Objekt generiert wird, muss die Namensvergabe den Konventionen für Oberflächen-Objekte genügen (Eigenschaft Name), ansonsten den Konventionen für externe Dateinamen.

Über die Optionen (int3) kann bestimmt werden, ob ein vorhandenes Objekt bzw. eine vorhandene Datei überschrieben werden soll. Folgende Konstanten stehen zur Verfügung:

_WinEmfCreateDefault (0) Bestehende externe Datei oder Bild-Objekt nicht überschreiben

_WinEmfCreateOverwrite (1) Bestehende externe Datei oder Bild-Objekt überschreiben

Das Argument (int4) gibt an, ab welcher Zeichenposition des enthaltenen Textes die EMF-Ausgabe erfolgt. Fehlt das Argument oder ist es 0, wird am Textanfang begonnen.

(int5) definiert die Bildhöhe in logischen Einheiten, bis zu der der Text ausgegeben wird. Es werden alle Zeichen geschrieben, die vollständig dargestellt werden können. Der Befehl gibt anschließend das nächste Zeichen zurück, mit dem das nächste Bild startet. Wurde der gesamte Text formatiert,

liefert der Befehl 0 zurück. Auf diese Weise kann der Text sukzessive in mehrere Bilder aufgeteilt werden. Wird die Bildhöhe nicht angegeben oder ist diese -1, wird der Text ab der Startposition bis zum Ende ausgegeben.

Die Breite des Bildes wird aus der Eigenschaft PageWidth des RtfEdit-Objekts entnommen.

• **_WinEmfProcessDelete (1)**

Diese Funktion entfernt ein als Storage-Objekt erzeugtes Bild aus dem Verzeichnis MetaPicture. Das übergeben Objekt, sowie die Parameter (int3) bis (int5) werden ignoriert. Externe Dateien können mit der Anweisung FsiDelete() gelöscht werden.

Tritt beim Erzeugen des Bildes ein Fehler bei der Verarbeitung auf, wird ein negativer Wert zurückgegeben. Beim Löschen des Bildes wird im Fehlerfall ein Wert ungleich 0 zurückgegeben. Dieser Wert kann mit den folgenden Konstanten verglichen werden:

<u>_ErrNameInvalid</u>	Der übergebene Name ist leer oder ungültig.
<u>_ErrExists</u>	Die Datei oder das <u>Storage</u> -Objekt existiert bereits und die Option <u>_WinEmfCreateOverwrite</u> wurde nicht angegeben.
<u>_ErrFsi...</u>	Beim Schreiben in eine externe Datei können Fehler aus diesem Bereich auftreten.
<u>_ErrOutOfMemory</u>	Es steht nicht genügend Speicher zur Durchführung der Operation zur Verfügung.
<u>_rNoKey</u>	Das mit <u>_WinEmfProcessDelete</u> zu löschende Bild existiert nicht.

Beispiel:

Folgendes Beispiel bietet eine einfache Möglichkeit EMF-Seiten für die Anzeige in einer Druckvorschau aus dem aktuellen Inhalt eines RtfEdit-Objekts zu generieren.

```
sub MakeEmfPages( aRtfEdit : handle;           // Deskriptor des RtfEdit  aPath      : alpha(250); // P
```

Beim RtfEdit-Objekt muss die Eigenschaft PrtDevice gesetzt sein, bevor der Befehl durchgeführt wird. Im anderen Fall wird der Laufzeitfehler _ErrHdlInvalid generiert.

Die erzeugten EMF-Dateien können vom PrtMetaPicture-Objekt angezeigt werden.

Mögliche Laufzeitfehler:

<u>_ErrValueInvalid</u>	In (int2) ist eine ungültige Option angegeben.
<u>_ErrHdlInvalid</u>	In (obj) wurde kein Deskriptor eines <u>RtfEdit</u> oder in dem angegebenen Objekt ist kein <u>PrintDevice</u> angegeben.

Konstanten für Rtf-Befehle
Konstanten für Rtf-Befehle
Siehe RTF-Befehle

- WinRtfLoadAscii
- WinRtfLoadAuto
- WinRtfLoadInsert
- WinRtfLoadMix
- WinRtfLoadOem
- WinRtfLoadRtf
- WinRtfPicModeAuto
- WinRtfPicModeQuality
- WinRtfPicModeSpeed
- WinRtfSaveAscii
- WinRtfSaveAuto
- WinRtfSaveMark
- WinRtfSaveMix
- WinRtfSaveOem
- WinRtfSaveRtf
- WinRtfSearchCase
- WinRtfSearchDelete
- WinRtfSearchReplace
- WinRtfSearchUp
- WinRtfSearchWord
- WinRtfTabCenter
- WinRtfTabDecimal
- WinRtfTabLeft
- WinRtfTabNone
- WinRtfTabRight
- WinStreamBufField
- WinStreamBufText
- WinStreamCaption
- WinStreamNameBin
- WinStreamNameFile
- WinStreamNameText

WinRtfLoadAscii
Text im ASCII-Format laden
Wert 2 / 0x0002
WinRtfLoad(),
Siehe WinRtfLoadName(),
WinRtfLoadBin()
Der Text wird im ASCII-Format geladen.

WinRtfLoadAuto
Format automatisch bestimmen
Wert 0 / 0x0000

WinRtfLoad(),
Siehe WinRtfLoadName(),
WinRtfLoadBin()

Das Format des zu ladenden Textes wird automatisch bestimmt. Handelt es sich nicht um einen RTF-Text, wird der Text im ASCII-Format gelesen.

WinRtfLoadInsert

Text einfügen

Wert 16 / 0x0010

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfLoadBin()

Der zu ladende Text wird in einen bestehenden Text eingefügt.

WinRtfLoadMix
Text mit Daten beim Laden mischen
Wert 8 / 0x0008

WinRtfLoad(),
Siehe WinRtfLoadName(),
WinRtfLoadBin()

Beim Laden des Textes werden die enthaltenen Platzhalter durch den entsprechenden Inhalt ersetzt. Die Platzhalter sind mit Markierungszeichen geklammert, die in der Eigenschaft RtfMixMarker des App-Objektes definiert werden kann.

Weitere Hinweise befinden sich im Abschnitt Text und Daten mischen.

WinRtfLoadOem
Text im OEM-Format laden
Wert 4 / 0x0004
Siehe WinRtfLoad(),
WinRtfLoadName()
Der Text wird im OEM-Format geladen.

WinRtfLoadRtf
Text im RTF-Format laden
Wert 1 / 0x0001
WinRtfLoad(),
Siehe WinRtfLoadName(),
WinRtfLoadBin()
Der Text wird im RTF-Format geladen.

WinRtfPicModeAuto

Qualität des Bildimports abhängig von der Farbtiefe und Größe des Bildes

Wert 0

WinRtfPicInsertMem(),

Siehe WinRtfPicInsertName(),

WinRtfPicModeSpeed,

WinRtfPicModeQuality

Option bei WinRtfPicInsertMem() und WinRtfPicInsertName(). Beim Einfügen eines Bildes in ein RtfEdit-Objekt wird abhängig von der Größe und der Farbtiefe des Bildes automatisch entweder WinRtfPicModeSpeed oder WinRtfPicModeQuality verwendet.

_WinRtfPicModeQuality
Bildimport auf Qualität optimiert
Wert 2

Siehe WinRtfPicInsertMem(),
WinRtfPicInsertName(),
WinRtfPicModeSpeed,
WinRtfPicModeAuto

Option bei WinRtfPicInsertMem() und WinRtfPicInsertName(). Das Einfügen großer Bilder kann sehr lange dauern. Dafür bleibt die Qualität des Bildes auch beim Ändern der Größe im RtfEdit-Objekt erhalten. Wird ein Rtf-Text mit dieser Option gespeichert, wird die resultierende Datei größer als mit der Option WinRtfPicModeSpeed.

WinRtfPicModeSpeed
Bildimport auf Performance optimiert
Wert 1

Siehe WinRtfPicInsertMem(),
WinRtfPicInsertName(),
WinRtfPicModeQuality,
WinRtfPicModeAuto

Option bei WinRtfPicInsertMem() und WinRtfPicInsertName(). Das Einfügen des Bildes in ein RtfEdit-Objekt ist performant, auch für große Bilder. Insbesondere beim Verkleinern des Bildes kann es jedoch zu Verlusten bei der Darstellung kommen. Beim Speichern des Rtf-Textes ist die resultierende Datei kleiner als mit der Option WinRtfPicModeQuality.

WinRtfSaveAscii
Speichern im ASCII-Format
Wert 2 / 0x0002

WinRtfSave(),
Siehe WinRtfSaveName(),
WinRtfSaveBin()

Der Text wird im ASCII-Format gespeichert. Die Formatierungen gehen dabei verloren.

WinRtfSaveAuto
Format automatisch bestimmen
Wert 0 / 0x0000

WinRtfSave(),
Siehe WinRtfSaveName(),
WinRtfSaveBin()

Das Format wird automatisch bestimmt. In der Regel wird der Text im RTF-Format gespeichert.

_WinRtfSaveMark
Markierung speichern
Wert 16 / 0x0010

WinRtfSave(),
Siehe WinRtfSaveName(),
WinRtfSaveBin()
Der markierte Bereich wird gespeichert.

_WinRtfSaveMix
Text mit Daten beim Laden mischen
Wert 8 / 0x0008

WinRtfSave(),
Siehe WinRtfSaveName(),
WinRtfSaveBin()

Beim Speichern des Textes werden die enthaltenen Platzhalter durch den entsprechenden Inhalt ersetzt. Die Platzhalter sind mit Markierungszeichen geklammert, die in der Eigenschaft RtfMixMarker des _App-Objektes definiert werden können.

Weitere Hinweise befinden sich im Abschnitt Text und Daten mischen.

_WinRtfSaveOEM
Speichern im OEM-Format
Wert 4 / 0x0004

Siehe WinRtfSave(),
WinRtfSaveName()

Der Text wird im OEM-Format gespeichert. Die Formatierungen gehen dabei verloren.

WinRtfSaveRtf
Speichern im RTF-Format
Wert 1 / 0x0001

WinRtfSave(),
Siehe WinRtfSaveName(),
WinRtfSaveBin()
Der Text wird im RTF-Format gespeichert.

WinRtfSearchCase
Option beim Durchsuchen eines RTF-Textes
Wert 2 / 0x0002
Siehe WinRtfSearch()
Option beim Befehl WinRtfSearch().

Mit dieser Option wird die Groß- und Kleinschreibung im Suchtext berücksichtigt. Der Suchtext muss in der gleichen Schreibweise im Text vorkommen.

_WinRtfSearchDelete

Option beim Durchsuchen eines RTF-Textes

Wert 16 / 0x0010

Siehe WinRtfSearch()

Option beim Befehl WinRtfSearch(). Bei der Angabe dieser Option wird der in Range angegebene Text-Bereich ausgeschnitten.

_WinRtfSearchReplace

Option beim Durchsuchen eines RTF-Textes

Wert 8 / 0x0008

Siehe WinRtfSearch()

Option beim Befehl WinRtfSearch(). Bei der Angabe dieser Option wird der Suchtext durch einen anderen anzugebenden Text ersetzt. Der entsprechende Text wird beim Befehl WinRtfSearch() angegeben.

WinRtfSearchUp
Option beim Durchsuchen eines RTF-Textes
Wert 1 / 0x0001
Siehe WinRtfSearch()
Option beim Befehl WinRtfSearch().

Mit dieser Option wird die Suchrichtung bestimmt. Wird die Option nicht angegeben, wird der Text vom Beginn zum Ende durchsucht. Mit der Option erfolgt die Suche vom Ende des Textes zum Anfang.

WinRtfSearchWord
Option beim Durchsuchen eines RTF-Textes
Wert 4 / 0x0004
Siehe WinRtfSearch()
Option beim Befehl WinRtfSearch().

Mit dieser Option wird nur nach ganzen Wörtern gesucht. Wörter werden durch Leerzeichen, Satzzeichen oder Bindestriche und Ähnlichem voneinander getrennt.

_WinRtfTabCenter
zentrierten Tabulator setzen
Wert 2

Siehe RtfTabMake()

Es wird ein zentrierter Tabulator angelegt. Der Text wird zum Tabulator zentriert dargestellt.

_WinRtfTabDecimal
Dezimal-Tabulator setzen
Wert 4

Siehe RtfTabMake()

Es wird ein Dezimal-Tabulator angelegt. Der Text wird zum Dezimalpunkt (kein Komma) zentriert.

_WinRtfTabLeft

linksbündigen Tabulator setzen

Wert 1

Siehe RtfTabMake()

Es wird ein linksbündiger Tabulator angelegt. Der Text wird linksbündig zum Tabulator dargestellt.

_WinRtfTabNone

bestehenden Tabulator löschen

Wert 0

Siehe RtfTabMake()

Der bestehende Tabulator an der angegebenen Position wird gelöscht.

_WinRtfTabRight

rechtsbündigen Tabulator setzen

Wert 3

Siehe RtfTabMake()

Es wird ein rechtsbündiger Tabulator angelegt. Der Text wird rechtsbündig zum Tabulator dargestellt.

_WinStreamBufField

Quelle des Textes

Wert 2

StreamSource,

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfSave(),

WinRtfSaveName()

Wird die Eigenschaft StreamSource auf den Wert _WinStreamBufField gesetzt, wird der Inhalt des in DbFieldName angegebenen Datenbankfeldes angezeigt.

Die Konstante wird ebenfalls verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad() bzw. WinRtfSave() anzugeben.

_WinStreamBufText

Quelle des Textes

Wert 3

Siehe WinRtfLoad(),
WinRtfSave()

Die Konstante wird verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad() bzw. WinRtfSave() anzugeben.

_WinStreamCaption

Quelle des Textes

Wert 1

StreamSource,

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfSave(),

WinRtfSaveName()

Wird die Eigenschaft StreamSource auf den Wert _WinStreamCaption gesetzt, wird der in der Eigenschaft Caption vorhandene Text dargestellt.

Die Konstante wird ebenfalls verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad() / WinRtfLoadName() bzw. WinRtfSave() / WinRtfSaveName() anzugeben.

_WinStreamNameBin

Quelle des Textes

Wert 6

StreamSource,

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfSave(),

WinRtfSaveName()

Wird die Eigenschaft StreamSource auf den Wert _WinStreamNameBin gesetzt, wird das in der Eigenschaft FileName binäre Objekt dargestellt.

Die Konstante wird ebenfalls verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad() / WinRtfLoadName() bzw. WinRtfSave() / WinRtfSaveName() anzugeben.

_WinStreamNameFile

Quelle des Textes

Wert 5

StreamSource,

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfSave(),

WinRtfSaveName()

Wird die Eigenschaft StreamSource auf den Wert _WinStreamNameFile gesetzt, wird der Inhalt der in der Eigenschaft FileName angegebenen externen Datei dargestellt.

Die Konstante wird ebenfalls verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad(), WinRtfLoadName(), WinRtfSave(), WinRtfSaveName(), WinDocLoadName() und WinDocSaveName() anzugeben.

_WinStreamNameText

Quelle des Textes

Wert 4

StreamSource,

WinRtfLoad(),

Siehe WinRtfLoadName(),

WinRtfSave(),

WinRtfSaveName()

Wird die Eigenschaft StreamSource auf den Wert _WinStreamNameText gesetzt, wird der in der Eigenschaft FileName angegebene interne Text dargestellt.

Die Konstante wird ebenfalls verwendet, um die Quelle bzw. das Ziel des Textes bei den Befehlen WinRtfLoad(), WinRtfLoadName(), WinRtfSave(), WinRtfSaveName(), WinDocLoadName() und WinDocSaveName() anzugeben.

Theme-Befehle

Befehle zum Bearbeiten eines Theme-Objekts

Verwandte

Siehe Befehle, Theme,
Befehlsgruppen,
Befehlsliste

Befehle

- WinThemeClose
- WinThemeDelete
- WinThemeOpen
- WinThemeSetDelete
- WinThemeSetOpen

obj ->

WinThemeClose([int1[,
alpha2]]) : int



Theme schließen

obj Deskriptor des Theme-Objektes

int1 Optionen (optional)

alpha2 Neuer Name (optional)

Fehlerwert

ErrOk Befehl wurde
erfolgreich
durchgeführt.

ErrNameInvalid Der Name
entspricht
nicht den
gültigen
Konventionen.

Resultat int



-70 Theme-Objekt
(obj) nicht
gesperrt.

Siehe Verwandte Befehle, Theme

Der Befehl entsperrt das Theme-Objekt, sofern dies durch die Option WinThemeOpenLock zuvor gesperrt wurde, und schließt es. Der Deskriptor ist nach dem Aufruf nicht mehr gültig.

Im Parameter (obj) wird der Deskriptor auf ein, mit WinThemeOpen() geöffnetes Theme-Objekt angegeben.

Folgende Optionen können im Parameter (int1) angegeben werden:

- **WinThemeCloseSave**

Das Theme wird gespeichert. Alternativ kann im Argument (alpha2) auch ein abweichender Name angegeben werden. Dies ist auch dann notwendig, wenn ein vordefiniertes Theme geöffnet und geändert wurde und nun in der Datenbank gespeichert werden soll.

Im Argument (alpha2) kann ein Name für das Thema angegeben werden, wenn die Option WinThemeCloseSave verwendet wird. Falls bereits ein Theme unter dem Name existiert, wird es überschrieben.

Von der Funktion können folgende Fehlerwerte zurückgegeben werden:

ErrOk Befehl wurde erfolgreich durchgeführt.


ErrNameInvalid Der Name entspricht nicht den gültigen Konventionen.

-70 Theme-Objekt (obj) nicht gesperrt.

Mögliche Laufzeitfehler

ErrHdlInvalid Der Deskriptor (obj) ist ungültig.

ErrValueInvalid Es wurde eine ungültige Option (int1) angegeben.

WinThemeDelete(alpha1) : 

int

Theme löschen

alpha1 Name des Themes

Fehlerwert

ErrOk

Befehl wurde
erfolgreich
durchgeführt.

ErrNameInvalid

Der Name
(alpha1)
entspricht
nicht den
gültigen
Konventionen.

Resultat int



ErrLocked

Das Theme
(alpha1) ist
gesperrt.

ErrUnavailable

Das Theme
(alpha1)
existiert nicht.

Siehe Verwandte Befehle, Theme

Der Befehl löscht ein in der Datenbank gespeichertes Theme. Der Name des zu löschenden Themes wird in (alpha1) angegeben.

Folgende Fehlerwerte können von der Funktion zurückgegeben werden:

ErrOk

Befehl wurde erfolgreich durchgeführt.

ErrNameInvalid

Der Name (alpha1) entspricht nicht den gültigen Konventionen.

ErrLocked

Das Theme (alpha1) ist gesperrt.

ErrUnavailable

Das Theme (alpha1) existiert nicht.

WinThemeOpen([alpha1[,
int2]]) : handle



Theme öffnen

alpha1 Name des Themes (optional)

Optionen (optional)

0 Theme mit dem Namen (alpha1) öffnen

_WinThemeOpenLock Theme exklusiv sperren

_WinThemeOpenFirst Erstes Theme öffnen

int2 _WinThemeOpenLast Letztes Theme öffnen

_WinThemeOpenNext Nächstes Theme, ausgehend vom Namen (alpha1) öffnen

_WinThemeOpenPrev Vorhergehendes Theme, ausgehend vom Namen (alpha1) öffnen

Deskriptor des Theme-Objektes oder Fehlerwert

> 0 Deskriptor des Theme-Objektes

_ErrNameInvalid Es handelt sich nicht um ein vordefiniertes Theme oder der Theme-Name entspricht nicht den gültigen Konventionen.

Resultat handle

_ErrUnavailable Theme existiert nicht.

_ErrLocked Das Theme wurde bereits von einem anderen Client gesperrt.

Siehe Verwandte Befehle, Theme, Blog

Der Befehl öffnet ein vordefiniertes oder ein in der Datenbank gespeichertes Theme und gibt einen Deskriptor vom Typ _HdlTheme zurück.

Der Name des zu ladenden Themes wird in (alpha1) angegeben. Es können Namen von vordefinierten oder in der Datenbank gespeicherten Themes angegeben werden:

- Beginnt der Name mit einem Unterstrich _, wird ein vordefiniertes Theme geöffnet. Es existieren drei vordefinierte Themes: '_OfficeBlue', '_OfficeDark' und '_WindowsColor'.
- Ist der Name eine leere Zeichenkette, dann wird das Theme '_OfficeBlue' geöffnet.
- Bei der Angabe anderer Namen, wird das Theme aus der Datenbank geladen.

Im Argument (int2) können folgende Optionen angegeben werden:

- **0**

Das durch (alpha1) angegebene Theme wird geladen. Falls kein Theme mit diesem Name existiert, wird der Wert _ErrUnavailable zurückgegeben.

- **_WinThemeOpenLock**

Lädt das Theme und sperrt es exklusiv. Dies ist die Voraussetzung, damit Theme-Eigenschaften geändert werden können. Soll ein Theme nicht geändert, sondern nur gelesen werden, kann die Option entfallen.

- **_WinThemeOpenFirst**

Lädt das erste Theme (aufsteigend nach Theme-Name) aus der Datenbank. Ist kein Theme in der Datenbank vorhanden, wird der Fehlerwert _ErrUnavailable zurückgegeben.

- **_WinThemeOpenLast**

Lädt das letzte Theme aus der Datenbank. Ist kein Theme in der Datenbank vorhanden, wird der Fehlerwert _ErrUnavailable zurückgegeben.

- **_WinThemeOpenNext**

Lädt ausgehend vom angegebenen Namen (alpha1) das nächste Theme. Ist kein nächstes Theme vorhanden, wird der Fehlerwert _ErrUnavailable zurückgegeben.

- **_WinThemeOpenPrev**

Lädt ausgehend vom angegebenen Namen (alpha1) das vorhergehende Theme. Ist kein vorhergehendes Theme vorhanden, wird der Fehlerwert _ErrUnavailable zurückgegeben.

Die Optionen _WinThemeOpenFirst, _WinThemeOpenLast, _WinThemeOpenNext und _WinThemeOpenPrev können mit der Option _WinThemeOpenLock kombiniert werden.



Ist eine Option außer 0 angegeben, bezieht sich diese auf ein Theme das in der Datenbank gespeichert ist. Die Optionen dürfen nicht für vordefinierte Themes angegeben werden.

Beim Erstellen eines Theme-Objektes wird die Eigenschaft ThemeBaseName auf das zugrundeliegende vordefinierte Theme gesetzt.

Außer einem Deskriptor auf ein Theme-Objekt können folgende Fehlerwerte von der Funktion zurückgegeben werden:

Kontakt

ErrNameInvalid Es handelt sich nicht um ein vordefiniertes Theme oder der Theme-Name entspricht nicht den gültigen Konventionen.

ErrUnavailable Theme existiert nicht.

ErrLocked Das Theme wurde bereits von einem anderen Client gesperrt.

Mögliche Laufzeitfehler

ErrValueInvalid In (int2) wurde eine unbekannte oder ungültige Option übergeben.

obj ->

WinThemeSetDelete(int1) : 

int

ThemeSet löschenobj Deskriptor des Themes

int1 Nummer des zu löschenden

ThemeSets

Optionen (optional)

0

ThemeSet
mit der
Nummer
(int1)
öffnen

int2

WinThemeSetOpenCreate Neues
ThemeSet
mit der
Nummer
(int1)
erstellen

Fehlerwert

ErrOkDas
ThemeSet
wurde
erfolgreich
gelöscht.Resultat int ErrUnavailable Das ThemeSet
mit der
Nummer
(int1)
existiert
nicht.Siehe Verwandte Befehle, Theme,
ThemeSet, WinThemeSetOpen()Dieser Befehl löscht ein vorhandenes ThemeSet-Objekt eines Theme-Objektes (obj).

In (obj) muss ein Theme-Objekt angegeben werden (siehe WinThemeOpen()). Im Argument (int1) wird eine eindeutige Nummer eines ThemeSets innerhalb des Themes angegeben.

Folgende Fehlerwerte können von der Funktion zurückgegeben werden:

ErrOk Das ThemeSet wurde erfolgreich gelöscht.ErrUnavailable Das ThemeSet mit der Nummer (int1) existiert nicht.

Damit das ThemeSet aus dem, in der Datenbank gespeicherten, Theme entfernt wird, muss das Theme mit WinThemeClose() und der Option WinThemeCloseSave gespeichert werden.

Mögliche Laufzeitfehler

Kontakt

- ErrHdlInvalid Der Deskriptor (obj) ist ungültig oder kein Deskriptor eines Theme-Objektes.
- ErrValueInvalid Die angegebene ThemeSet-Nummer (int1) ist ≤ 0 .

obj -> WinThemeSetOpen(int1[,
int2]) : handle



ThemeSet öffnen

obj	Deskriptor des <u>Themes</u>
int1	Nummer des zu erstellenden oder zu öffnenden <u>ThemeSets</u>
	Optionen (optional)
	0 ThemeSet mit der Nummer (int1) öffnen
int2	<u>WinThemeSetOpenCreate</u> Neues ThemeSet mit der Nummer (int1) erstellen
	Deskriptor des ThemeSets oder Fehlerwert
	> 0 Deskriptor des ThemeSets
	<u>ErrUnavailable</u> Das ThemeSet mit der Nummer (int1) existiert nicht.
Resultat <u>handle</u>	<u>ErrExists</u> Das ThemeSet mit der Nummer (int1) existiert bereits (bei Angabe von <u>WinThemeSetOpenCreate</u>).

Siehe Verwandte Befehle, Theme, ThemeSet, WinThemeOpen(), WinThemeSetDelete()

Dieser Befehl erstellt ein neues oder öffnet ein vorhandenes ThemeSet-Objekt eines Theme-Objektes (obj) und gibt einen Deskriptor vom Typ HdlThemeSet zurück. Das ThemeSet enthält alle im Theme definierten Theme-Elemente und deren Eigenschaften.

In (obj) muss ein Theme-Objekt angegeben werden (siehe WinThemeOpen()). Im Argument (int1) wird eine eindeutige Nummer eines ThemeSets innerhalb des Themes angegeben.

Im Argument (int2) können folgende Optionen angegeben werden:

- 0

Das durch (int1) angegebene ThemeSet wird geladen. Falls kein ThemeSet mit dieser Nummer im Theme (obj) existiert, wird der Wert ErrUnavailable zurückgegeben.

- **WinThemeSetOpenCreate**

Das durch (int1) angegebene ThemeSet wird erstellt. Falls bereits ein ThemeSet mit dieser Nummer im Theme (obj) existiert, wird der Wert ErrExists zurückgegeben.

Bei der Erstellung eines ThemeSet mit der Option WinThemeSetOpenCreate enthalten die Eigenschaften zunächst Default-Werte. Dadurch wird definiert, dass die Eigenschaften des ThemeSets dieselben Werte enthalten, wie das Theme (obj) selber.

Für die Default-Werte gibt es folgende Konstanten, die bei den Eigenschaften gesetzt oder gelesen werden können:

WinThemeIntNULL Integer-Eigenschaften

WinThemeColorNULL Farb-Eigenschaften

WinThemeFontNULL Font-Eigenschaften

Außer einem Deskriptor auf ein ThemeSet können folgende Fehlerwerte von der Funktion zurückgegeben werden:

ErrUnavailable Das ThemeSet mit der Nummer (int1) existiert nicht.

ErrExists Das ThemeSet mit der Nummer (int1) existiert bereits (bei Angabe von WinThemeSetOpenCreate).



Damit neue oder geänderte ThemeSets gespeichert werden, muss das jeweilige Theme mit WinThemeClose() und der Option WinThemeCloseSave gespeichert werden.

Beispiel:

```
// ThemeSet mit Nummer 1 anlegen und Füll- sowie Textfarbe von Button-Objekten setzenThemeSet1 #
```

Mögliche Laufzeitfehler

ErrHdlInvalid Der Deskriptor (obj) ist ungültig oder kein Deskriptor eines Theme-Objektes.

ErrValueInvalid Die angegebene ThemeSet-Nummer (int1) ist ≤ 0 oder in (int2) wurde eine unbekannte oder ungültige Option übergeben.

ErrMemExhausted Das ThemeSet konnte nicht allokiert werden (bei Angabe von WinThemeSetOpenCreate in (int2)).

Konstanten für Theme-Befehle

Konstanten für Theme-Befehle

Siehe Theme-Befehle

<u>_WinThemeCloseSave</u>	<u>Theme</u> beim Schließen speichern
<u>_WinThemeOpenFirst</u>	Erstes <u>Theme</u> öffnen
<u>_WinThemeOpenLast</u>	Letztes <u>Theme</u> öffnen
<u>_WinThemeOpenLock</u>	<u>Theme</u> exklusiv sperren
<u>_WinThemeOpenNext</u>	Nächstes <u>Theme</u> öffnen
<u>_WinThemeOpenPrev</u>	Vorheriges <u>Theme</u> öffnen
<u>_WinThemeSetOpenCreate</u>	Neues <u>ThemeSet</u> erstellen

_WinThemeCloseSave

Theme speichern

Wert 1

Siehe WinThemeClose()

Option bei WinThemeClose(), mit der das Theme beim Schließen gespeichert wird.

_WinThemeOpenFirst

Erstes Theme öffnen

Wert 16 / 0x0010

Siehe WinThemeOpen()

Option bei WinThemeOpen(), mit der das erste Theme geöffnet wird.

_WinThemeOpenLast

Letztes Theme öffnen

Wert 128 / 0x0080

Siehe WinThemeOpen()

Option bei WinThemeOpen(), mit der das letzte Theme geöffnet wird.

_WinThemeOpenLock

Theme beim Öffnen sperren

Wert 1 / 0x01

Siehe WinThemeOpen()

Option bei WinThemeOpen(), mit der das Theme beim Öffnen exklusiv gesperrt wird.

Dies ist die Voraussetzung, damit Theme-Eigenschaften geändert werden können.

_WinThemeOpenNext
Nächstes Theme öffnen
Wert 32 / 0x0020

Siehe WinThemeOpen()

Option bei WinThemeOpen(), mit der das nächste Theme geöffnet wird.

_WinThemeOpenPrev

Vorhergehendes Theme öffnen

Wert 64 / 0x0040

Siehe WinThemeOpen()

Option bei WinThemeOpen(), mit der das vorhergehende Theme geöffnet wird.

_WinThemeSetOpenCreate

Neues ThemeSet erstellen

Wert 1 / 0x0001

Siehe WinThemeSetOpen()

Option bei WinThemeSetOpen(), mit der ein neues ThemeSet angelegt wird.

TreeView-Befehle

Befehle zum Bearbeiten eines TreeView-Objekts

Verwandte

Befehle,

Siehe TreeView,


Befehlsgruppen,

Befehlsliste

Befehle

- WinTreeNodeAdd
- WinTreeNodeRemove
- WinTreeNodeSearch

obj ->

WinTreeNodeAdd([alpha1[, ,
alpha2[, int3]]]) : handle

Knoten zu Baum hinzufügen

obj Baum oder Knoten

alpha1 Knotenname
(optional)

alpha2 Knotentitel (optional)

int3 Nachfolgeknoten
(optional)

Resultat handle Knoten-Objekt

Siehe Verwandte Befehle,
TreeNode, TreeView

Mit diesem Befehl wird einem Baum-Objekt oder einem Knoten-Objekt (obj) ein neuer Knoten hinzugefügt. In (int3) kann ein Knoten der gleichen Ebene angegeben werden, der neue Knoten wird dann vor diesem Knoten in den Baum eingefädelt. Wird dieser Parameter nicht angegeben, wird der Knoten als letzter Knoten des in (obj) angegebenen Objektes angehängt. Name (alpha1) (siehe Name) und Titel (alpha2) (siehe Caption) können direkt beim Aufruf bestimmt werden, alle weiteren Eigenschaften müssen mit Hilfe des Resultats gesetzt werden.

Als Resultat wird der Deskriptor des neuen Knoten zurückgeliefert. Im Fehlerfall ist das Ergebnis 0.

Beispiel:

```
// Fügt dem Baum $TreeView einen Knoten mit dem// Namen 'node1' und dem Titel 'Node 1' hinzuHdlM
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Baum oder Knoten (obj) ungültig

obj ->
WinTreeNodeRemove([logic1])
Knoten aus Baum entfernen



obj Baum /
 Knoten
 Knoten
logic1 leeren
 (optional)
 Verwandte
Siehe Befehle,
 TreeNode,
 TreeView

Dieser Befehl entfernt ein Knoten-Objekt (obj) inklusive aller Unterknoten aus einem Baum-Objekt. Wird der optionalen Parameter (logic1) auf true gesetzt, werden nur die Unterknoten des Knotens (obj) entfernt.


In (obj) kann auch ein Baum-Objekt angegeben werden. Es werden dann alle Knoten des Baums entfernt.

Ist die Eigenschaft AutoUpdate auf false gesetzt (siehe WinUpdate()), werden die Objekte erst beim Aktivieren des Updates entfernt.

Beispiele:

```
// Knoten tHdlNode entfernen tHdlNode->WinTreeNodeRemove();// Baum leeren tHdlTree->WinTreeNodeRemove()
```

obj ->

WinTreeNodeSearch(alpha1[, int2[,  handle3]]) : handle

Knoten im Baum suchen

obj Startobjekt der Suche: TreeView- oder TreeNode-Objekt

alpha1 Suchtext

Optionen (optional)

WinTreeNodeSearchCaption Durchsucht die Caption-Eigenschaft

WinTreeNodeSearchCustom Durchsucht die Custom-Eigenschaft

WinTreeNodeSearchCI Ignoriert Groß-/Kleinschreibung

WinTreeNodeSearchLike Wildcard-Vergleich

WinTreeNodeSearchLikeAuto Wildcard-Vergleich mit automatischer Maskierung

int2 WinTreeNodeSearchToken Begriffsorientierte Suche

WinTreeNodeSearchRegEx Vergleich mit regulären Ausdrücken

WinTreeNodeSearchChildrenOnly Nur Kind-Knoten durchsuchen

WinTreeNodeSearchUp Sucht nach oben

WinTreeNodeSearchNoSelect Treffer nicht selektieren

WinTreeNodeSearchStart Sucheingabefeld öffnen

handle3 Referenzknoten für untergeordnete Suche

> 0 Deskriptor des gefundenen Knotens

Resultat handle = 0 Kein Knoten gefunden

< 0 Fehlerwert

Siehe Verwandte Befehle, TreeNode, TreeView, EvtNodeSearch, Blog

Mit diesem Befehl wird in einem TreeView-Objekt oder einem TreeNode-Objekt (obj) nach einem bestimmten TreeNode gesucht.



Ist ein Referenzknoten (handle3) angegeben und wird das Ende der Unterelemente des übergebenen TreeView bzw. TreeNode-Objektes (obj) erreicht, beginnt die Suche von vorne, bis zum wieder Erreichen des Referenzknotens.

Wird als Startobjekt (obj) ein TreeNode-Objekt übergeben, wird standardmäßig auch in höheren Baumebenen weiter gesucht.

Folgende Optionen (int2) können angegeben werden:

- **WinTreeNodeSearchCaption**

Vergleicht den Suchtext mit Caption-Eigenschaft des TreeNode-Objektes.

- **WinTreeNodeSearchCustom**

Vergleicht den Suchtext mit Custom-Eigenschaft des TreeNode-Objektes.

- **WinTreeNodeSearchCI**

Die Groß-/Kleinschreibung wird bei der Suche ignoriert.

- **WinTreeNodeSearchLike**

Führt einen Wildcard-Vergleich durch.

- **WinTreeNodeSearchLikeAuto**

Führt einen Wildcard-Vergleich durch. Der Suchtext wird automatisch mit *<Suchtext>* maskiert. Dies entspricht dem Verhalten im Designer.

- **WinTreeNodeSearchToken**

Führt eine begriffsorientierte Suche durch.

- **WinTreeNodeSearchRegEx**

Vergleich mittels regulären Ausdrücken.

- **WinTreeNodeSearchChildrenOnly**

Nur untergeordnete Knoten werden durchsucht.

- **WinTreeNodeSearchUp**

Die Suche wird nach oben durchgeführt.

- **WinTreeNodeSearchNoSelect**

Der gefundene Knoten wird nicht ausgewählt.

- **WinTreeNodeSearchStart**

Das Sucheingabefeld wird geöffnet. Hierbei wird das Sucheingabefeld mit dem Wert von (alpha1) vorbelegt. Für die Suche werden die restlichen angegebenen Optionen verwendet.



Die Prozedur bleibt beim Öffnen des Sucheingabefeldes **nicht** stehen. Dadurch ist der Rückgabewert der Funktion, bei Angabe dieser Option, immer 0.

Mehrere Optionen können miteinander kombiniert werden. Die Kombination erfolgt durch eine binäre ODER-Verknüpfung.



Die Optionen WinTreeNodeSearchLike, WinTreeNodeSearchLikeAuto, WinTreeNodeSearchToken und WinTreeNodeSearchRegEx können nicht miteinander kombiniert werden.

Ist weder WinTreeNodeSearchCaption noch WinTreeNodeSearchCustom gesetzt wird trotzdem das Ereignis EvtNodeSearch aufgerufen.

Wird die Option (int2) WinTreeNodeSearchChildrenOnly angegeben, kann in (handle4) ein Referenzknoten angegeben werden. Dabei wird nur unterhalb des Knotens (obj) vom Referenzknoten (handle4) ausgehend gesucht. Ist bei der Option kein Referenzknoten (handle4) angegeben, wird ab dem ersten untergeordneten Knoten gesucht.

Resultat

Wird ein TreeNode gefunden, ist das Resultat der Deskriptor des TreeNode-Objektes. Wurde kein TreeNode gefunden, ist das Resultat 0. Zusätzlich sind die folgenden Fehlerwerte möglich:

Fehlerwert

Bedeutung

<u>ErrUnavailable</u>	Der aktuell geprüfte Knoten wurde im Ereignis <u>EvtNodeSearch</u> gelöscht. Daher kann die Suche nicht fortgesetzt werden.
<u>ErrRegExRuleSyntax</u>	Syntaxfehler im regulären Ausdruck
<u>ErrRegExBadEscapeSequence</u>	Nicht aufgelöste Escape-Sequenz im Ausdruck
<u>ErrRegExPropertySyntax</u>	Ungültige Unicode-Eigenschaft
<u>ErrRegExNotSupported</u>	Verwendung einer Funktion, die nicht unterstützt wird
<u>ErrRegExMismatchedParentheses</u>	Falsch verschachtelte Klammern im regulären Ausdruck
<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 Klammerausdruck
<u>ErrRegExInvalidRange</u>	In einer Zeichenmenge [x-y] ist x größer als y
<u>ErrRegExStackOverflow</u>	Stapelüberlauf in der Ablaufverfolgung des regulären Ausdrucks

Die Suche wird so lange durchgeführt, bis ein TreeNode gefunden, die Suche im Ereignis EvtNodeSearch mit dem Rückgabewert false abgebrochen oder das Startobjekt (obj) bzw. der Referenzknoten (handle4) bei WinTreeNodeSearchChildrenOnly wieder erreicht wurde.

Beispiel:

```
// Findet alle untergeordneten TreeNodes, die mit '.doc' enden und ändert die Schriftartfor{ tF
```

Mögliche Laufzeitfehler:

Kontakt

__ErrHdlInvalid In (obj) oder (handle3) wurde kein gültiger Deskriptor eines TreeView- oder TreeNode-Objektes übergeben.
__ErrValueInvalid In (int2) ist __WinTreeNodeSearchChildrenOnly gesetzt und (handle3) ist kein untergeordnetes TreeNode-Objekt von (obj).
__ErrValueInvalid Es wurde eine ungültige Kombination an Optionen (int2) übergeben.

Konstanten für TreeView-Befehle

Liste der Konstanten für TreeView-Befehle

Siehe TreeView-Befehle

Konstanten

- WinTreeNodeSearchCaption
- WinTreeNodeSearchChildrenOnly
- WinTreeNodeSearchCI
- WinTreeNodeSearchCustom
- WinTreeNodeSearchLike
- WinTreeNodeSearchLikeAuto
- WinTreeNodeSearchNoSelect
- WinTreeNodeSearchRegEx
- WinTreeNodeSearchStart
- WinTreeNodeSearchToken
- WinTreeNodeSearchUp

_WinTreeNodeSearchCaption

Caption-Eigenschaft der TreeNode-Objekte durchsuchen

Wert 1 / 0x0001

Siehe WinTreeNodeSearch(),

SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der die Caption-Eigenschaft der TreeNode-Objekte durchsucht wird.

_WinTreeNodeSearchChildrenOnly

Suche nur nach untergeordneten TreeNode-Objekten

Wert 8 / 0x0008

Siehe WinTreeNodeSearch(),
SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der nur nach untergeordneten TreeNode-Objekten gesucht wird.

_WinTreeNodeSearchCI

Groß-/Kleinschreibung bei der Suche ignorieren

Wert 4 / 0x0004

Siehe WinTreeNodeSearch(),
SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der bei der Suche die Groß-/Kleinschreibung ignoriert wird.

_WinTreeNodeSearchCustom
Custom-Eigenschaft der TreeNode-Objekte durchsuchen
Wert 2 / 0x0002

Siehe WinTreeNodeSearch(),
SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der die
Custom-Eigenschaft der TreeNode-Objekte durchsucht wird.

_WinTreeNodeSearchLike

Suche mit Wildcard-Vergleich durchführen

Wert 32 / 0x0020

Siehe WinTreeNodeSearch(),
SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der ein Wildcard-Vergleich bei der Suche durchgeführt wird.

_WinTreeNodeSearchLikeAuto

Suche mit Wildcard-Vergleich und automatischer Maskierung durchführen

Wert 16 / 0x0010

Siehe WinTreeNodeSearch(),
SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der ein Wildcard-Vergleich bei der Suche durchgeführt wird. Der Suchbegriff wird automatisch mit *<Suchbegriff>* maskiert.

_WinTreeNodeSearchNoSelect

Suchtreffer nicht selektieren

Wert 8192 / 0x2000

Siehe WinTreeNodeSearch()

Option bei WinTreeNodeSearch(), mit der der Treffer bei der Suche nicht ausgewählt wird.

_WinTreeNodeSearchRegEx

Suche mit regulären Ausdrücken durchführen

Wert 128 / 0x0080

Siehe WinTreeNodeSearch(),
SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der die Suche mit regulären Ausdrücken durchgeführt wird.

_WinTreeNodeSearchStart

Sucheingabefeld öffnen

Wert 16384 / 0x4000

Siehe WinTreeNodeSearch()

Option bei WinTreeNodeSearch(), mit der das Sucheingabefeld geöffnet wird.



Die Prozedur bleibt beim Öffnen des Sucheingabefeldes **nicht** stehen. Dadurch ist der Rückgabewert der Funktion, bei Angabe dieser Option, immer 0.

_WinTreeNodeSearchToken

Begriffsorientierte Suche durchführen

Wert 64 / 0x0040

Siehe WinTreeNodeSearch(),
SearchFlags

Option bei WinTreeNodeSearch() und in den SearchFlags, mit der eine begriffsorientierte Suche durchgeführt wird.

_WinTreeNodeSearchUp

Suche nach oben durchführen

Wert 4096 / 0x1000

Siehe WinTreeNodeSearch()

Option bei WinTreeNodeSearch(), mit der die Suche vom angegebenen Startobjekt nach oben durchgeführt wird.

Fokusbefehle

Befehle zum Ermitteln/Setzen des Eingabefokus

Verwandte

Siehe Befehle,
Befehlsgruppen,
Befehlsliste

Befehle

- WinFocusGet
- WinFocusSet

Konstanten

- _StandardClient

Kontakt

WinFocusGet() : int



Objekt mit Eingabefokus ermitteln


Resultat int Objekt mit Eingabefokus

Siehe Verwandte Befehle, TabPos

Es wird das Objekt zurückgegeben, das den Eingabefokus besitzt. 0 wird dann zurückgegeben, wenn das Objekt nicht ermittelt werden kann. Dies ist zum Beispiel dann der Fall, wenn zum Zeitpunkt der Befehlsausführung eine andere Applikation aktiv ist.

Beispiel:

```
// Objekt mit Eingabefokus ermitteltHdlObj # WinFocusGet();
```

obj -> WinFocusSet([logic1]) : 
int

Objekt mit Eingabefokus setzen

obj Zu fokussierendes Objekt

logic1 Ereignisaktivierung (optional)

Resultat int Momentan fokussiertes Objekt

Siehe Verwandte Befehle, TabPos

Dem Befehl wird das Objekt übergeben, auf den der Fokus gesetzt werden soll. Der Rückgabewert dieses Befehls ist entweder der Deskriptor des vorhergehenden Objekts oder der Wert 0. Ist der Wert 0, kann das unterschiedliche Ursachen haben. Zum einen kann das Objekt als nicht Visible oder als Disabled definiert worden sein, zum anderen kann das Objekt der Standardclient sein oder einem anderen Prozess gehören.

Durch die Übergabe der Konstanten StandardClient kann der Fokus auf den textbasierten Standardclient gesetzt werden.

Es ist zu beachten, dass ein Objekt nicht den Fokus erlangen kann, wenn es nicht sichtbar (Visible = false) oder gesperrt (Disabled = true) ist. Unterstützt ein Objekt keine Eingabe (zum Beispiel das Objekt Label), kann es ebenfalls keinen Fokus bekommen.

Mit dem optionalen Parameter (logic1) können während der Verarbeitung eines Ereignisses weitere Ereignisse unterdrückt (logic1 = false oder Parameter nicht angeben) werden. Wird in einer Funktion, die durch EvtFocusTerm aufgerufen wurde, der Fokus neu gesetzt, ist das nächste Ereignis das EvtFocusInit des neuen Eingabeobjekts.

Beispiel:

```
// Fokus auf Objekt '$edCltno' setzen $edCltno->WinFocusSet();
```



Der Befehl kann ebenfalls in dem Ereignis EvtInit eines Frame-Objekts aufgerufen bzw. zwischen dem Laden des Dialogs mit WinOpen() und dem Anzeigen mit WinDialogRun() durchgeführt werden. Da zu diesem Zeitpunkt ein Fokus nicht zugewiesen werden kann, erfolgt die Ausführung des Befehls verzögert. Der Rückgabewert des Befehls ist dann immer 0.

Mögliche Laufzeitfehler:

ErrHdlInvalid Objekt (obj) ungültig

_StandardClient

Konstante für den Standardclient

Wert 10003

Siehe WinFocusSet()

Mit dieser Konstante kann der Fokus auf den Standardclient gesetzt werden.

Der Befehl WinFocusGet() liefert nicht diese Konstante zurück. Es kann also nicht ermittelt werden, ob der Standardclient den Fokus besitzt.

Ereignisbefehle

Verändern der Ereignisse

Liste sortiert

nach

Siehe Gruppen,

Alphabetische

Liste aller

Befehle

Befehle

- ComEvtProcessGet
- ComEvtProcessSet
- WinEvtProcessGet
- WinEvtProcessSet
- WinEvtProcNameGet
- WinEvtProcNameSet



obj -> ComEvtProcessGet(int1) : logic

Ermitteln, ob ein COM-Ereignis aktiviert oder deaktiviert ist

obj Objekt

int1 Konstante des Ereignisses

Resultat logic Aktivierung des Ereignisses

Siehe Verwandte Befehle,

ComEvtProcessSet(), CtxDocEdit

Mit diesem Befehl kann ermittelt werden, ob das in (int1) angegebene COM-Ereignis bei dem Objekt (obj) aktiviert (Resultat = true) oder deaktiviert (Resultat = false) ist.

Eine Liste der Ereignis-Konstanten ist in der Prozedur CtxDocEdit.Define.prc in dem CodeLibrary-Beispiel "CtxDocEdit" enthalten. Diese Prozedur kann auch in der eigenen Applikation inkludiert werden. Die Liste der Ereignisse und deren Beschreibung finden Sie zusätzlich auf der Hersteller-Seite des Moduls. Die dort genannten Ereignisse können dann mit dem Präfix txEvt verwendet werden.

Beispiel

```
if (!$ctxDocEdit->ComEvtProcessGet(txEvtPosChange)){ // Bei der Änderung der Cursorposition wird
```

obj -> ComEvtProcessSet(int1, logic2) : logic
 Aktivieren oder Deaktivieren eines COM-Ereignisses



obj Objekt

int1 Konstante des Ereignisses

logic2 Ereignis aktivieren (true) oder
 deaktivieren (false)

Resultat logic vorhergehender Zustand

Verwandte Befehle,
 Siehe ComEvtProcessGet(),
 CtxDocEdit

Mit diesem Befehl kann die Verarbeitung eines COM-Ereignisses aktiviert oder unterdrückt werden. In (int1) wird die Ereignis-Nummer übergeben. Wird in (logic2) false übergeben, wird dieses Ereignis bei dem Objekt (obj) nicht mehr ausgelöst.

Eine Liste der Ereignis-Konstanten ist in der Prozedur CtxDocEdit.Define.prc in dem CodeLibrary-Beispiel "CtxDocEdit" enthalten. Diese Prozedur kann auch in der eigenen Applikation inkludiert werden. Die Liste der Ereignisse und deren Beschreibung finden Sie zusätzlich auf der Hersteller-Seite des Moduls. Die dort genannten Ereignisse können dann mit dem Präfix txEvt verwendet werden.

Wird ein aktiviertes COM-Ereignis ausgelöst, wird das Ereignis EvtCtxEvent des Objektes (obj) aufgerufen. In dem Parameter (aEventID) wird die Nummer des COM-Ereignisses übergeben.

Der Rückgabewert enthält den vorhergehenden Aktivierungszustand. Ob ein COM-Ereignis aktiviert oder deaktiviert ist, kann mit dem Befehl ComEvtProcessGet() ermittelt werden.

Beispiele:

```
// Ereignis bei Änderung der Cursor-Position aktivieren $ctxDocEdit->ComEvtProcessSet(txEvtPosChar
```



WinEvtProcessGet(int1) : logic

Ermitteln, ob ein Ereignis aktiviert oder deaktiviert ist

int1 Konstante des Ereignisses

Resultat logic Aktivierung des Ereignisses

Siehe Verwandte Befehle,
WinEvtProcessSet(),
WinEvtProcNameGet(),
WinEvtProcNameSet()

Mit diesem Befehl kann ermittelt werden, ob das in (int1) angegebene Ereignis aktiviert (Resultat = true) oder deaktiviert (Resultat = false) ist.

Folgende Konstanten können in (int1) angegeben werden:

- WinEvtAdviseDDE
- WinEvtAttachState
- WinEvtChanged
- WinEvtChangedActive
- WinEvtChangedChild
- WinEvtChangedDesign
- WinEvtClicked
- WinEvtClose
- WinEvtCreated
- WinEvtCroNavigate
- WinEvtCtxEvent
- WinEvtDbFldUpdate
- WinEvtDragInit
- WinEvtDragTerm
- WinEvtDrop
- WinEvtDropEnter
- WinEvtDropLeave
- WinEvtDropOver
- WinEvtEndSession
- WinEvtFocusCancel
- WinEvtFocusInit
- WinEvtFocusTerm
- WinEvtFsiMonitor
- WinEvtHelpTip
- WinEvtHyphenate
- WinEvtInit
- WinEvtIvlDropItem
- WinEvtJob
- WinEvtKeyItem
- WinEvtLstDataInit
- WinEvtLstEditActivate
- WinEvtLstEditCommit
- WinEvtLstEditEndGroup
- WinEvtLstEditEndItem
- WinEvtLstEditFinished
- WinEvtLstEditStart
- WinEvtLstEditStartGroup

- WinEvtLstEditStartItem
- WinEvtLstGroupArrange
- WinEvtLstGroupInit
- WinEvtLstRecControl
- WinEvtLstSelect
- WinEvtLstSelectRange
- WinEvtLstViewInit
- WinEvtMdiActivate
- WinEvtMenuCommand
- WinEvtMenuContext
- WinEvtMenuInitPopup
- WinEvtMenuPopup
- WinEvtMouse
- WinEvtMouseEvent
- WinEvtMouseMove
- WinEvtNodeExpand
- WinEvtNodeSearch
- WinEvtNodeSelect
- WinEvtPageSelect
- WinEvtPosChanged
- WinEvtReadOnlyChanged
- WinEvtSocket
- WinEvtSystem
- WinEvtTapi
- WinEvtTerm
- WinEvtTimer
- WinEvtUser

Beispiel

```
if (!WinEvtProcessGet(_WinEvtFocusTerm)){ // EvtFocusTerm ist deaktiviert ...}
```



WinEvtProcessSet(int1, logic2) : logic
Aktivieren oder Deaktivieren eines Ereignisses

int1 Konstante des Ereignisses

logic2 Ereignis aktivieren (true) oder
deaktivieren (false)

Resultat logic vorhergehender Zustand

Siehe Verwandte Befehle,
WinEvtProcessGet(),
WinEvtProcNameGet(),
WinEvtProcNameSet()

Mit diesem Befehl kann die Verarbeitung eines Ereignisses oder aller Ereignisse unterdrückt werden. In (int1) wird das Ereignis übergeben. Wird in (logic2) false übergeben, wird dieses Ereignis bei allen Objekten nicht mehr ausgelöst.

Das Ereignis wird erst dann wieder ausgelöst, wenn es mit (logic2) = true wieder aktiviert wird.

Folgende Konstanten können in (int1) angegeben werden:

- WinEvtAdviseDDE
- WinEvtAttachState
- WinEvtChanged
- WinEvtChangedActive
- WinEvtChangedChild
- WinEvtChangedDesign
- WinEvtClicked
- WinEvtClose
- WinEvtCreated
- WinEvtCroNavigate
- WinEvtCtxEvent
- WinEvtDbFldUpdate
- WinEvtDragInit
- WinEvtDragTerm
- WinEvtDrop
- WinEvtDropEnter
- WinEvtDropLeave
- WinEvtDropOver
- WinEvtEndSession
- WinEvtFocusCancel
- WinEvtFocusInit
- WinEvtFocusTerm
- WinEvtFsiMonitor
- WinEvtHelpTip
- WinEvtHyphenate
- WinEvtInit
- WinEvtIvlDropItem
- WinEvtJob
- WinEvtKeyItem
- WinEvtLstDataInit
- WinEvtLstEditActivate

- WinEvtLstEditCommit
- WinEvtLstEditEndGroup
- WinEvtLstEditEndItem
- WinEvtLstEditFinished
- WinEvtLstEditStart
- WinEvtLstEditStartGroup
- WinEvtLstEditStartItem
- WinEvtLstGroupArrange
- WinEvtLstGroupInit
- WinEvtLstRecControl
- WinEvtLstSelect
- WinEvtLstSelectRange
- WinEvtLstViewInit
- WinEvtMdiActivate
- WinEvtMenuCommand
- WinEvtMenuContext
- WinEvtMenuInitPopup
- WinEvtMenuPopup
- WinEvtMouse
- WinEvtMouseEvent
- WinEvtMouseMove
- WinEvtNodeExpand
- WinEvtNodeSearch
- WinEvtNodeSelect
- WinEvtPageSelect
- WinEvtPosChanged
- WinEvtReadOnlyChanged
- WinEvtSocket
- WinEvtSystem
- WinEvtTapi
- WinEvtTerm
- WinEvtTimer
- WinEvtUser

Sollen alle Ereignisse aktiviert oder deaktiviert werden, kann die Konstante WinEvtAll verwendet werden.

Der Rückgabewert enthält den vorhergehenden Aktivierungszustand. Ob ein Ereignis aktiviert oder deaktiviert ist, kann mit dem Befehl WinEvtProcessGet() ermittelt werden.



Ereignisse, die nur dann aufgerufen werden, wenn ein anderes Ereignis einen bestimmten Wert zurück gibt (zum Beispiel EvtFocusCancel), werden durch die Deaktivierung ebenfalls nicht mehr ausgeführt.

Beispiele:

```
// Keine Ereignisse mehr auslösenWinEvtProcessSet(_WinEvtAll, false);// Alle Ereignisse wieder ak
```



obj -> WinEvtProcNameGet(int1) : alpha

Ermittelt den Funktionsnamen eines Ereignisses

obj Fenster-Objekt

int1 Konstante des Ereignisses

Resultat alpha Prozedur und Funktionsname

Verwandte Befehle,

Siehe WinEvtProcessGet(),

WinEvtProcessSet(),

WinEvtProcNameSet()

Der Befehl liefert den Namen der Ereignissprozedur zurück, die beim Objekt (obj) beim Ereignis (int1) hinterlegt ist.

Folgende Konstanten können in (int1) angegeben werden:

- WinEvtAdviseDDE
- WinEvtAttachState
- WinEvtChanged
- WinEvtChangedActive
- WinEvtChangedChild
- WinEvtChangedDesign
- WinEvtClicked
- WinEvtClose
- WinEvtCreated
- WinEvtCroNavigate
- WinEvtCtxEvent
- WinEvtDbFldUpdate
- WinEvtDragInit
- WinEvtDragTerm
- WinEvtDrop
- WinEvtDropEnter
- WinEvtDropLeave
- WinEvtDropOver
- WinEvtEndSession
- WinEvtFocusCancel
- WinEvtFocusInit
- WinEvtFocusTerm
- WinEvtFsiMonitor
- WinEvtHelpTip
- WinEvtHyphenate
- WinEvtInit
- WinEvtIvlDropItem
- WinEvtJob
- WinEvtKeyItem
- WinEvtLstDataInit
- WinEvtLstEditActivate
- WinEvtLstEditCommit
- WinEvtLstEditEndGroup
- WinEvtLstEditEndItem
- WinEvtLstEditFinished
- WinEvtLstEditStart

- WinEvtLstEditStartGroup
- WinEvtLstEditStartItem
- WinEvtLstGroupArrange
- WinEvtLstGroupInit
- WinEvtLstRecControl
- WinEvtLstSelect
- WinEvtLstSelectRange
- WinEvtLstViewInit
- WinEvtMdiActivate
- WinEvtMenuCommand
- WinEvtMenuContext
- WinEvtMenuInitPopup
- WinEvtMenuPopup
- WinEvtMouse
- WinEvtMouseItem
- WinEvtMouseMove
- WinEvtNodeExpand
- WinEvtNodeSearch
- WinEvtNodeSelect
- WinEvtPageSelect
- WinEvtPosChanged
- WinEvtReadOnlyChanged
- WinEvtSocket
- WinEvtTapi
- WinEvtTerm
- WinEvtTimer
- WinEvtUser

Beispiel

```
tHdl->WinEvtProcNameGet(_WinEvtTimer) // -> "Prozedur:EvtTimer"
```

obj -> WinEvtProcNameSet(int1,
alpha2) : logic



Ereignisfunktion eines Objekts setzen

obj Deskriptor des Objekts

int1 ID des Ereignisses

alpha2 Name der Ereignisfunktion

Resultat logic Erfolg der Anweisung

Siehe Verwandte Befehle,
 WinEvtProcessGet(),
 WinEvtProcessSet(),
 WinEvtProcNameGet()

Mit dem Befehl kann bei dem angegebenen Objekt (obj) eine Ereignisfunktion gesetzt werden.

Die ID des Ereignisses wird in (int1) übergeben. Folgende Konstanten können übergeben werden:

- WinEvtAdviseDDE
- WinEvtAttachState
- WinEvtChanged
- WinEvtChangedActive
- WinEvtChangedChild
- WinEvtChangedDesign
- WinEvtClicked
- WinEvtClose
- WinEvtCreated
- WinEvtCroNavigate
- WinEvtCtxEvent
- WinEvtDbFldUpdate
- WinEvtDragInit
- WinEvtDragTerm
- WinEvtDrop
- WinEvtDropEnter
- WinEvtDropLeave
- WinEvtDropOver
- WinEvtEndSession
- WinEvtFocusCancel
- WinEvtFocusInit
- WinEvtFocusTerm
- WinEvtFsiMonitor
- WinEvtHelpTip
- WinEvtHyphenate
- WinEvtInit
- WinEvtIvlDropItem
- WinEvtJob
- WinEvtKeyItem
- WinEvtLstDataInit
- WinEvtLstEditActivate
- WinEvtLstEditCommit
- WinEvtLstEditEndGroup

- WinEvtLstEditEndItem
- WinEvtLstEditFinished
- WinEvtLstEditStart
- WinEvtLstEditStartGroup
- WinEvtLstEditStartItem
- WinEvtLstGroupArrange
- WinEvtLstGroupInit
- WinEvtLstRecControl
- WinEvtLstSelect
- WinEvtLstSelectRange
- WinEvtLstViewInit
- WinEvtMdiActivate
- WinEvtMenuCommand
- WinEvtMenuContext
- WinEvtMenuInitPopup
- WinEvtMenuPopup
- WinEvtMouse
- WinEvtMouseEvent
- WinEvtMouseMove
- WinEvtNodeExpand
- WinEvtNodeSearch
- WinEvtNodeSelect
- WinEvtPageSelect
- WinEvtPosChanged
- WinEvtReadOnlyChanged
- WinEvtSocket
- WinEvtTapi
- WinEvtTerm
- WinEvtTimer
- WinEvtUser

In (alpha2) wird der Prozedur- und der Funktionsname übergeben. Prozedur- und Funktionsname werden dabei durch einen : voneinander getrennt.

Konnte das Ereignis nicht gesetzt werden, ist der Rückgabewert false. In diesem Fall verfügt das Objekt nicht über das angegebene Ereignis. Bei erfolgreicher Durchführung ist der Rückgabewert true.



Bei dynamisch erzeugten Objekten können die Ereignisse EvtInit und EvtTerm nicht gesetzt werden.

Beispiel:

```
tButton->WinEvtProcNameSet(_WinEvtClicked, 'Proc:EvtClicked');
```

Sonstige Befehle

Sonstige Befehle

Liste sortiert

nach

Siehe Gruppen,
Alphabetische

Liste aller

Befehle

Befehle

- ColorMake
- FontMake
- PointMake
- RangeMake
- RectMake
- RtfTabMake
- WinBarcodeSaveImage
- WinBeep
- WinBoxScrollVisible
- WinColorOpacityGet
- WinColorOpacitySet
- WinFlash
- WinHalt
- WinIconPreload
- WinLayer
- WinLstEdit
- WinPicSaveImage
- WinSearch
- WinSearchClear
- WinSearchPath
- WinSearchPathGet
- WinShutdownBlock
- WinUpdate
- WinWbnExecCommand

Konstanten

- _WinBeepDefault
- _WinBeepError
- _WinBeepInformation
- _WinBeepQuestion
- _WinBeepWarning
- _WinDialogMaximized
- _WinDialogMinimized
- _WinDialogNormal
- _WinDialogPrimary
- _WinFlashCaption
- _WinFlashDefault
- _WinFlashStop
- _WinFlashTray
- _WinFlashUntilFore

- WinFlashUntilStop
- WinGntRecalc
- WinGntRefresh
- WinImageFormatBmp
- WinImageFormatJpg
- WinImageFormatPng
- WinImageFormatTif
- WinLayerDarken
- WinLayerEnd
- WinLayerStart
- WinLayerUpdate
- WinLstActiveOnly
- WinLstEditClearChanged
- WinLstEditLst
- WinLstEditLstAlpha
- WinLstFromFirst
- WinLstFromLast
- WinLstFromSelected
- WinLstFromTop
- WinLstIgnoreHidden
- WinLstPosBottom
- WinLstPosMiddle
- WinLstPosSelected
- WinLstPosTop
- WinLstRecDoSelect
- WinLstRecEvtSkip
- WinLstRecFromBuffer
- WinLstRecFromRecId
- WinLstReorderColumns
- WinMsgAll
- WinMsgNoInput
- WinMsgNoKeyboardInput
- WinMsgNoMouseInput
- WinPicSignReset
- WinPicSignSaveAsExt
- WinPicSignSaveAsJpg
- WinPicSignSaveAsPng
- WinPicSignSaveAsTif
- WinPicSignSavePic
- WinPicSignSaveSign
- WinRvwEditAbortEditing
- WinRvwUpdateDoKeepSelect
- WinRvwUpdateDoSelect
- WinRvwUpdateFromFirst
- WinRvwUpdateFromLast
- WinRvwUpdateFromRecBuf
- WinRvwUpdateFromSelected
- WinRvwUpdateFromTop
- WinRvwUpdateFromTopLocation
- WinRvwUpdateOptClearCache
- WinRvwUpdateOptClearSelected

- WinUpdActivate
- WinUpdBuf2Obj
- WinUpdFld2Buf
- WinUpdFld2Obj
- WinUpdObj2Buf
- WinUpdObj2Fld
- WinUpdOff
- WinUpdOn
- WinUpdSort
- WinUpdState
- WinWbnCommandExecPrint
- WinWbnCommandNavBack
- WinWbnCommandNavForward
- WinWbnCommandNavHome
- WinWbnCommandNavSearch
- WinWbnCommandSelCopy
- WinWbnCommandSelCut
- WinWbnCommandSelDelete
- WinWbnCommandSelPaste
- WinWbnCommandSelSelectAll
- WinWbnCommandUpdNoCache
- WinWbnCommandUpdNormal
- WinWbnCommandZoomIn
- WinWbnCommandZoomOut
- WinWbnCommandZoomReset

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 TransparenztColor # ColorMake(_WinColAct
```

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
```


FontMake(alpha1[, int2[, int3]]) :
font



Zusammenstellen eines font-Wertes

alpha1 Name der
Schriftart
Größe der
int2 Schriftart
(optional)
int3 Schrift-Attribute
(optional)

Resultat font Schriftart
font (Datentyp),

Siehe Font
(Eigenschaft)

Dieser Befehl erzeugt aus den Angaben eine font-Struktur. Im Parameter (alpha1) wird der Name der Schriftart angegeben. Wird eine leere Zeichenkette übergeben wird die Schriftart des Systems verwendet.

In den Parametern (int2) und (int3) können die Größe der Schriftart in 1/10 Punkten und die Schriftattribute angegeben werden. Für die Schriftattribute stehen folgende Konstanten zur Verfügung:

<u>WinFontAttrNormal</u>	Normale Darstellung
<u>WinFontAttrBold</u>	Fett
<u>WinFontAttrItalic</u>	Kursiv
<u>WinFontAttrStrikeOut</u>	Durchgestrichen
<u>WinFontAttrUnderline</u>	Unterstrichen

Die Schrift-Attribute können miteinander kombiniert werden.

Beispiele:

```
// Arial 10pttFont # FontMake('Arial', 100, _WinFontAttrNormal);// System font 12pt bold and unde
```

Mögliche Laufzeitfehler:

<u>ErrStringOverflow</u>	In (alpha1) wurde eine Zeichenkette länger als 31 Zeichen übergeben.
<u>ErrValueInvalid</u>	Im Parameter (int3) wurde ein ungültiger Wert übergeben.

PointMake(int1, int2) :



point

Erzeugung eines Point

int1 X-Wert

int2 Y-Wert

Resultat point Erzeugter Punkt

Siehe Verwandte Befehle,

point

Dieser Befehl erzeugt durch die Angaben eines Punktes eine point-Struktur.

Beispiel:

```
local{ tArea      : point;} ... // Erzeugen eines Punktes auf 100, 100 tArea = PointMake(1
```

RangeMake(int1, int2) :



range

Erzeugung einer Markierung

int1 Anfang


int2 Ende

Resultat range Bereich

Siehe range, Range

Dieser Befehl erzeugt aus den Angaben des Bereichsanfangs und -endes eine range-Struktur.

Der Bereichsanfang und -ende wird in Zeichen angegeben. Bei den Eingabe-Objekten kann der komplette Text mit RangeMake(0, -1) selektiert werden.

RectMake(int1, int2, ,
int3, int4) : rect

Erzeugung eines Rects

int1 Abstand des linken
Randes

int2 Abstand des oberen
Randes

int3 Abstand des rechten
Randes

int4 Abstand des unteren
Randes

Resultat rect Erzeugtes Rechteck

Siehe Verwandte Befehle, rect

Dieser Befehl erzeugt aus den Angaben des Anfangs- und Endpunktes eine rect-Struktur.

Beispiel:

```
local{ tArea : rect;} ... // Erzeugen eines Rechteck auf 100, 100 mit der Größe 100, 100 tA
```

WinBeep([int1])



Systemklang wiedergeben

int1 Systemklang
 (optional)

Verwandte

Befehle,

Siehe SysBeep(),

Benutzerdefinierte

Sounds (Blog)

Mit diesem Befehl kann ein Systemklang wiedergegeben werden. Folgende Konstanten sind zulässig:

_WinBeepDefault Systemklang für "Standardton Warnsignal"

_WinBeepInformation Systemklang für "Sternchen"

_WinBeepError Systemklang für "Kritischer Fehler"


_WinBeepWarning Systemklang für "Hinweis"

_WinBeepQuestion Systemklang für "Frage"

Wird kein Argument angegeben, wird _WinBeepDefault verwendet.

Mögliche Laufzeitfehler:

_ErrValueInvalid In (int1) wurde kein gültiger Wert angegeben.

obj -> WinBoxScrollVisible(handle1) : 

logic

Scrollt auf ein nicht sichtbares Objekt

obj Scrollbox-Objekt

Objekt, das in den

handle1 Anzeigebereich gescrollt

werden soll

Resultat logic Scrollen notwendig

Siehe Verwandte Befehle

Dieser Befehl scrollt das Oberflächenobjekt (handle1) in den Anzeigebereich der Scrollbox (obj), sofern es nicht bereits im sichtbaren Bereich ist.

Der Rückgabewert gibt Aufschluss darüber, ob ein Scrolling notwendig war (true) oder nicht (false).

Mögliche Laufzeitfehler:

_ErrHdlInvalid In (obj) wurde kein Deskriptor eines Scrollbox-Objektes angegeben oder (handle1) ist kein untergeordnetes Objekt der Scrollbox.

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,
int2) : 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.

obj ->



WinCroNavigate(int1)

Navigiert zu einer URL

obj Objekt

 (Chromium-Objekt)

int1 Art der Navigation

Siehe Verwandte Befehle

Navigiert zu einer URL, deren Inhalt im Verlauf der Navigation bereits angezeigt wurde, falls eine solche Navigation stattgefunden hat.

Im Parameter (int1) können folgende Optionen übergeben werden:

- **WinCroNavBack (1)**

 Navigiert zur vorhergehenden Seite.

- **WinCroNavForward (2)**

 Navigiert zur nächsten Seite.

Mögliche Laufzeitfehler

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein Chromium-Objekt.

ErrValueInvalid In Argument (int1) wurde keiner der gültigen Werte für die Navigation angegeben.

obj ->

WinCroPrint()

Inhalt drucken

obj Objekt

(Chromium-Objekt)

Siehe Verwandte Befehle

Der Befehl ruft einen Dialog zum Drucken des Inhaltes der aktuell angezeigten Seite aus.

Mögliche Laufzeitfehler

_ErrHdlInvalid Bei (obj) handelt es sich nicht um ein Chromium-Objekt.





obj -> WinCroReload(int1)

Lädt den Inhalt der aktuellen URL erneut

obj Objekt

 (Chromium-Objekt)

int1 Optionen

Siehe Verwandte Befehle

Der Befehl lädt den Inhalt der aktuell angezeigten Seite erneut. Über Optionen kann gesteuert werden, ob die Seite neu angefordert oder aus dem Cache geladen werden soll.

Im Parameter (int1) können folgende Optionen übergeben werden:

- **WinCroReloadIgnoreCache (1)**

 Zwischengespeicherte Inhalte werden erneut angefordert.

- **WinCroReloadNormal (0)**

 Zwischengespeicherte Inhalte werden nicht erneut angefordert (default).

Mögliche Laufzeitfehler

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein Chromium-Objekt.

ErrValueInvalid In Argument (int1) wurde keine der gültigen Optionen angegeben.



obj -> WinCroSelection(int1)

Aktionen für den ausgewählten Inhalt

obj Objekt

(Chromium-Objekt)

int1 Aktion

Siehe Verwandte Befehle

Der Befehl führt Aktionen auf den aktuell angezeigten Inhalt aus, sofern möglich.

Im Parameter (int1) können folgende Optionen übergeben werden:

- **_WinCroSelCopy (1)**

Kopiert den ausgewählten Inhalte in die Zwischenablage.

- **_WinCroSelCut (2)**

Schneidet den ausgewählten Inhalt in die Zwischenablage aus.

- **_WinCroSelDelete (4)**

Löscht den ausgewählten Inhalt.

- **_WinCroSelPaste (3)**

Ersetzt den ausgewählten Inhalt durch den Inhalt in der Zwischenablage.

- **_WinCroSelSelectAll (5)**

Wählt den gesamten Inhalt aus.

Mögliche Laufzeitfehler

_ErrHdlInvalid Bei (obj) handelt es sich nicht um ein Chromium-Objekt.

_ErrValueInvalid In Argument (int1) wurde keine der gültigen Optionen angegeben.



obj -> WinFlash([int1])

Anwendung in der Taskleiste blinken lassen

obj Deskriptor des Fensters

Optionen (optional)

WinFlashDefault Lässt die Titelzeile und das Icon in der Taskleiste blinken.

WinFlashTray Lässt das Icon in der Taskleiste blinken.

WinFlashCaption Lässt die Titelzeile der Anwendung blinken.

int1 WinFlashUntilStop Fenster blinken lassen, bis WinFlashStop aufgerufen oder das Fenster beendet wird.

WinFlashUntilFore Fenster blinken lassen, bis es in den Vordergrund geholt wird.

WinFlashStop Blinken des Fensters stoppen.

Siehe Verwandte Befehle

Mit diesem Befehl kann der Anwender darauf hingewiesen werden, dass die Anwendung seine Aufmerksamkeit erfordert. Im Regelfall wird der Befehl verwendet, wenn die Anwendung nicht im Vordergrund oder minimiert ist.

Der Deskriptor des Fensters, dass die Aufmerksamkeit des Anwenders erfordert, wird in (obj) angegeben. Folgende Konstanten können in (int1) angegeben werden:

WinFlashDefault Lässt die Titelzeile und das Icon in der Taskleiste blinken.

WinFlashTray Lässt das Icon in der Taskleiste blinken.

WinFlashCaption Lässt die Titelzeile der Anwendung blinken.

WinFlashUntilStop Fenster blinken lassen, bis WinFlashStop aufgerufen oder das Fenster beendet wird.

WinFlashUntilFore Fenster blinken lassen, bis es in den Vordergrund geholt wird.

WinFlashStop Blinken des Fensters stoppen.

Kontakt

Die Optionen _WinFlashDefault, _WinFlashTray und _WinFlashCaption können mit einer der Optionen _WinFlashUntilStop bzw. _WinFlashUntilFore kombiniert werden.

Ist weder die Option _WinFlashUntilStop noch _WinFlashUntilFore angegeben, bewirkt dies ein dreimaliges Aufblinken.

Wird (int1) nicht angegeben, wird _WinFlashDefault verwendet.

 Ist (obj) der Deskriptor eines MdiFrame, wirkt sich die Option _WinFlashTray nicht aus.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Im Argument (obj) wurde kein gültiger Deskriptor eines Fenster-Objektes

_ErrValueInvalid In (int1) wurde kein gültiger Wert angegeben.

WinHalt()



CONZEPT 16-Client beenden / Prozedur beenden


Siehe Alle
Befehle

Der Befehl WinHalt() beendet die aktuelle Prozedur und den gesamten CONZEPT 16-Client. Geladene Dialoge müssen zuvor über WinClose() geschlossen werden.

Sofern die Prozedur im Designer gestartet worden ist, wird nur die Prozedur abgebrochen und der Designer wieder aktiviert.



Der Befehl hat keine Auswirkung auf den Hauptprozess, wenn er in einem, per JobStart() gestarteten, Task bzw. Prozess ausgeführt wird.

WinIconPreload(alpha1) : 
int

Icons in den Cache laden

alpha1 Zu ladende Icons

Resultat int Anzahl geladener Icons

Siehe Icon

Normalerweise werden anzuzeigende Icons erst direkt vor der Anzeige im Dialog vom Datenbankserver geladen. Werden viele Icons auf einmal geladen, kann dies zu einem sichtbaren Aufbau der Anzeige führen. Mit diesem Befehl können darzustellende Icons direkt in den Icon-Cache geladen werden. Die Icons müssen dann später bei der Anzeige nicht mehr vom Datenbank-Server abgeholt werden, da diese dann bereits im Cache vorliegen.



Der Cache speichert maximal 2000 Icons. Ist der Cache vollständig gefüllt kehren weitere Aufrufe mit dem Resultat 0 zurück.


Im Argument (alpha1) wird dem Befehl eine kommaseparierte Liste der zu ladenden Icon-Namen übergeben. Diese Liste kann auch die Wildcard-Zeichen '*' und '?' enthalten.

Der Rückgabewert gibt die Anzahl der geladenen Icons an.

Beispiele:

```
// Icons mit den Namen Icon1, Icon2 und Icon3 in den Icon-Cache ladentLoaded # WinIconPreload('Icon1,Icon2,Icon3')
```

Wird der Befehl mit einer leeren Zeichenkette aufgerufen, wird der Icon-Cache geleert. Dies kann sinnvoll sein, da die im Cache vorliegenden Icons bis zur Beendigung des Clients im Cache verbleiben.

WinLayer(int1[, handle2[, int3[,  alpha4[, int5]]]) : int

Layer-Modus für Frame-Objekte

int1 Layer-Modus

handle2 Frame-Deskriptor


int3 Anzeigedauer

alpha4 Hinweistext

int5 Layer-Optionen

Resultat der Durchführung

ErrOK Der Befehl wurde erfolgreich durchgeführt

ErrInUse Der Befehl wurde wiederholt mit  _WinLayerStart aufgerufen

Resultat int ErrUnavailable Der Befehl kann nicht durchgeführt werden (siehe Text)

Siehe Verwandte Befehle

Der Befehl "friert" die Ausgabe eines vorgegebenen Frame-Objektes für eine bestimmte Zeitspanne ein. Änderungen an untergeordneten graphischen Objekten, während dieser Zeitspanne werden zwar durchgeführt, jedoch erst sichtbar nachdem die Zeitspanne abgelaufen ist, bzw. Layer-Modus abgeschaltet wird.

Für den Layer-Modus (int1) kann eine der folgenden Konstanten angegeben werden:

- WinLayerStart

Diese Option startet den Layer-Modus für das Frame-Objekt (handle2). Es kann sich zu einem bestimmten Zeitpunkt jeweils nur ein Frame im Layer-Modus befinden. Das Argument (int3) definiert die Zeitspanne in Millisekunden, bis zur Beendigung des Layer-Modus. Wird das Argument nicht übergeben oder ist es 0, wird eine Zeitspanne von einer Sekunde angenommen. Im Argument (alpha4) kann eine Meldung übergeben werden, die in der Mitte des Anwendungsfensters ausgegeben wird. So kann der Benutzer über den Vorgang informiert werden.

- WinLayerEnd

Diese Option beendet den Layer-Modus vor Ablauf der Zeitspanne. Es darf kein Frame-Deskriptor (handle2) angegeben werden. Alle anderen Argumente werden ignoriert.

- WinLayerUpdate

Wird das Frame-Objekt während des Layer-Modus in Position oder Größe

Kontakt

verändert, kann der Layer durch Aufruf mit dieser Option an das neue Fensterrechteck angepasst werden. Es darf kein Frame-Deskriptor (handle2) angegeben werden. Die übrigen Argumente können angegeben werden, um eine Änderung der Zeitspanne (int3) oder der Meldung (alpha4) durchzuführen. Das Argument (int5) wird ignoriert.

Im Argument (int5) können beim Modus (int1) WinLayerStart folgende Optionen angegeben werden:

WinLayerDarken Fenster abdunkeln

WinLayerDCross Fenster mit Kreuz-Schraffur überdecken

Diese Optionen können kombiniert werden.

War die Durchführung erfolgreich, liefert der Befehl ErrOk. Im Fehlerfall ist das Resultat ein negativer Wert. Wird der Fehlerwert zurückgegeben, liegt einer der folgenden Fälle vor:

- Es gibt keine Fensterrepräsentation des Frame auf dem Bildschirm. Dies ist z. B. im Ereignis EvtInit der Fall. Die Fensterrepräsentation wird durch WinCreate() bzw. WinDialogRun() erzeugt.
- Der Befehl wurde mit WinLayerStart / WinLayerEnd aufgerufen, es existiert jedoch kein Layer, entweder weil zuvor nicht WinLayerStart durchgeführt wurde oder weil der Layer bereits beendet wurde.

Beispiele:

```
// Während des Theme-Wechsels, Hinweis für 1 Sekunde aktivierenWinLayer(_WinLayerStart, $Frame, 1
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der Deskriptor (handle2) bezeichnet kein Frame-Objekt.

ErrRange Die angegebene Zeitspanne (int3) ist negativ oder größer als MaxInt Millisekunden.

ErrValueInvalid Der Wert für (int5) ist ungültig.

obj ->

WinPicSaveImage(alpha1,
handle2[, int3[,
int4[, int5]]]) : int



Bildinhalt speichern

obj Deskriptor des
Picture-Objektes

alpha1 Dateiname der
externen Datei

handle2 Deskriptor eines
Memory-Objektes
(optional)

int3 Optionen
(optional)

int4 Hintergrundfarbe
der Zeichnung
(optional)

int5 Qualitätsstufe
(optional)

Resultat int Fehlercode

Verwandte

Siehe Befehle,
SignMode

Dieser Befehl speichert die Zeichnung aus dem Picture-Objekt (handle) in der Datei (alpha1) oder einem Memory-Objekt (handle2). Wird im Argument (handle2) ein Wert ungleich 0 angegeben, wird der Dateiname (alpha1) ignoriert.

Das Argument (int3) definiert das Dateiformat und die zu speichernden Inhalte. Folgende Konstanten können angegeben werden:

- WinPicSignSaveAsExt - Format wird anhand des Dateinamens (alpha1) bestimmt. Dies ist nur möglich, wenn kein Memory-Objekt (handle2) angegeben wurde.
- WinPicSignSaveAsJpg - Zeichnung im JPEG-Format speichern.
- WinPicSignSaveAsPng - Zeichnung im PNG-Format speichern.
- WinPicSignSaveAsTif - Zeichnung im TIFF-Format speichern.
- WinPicSignSavePic - Inhalt des Picture-Objektes wird gespeichert.
- WinPicSignSaveSign - Zeichnung wird gespeichert.
- WinPicSignReset - Zeichnung wird gelöscht und nur das originale Bild im Picture-Objekt angezeigt.

Die Optionen WinPicSignSavePic, WinPicSignSaveSign und WinPicSignReset können miteinander und je einer der WinPicSignSaveAs...-Optionen kombiniert werden.

Wird das Bild extern als Datei gespeichert und die Option WinPicSignSaveAsExt ist nicht angegeben, wird die Dateiendung des ausgewählten Formats dem Dateinamen hinzugefügt.

Kontakt

Ist keine der Optionen _WinPicSignSavePic und _WinPicSignSaveSign angegeben, wird auch kein Bild erzeugt. Wird hingegen das Argument (int3) nicht angegeben oder 0 übergeben, werden die Optionen _WinPicSignSaveAsJpg, _WinPicSignSavePic und _WinPicSignSaveSign kombiniert.

Im Argument (int4) kann ein Farbwert (siehe _WinCol...) übergeben werden, mit dem der Hintergrund der Zeichnung gefüllt wird, wenn die Option _WinPicSignSavePic nicht angegeben ist. Wird _WinColUndefined angegeben, oder das Argument weggelassen, wird die Hintergrundfarbe vom Picture-Objekt verwendet. Für PNG- und TIFF-Dateien kann auch _WinColTransparent angegeben werden.

Das Argument (int5) definiert die Qualität des Bildes, wenn die Option _WinPicSignSaveAsJpg oder die Option _WinPicSignSaveAsExt in Kombination mit dem Dateityp '.jpg' angegeben ist. Es können Werte zwischen 1 und 100 verwendet werden. Standardmäßig wird eine Qualitätsstufe von 90 Prozent angewendet.

Resultat

Die Funktion gibt _ErrOk zurück, wenn die Zeichnung erfolgreich gespeichert wurde. Zusätzlich können folgende Fehlercodes zurückgegeben werden:

<u>_ErrGeneric</u>	Allgemeiner Fehler aufgetreten.
<u>_ErrFsiOpenFailed</u>	Externe Datei kann nicht geöffnet werden.
<u>_ErrFsiWriteFault</u>	Externe Datei kann nicht geschrieben werden.
<u>_ErrUnavailable</u>	Der Zeichnungsmodus ist nicht aktiv, die Dateiendung ist unbekannt oder es ist keine <u>_WinPicSignSaveAs...</u> -Konstante angegeben.
<u>_ErrMemExhausted</u>	Speicher nicht ausreichend.

Beispiel:

```
// Zeichnung inklusive Hintergrundbild als Datei Sign.jpg im temporären Verzeichnis speichern $p
```

Mögliche Laufzeitfehler:

<u>_ErrHdlInvalid</u>	Deskriptor des <u>Picture</u> -Objektes (obj) oder des <u>Memory</u> -Objektes (handle2) ist ungültig.
-----------------------	--



obj -> WinSearch(alpha1) : handle
 Suchen eines Objekts über den Namen
 obj Startobjekt der Suche
 alpha1 Name des zu suchenden
 Objekts

Resultat handle Deskriptor des Objekts

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

Liefert den Deskriptor des gesuchten Objekts. Die Wildcard-Operatoren '*' und '?' können im Objektname angegeben werden und werden bei der Suche entsprechend berücksichtigt. Als Startobjekt wird das Eltern-Objekt (Frame, GroupBox ...) des zu suchenden Objekts (alpha1) angegeben.

Als Resultat wird der Deskriptor des gefundenen Objekts zurückgegeben. Wurde kein Objekt gefunden ist das Resultat 0.



Unterobjekte von [ToolbarMenu](#)-Objekten können aus Kompatibilitätsgründen nicht gefunden werden. Siehe Kompatibilitätshinweise bei [ToolbarMenu](#).

Beispiel:

```
$Form->WinSearch('rlsAdress');$Form->WinSearch('rls*')
```

Mögliche Laufzeitfehler:

[_ErrHdlInvalid](#) Deskriptor ungültig

Kontakt

WinSearchClear()



Löschen des Suchpfades für Objekte

Verwandte

Siehe Befehle,

WinSearchPath()

Mit diesem Befehl wird ein mit WinSearchPath() gesetzter Suchpfad gelöscht.



obj -> WinSearchPath()

Setzen des Suchpfades für Objekte

obj Startobjekt des
Suchpfades

Verwandte

Siehe Befehle,
WinSearchPathGet(),
WinSearchClear()

Werden innerhalb von Prozeduren Referenzen auf Objekte mit dem Objektnamen (\$Name) verwendet, wird der Suchpfad benutzt, um das Objekt mit dem Namen im Objektbaum zu finden und den entsprechenden Deskriptor zu ermitteln. Die Suche wird dabei von einem bestimmten Objekt innerhalb des Suchbaums gestartet und durchsucht alle untergeordneten Objekte. Wird das Objekt nicht gefunden, werden auch die anderen Bäume durchsucht. Die Suche wird beendet, sobald das erste Objekt mit dem entsprechenden Namen gefunden wird.

Dieses Startobjekt bestimmt wie lange die Suche dauert, wenn das Objekt gefunden werden kann. Der Suchpfad wird automatisch gesetzt, wenn ein Ereignis ausgelöst oder ein Frame-Objekt geladen wird. Alle Namensreferenzen werden dann innerhalb des Fenster-Objekts aufgelöst.

In einer Applikation mit MDI-Fenstern wird nicht nur das aktive Fenster durchsucht, sondern auch alle anderen MDI-Fenster, wenn das angegebene Objekt nicht im aktiven MDI-Fenster gefunden wurde. Dies verlängert die Suche. Die Suche kann über das Flags WinAppSearchMdiFrame des Applikation-Objekts auf das aktuelle MDI-Fenster beschränkt werden. Der Suchpfad muss dann nicht gesetzt werden.

Mit dem Befehl wird das Startobjekt des Suchpfades 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.

Durch die Angabe eines Suchpfades wird das Startobjekt (obj) für die Suche festgelegt. Bei der Suche werden ebenfalls die Unterobjekte des Startobjektes durchlaufen. Im Beispiel werden zwei Dialoge geladen, die beide ein Objekt mit dem Namen "Text" besitzen. Die Identifizierung des Objektes mit \$Text ist nicht eindeutig. Durch die Angabe eines Suchpfades mit dem entsprechenden Dialog wird das Objekt eindeutig gefunden.

Nach dem die Objekte gefunden wurden, muss der Suchpfad mit der Anweisung WinSearchClear() wieder zurückgesetzt werden.

Beispiel:

```
WinOpen('Meldung1', _WinOpenDialog);WinOpen('Meldung2', _WinOpenDialog);$Meldung1->WinSearchPath
```



Namensreferenzen innerhalb von Prozeduren werden für jeden Aufruf nur einmal aufgelöst. D. h. wurde in dem obigen Beispiel der Deskriptor des Objektes Text bereits zuvor ermittelt, wird unabhängig vom gesetzten Suchpfad, in der Folge der Prozedur immer das gleiche Objekt angesprochen. Soll ein anderes Objekt

Kontakt

mit gleichen Namen angesprochen werden, muss der Deskriptor des Objektes mit dem Befehl WinSearch() ermittelt werden.

Kontakt



WinSearchPathGet() : handle

Ermitteln des Suchpfades für Objekte

Resultat handle Deskriptor des Objektes

Verwandte Befehle,


Siehe WinSearchPath(),

WinSearchClear()

Mit diesem Befehl kann ein mit WinSearchPath() gesetzter Suchpfad abgefragt werden. Es wird der Deskriptor des Objektes zurückgegeben.


Beispiel:

```
WinOpen('Message1', _WinOpenDialog);WinOpen('Message2', _WinOpenDialog);tHdlSearchPath # WinSearchPath
```

obj -> WinShutdownBlock([alpha1]) : int; 
 Beenden der Windows-Sitzung verhindern
 obj Deskriptor des Fenster-Objekts
 alpha1 Blockierungsgrund (optional)

Fehlerwert

ErrOk Kein Fehler aufgetreten

Resultat int ErrUnavailable Die Funktion steht nicht 
 zur Verfügung.

ErrGeneric Unerwarteter Fehler.


Siehe Verwandte Befehle, EvtEndSession

Mit dem neuen Befehl WinShutdownBlock kann verhindert werden, dass Windows die Sitzung beendet. Er sollte nur dann verwendet werden, wenn eine Operation in Verarbeitung ist, die abgeschlossen werden muss.

 Dieser Befehl verhindert nicht das Beenden des Clients mittels Taskmanager.

Als (obj) muss der Deskriptor auf ein Fenster-Objekt (Frame, AppFrame, MdiFrame oder TrayFrame) übergeben werden. Die Fenster-Erstellung muss bereits abgeschlossen sein. Dies ist der Fall, wenn das Ereignis EvtCreated durchlaufen wurde.

Das optional Argument (alpha1) enthält die Nachricht, die im Abmeldebildschirm von Windows bei der blockierenden Anwendung angezeigt wird.

 Die Nachricht sollte möglichst kurz und prägnant sein.

Fehlt das Argument (alpha1) oder ist es leer, wird die Blockierung für das Fenster (obj) wieder aufgehoben.

Der Rückgabewert der Anweisung kann mit folgenden Konstanten verglichen werden:

ErrOk Der Befehl wurde erfolgreich durchgeführt.

ErrUnavailable Die Funktion steht nicht zur Verfügung. Für die Ausführung wird Windows Vista vorausgesetzt.

ErrGeneric Es ist ein unerwarteter Fehler aufgetreten.

Beispiele:

```
// Beenden blockieren$Frame->WinShutdownBlock('Datenspeicherung abschließen');// Operation durchf
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig, ist kein Fenster-Objekt oder die Blockierung für das Fenster-Objekt wurde bereits aufgehoben.

WinSleep(int1[, int2])



Prozedurverarbeitung anhalten

int1 Anhaltedauer

int2 Nachrichtenmodus

Verwandte Befehle,

Siehe WinHalt(),

SysSleep()

Diese Funktion hält die Prozedurausführung für die in (int1) angegebene Zeitspanne in Millisekunden an.


Im Gegensatz zu dem Befehl SysSleep() wird die Nachrichtenschleife von Windows abgefragt und neu eingetroffene Ereignisse bearbeitet. Ist die Zeitspanne abgelaufen oder keine Windows-Nachricht vorhanden, kehrt der Befehl zurück und setzt die Prozedurausführung fort.

Über das optionale Argument (int2) kann definiert werden, welche Nachrichten verarbeitet werden. Nachrichten, die nicht verarbeitet werden, verbleiben in der Nachrichtenschleife, um sie zu einem späteren Zeitpunkt verarbeiten zu können. Folgende Konstanten können angegeben werden:

<u>_WinMsgAll</u>	Es werden alle von Windows gesendeten Nachrichten verarbeitet (default).
<u>_WinMsgNoKeyboardInput</u>	Es werden keine Tastatureingabe-Nachrichten verarbeitet.
<u>_WinMsgNoMouseInput</u>	Es werden keine Mauseingabe-Nachrichten verarbeitet.
<u>_WinMsgNoInput</u>	Es werden keine Tastatur- und Mauseingabe-Nachrichten verarbeitet.

Beispiel:

Im Ereignis EvtChanged eines Eingabe-Objektes wird während einer längeren Verarbeitung WinSleep aufgerufen, damit eingehende Windows-Nachrichten verarbeitet werden können. Hierzu zählen auch Tastatureingabe-Nachrichten. Macht der Anwender während der Durchführung des Ereignisses EvtChanged Tastatureingaben, würde dies zur Ausführung des Ereignisses EvtChanged führen. Da dieses jedoch bereits ausgeführt wird, kann die Eingabe nicht verarbeitet werden. Die Eingabe wird ignoriert und nicht an das Eingabeobjekt weitergeleitet. Durch den Aufruf von WinSleep(1, _WinMsgNoKeyboardInput) im Ereignis EvtChanged werden Tastaturnachrichten ignoriert und verbleiben in der Warteschlange. Dadurch gehen keine Tastatureingaben verloren.

obj ->
WinUpdate([int1[, 
int2]]) : int
Objekt aktualisieren
obj Objekt
int1 Update-Modus
 (siehe Text)
 Optionen für
int2 RecList- und
 Frame-Objekte
 (siehe Text)
Resultat int Fehlerwert
 Verwandte
Siehe Befehle,
 WinRvwUpdate()


Mit diesem Befehl wird ein Objekt aktualisiert. Sind dem übergebenen Objekt weitere Objekte untergeordnet, werden diese ebenfalls aktualisiert. Sind in einem Fenster-Objekt RecList- oder DataList-Objekte enthalten, werden diese aus Performancegründen nicht aktualisiert. Diese Objekte müssen gesondert aktualisiert werden.

Wird ein RecList-Objekt übergeben, werden alle in dem Objekt angezeigten Datensätze neu gelesen. Bei allen anderen Objekten bewirkt WinUpdate() die Anzeige der mit dem Objekt (oder dessen Unterobjekte) verknüpften Datenbankfelder. Die gleiche Wirkung wird mit WinUpdate(_WinUpdFld2Obj) erzielt.

Folgende Parameter können als Update-Modus (int1) übergeben werden:

_WinUpdOn Setzt die AutoUpdate-Eigenschaft von (obj) auf true
_WinUpdOff Setzt die AutoUpdate-Eigenschaft von (obj) auf false
_WinUpdFld2Obj Überträgt den Inhalt der Feldpuffer in das Objekt
_WinUpdObj2Fld Überträgt den Inhalt des Objektes in den Feldpuffer
_WinUpdFld2Buf Überträgt den Inhalt der Feldpuffer in den Datensatzpuffer des Objekts
_WinUpdBuf2Obj Überträgt den Inhalt des Textpuffers in das Objekt TextEdit
_WinUpdObj2Buf Überträgt den Inhalt des Objektes TextEdit in den Textpuffer
_WinUpdActivate Holt das Frame-Objekt in den Vordergrund
_WinUpdState Status setzen
_WinUpdSort DataList-Objekt neu sortieren

Die Konstanten können nicht miteinander kombiniert werden. Die Konstanten _WinUpdOff und _WinUpdOn können mit _WinUpdScrollPos kombiniert werden, um die Scrollposition zu speichern und wieder herzustellen.

 Wird als Update-Modus (int1) _WinUpdOn mit der Konstanten _WinUpdScrollPos kombiniert, wird nur die Scrollposition wiederhergestellt. Zur Aktualisierung des Objektes muss WinUpdate() erneut mit _WinUpdOn aufgerufen werden.

Für die Objekte RecList, DataList, Frame und GanttGraph können weitere Optionen in (int2) angegeben werden.

Optionen für RecList- und DataList-Objekte (in (int1) muss einer der obigen Parameter angegeben werden):

<u>WinLstFromFirst</u>	Neuaufbau der Liste ab der ersten Zeile
<u>WinLstFromLast</u>	Neuaufbau der Liste ab der letzten Zeile
<u>WinLstRecFromBuffer</u>	Neuaufbau der Liste ausgehend von den Werten des Feldpuffers
<u>WinLstRecFromRecId</u>	Neuaufbau der Liste ausgehend von der Datensatz-ID
<u>WinLstFromSelected</u>	Neuaufbau der Liste ausgehend von der selektierten Zeile
<u>WinLstFromTop</u>	Neuaufbau der Liste ab der ersten sichtbaren Zeile
<u>WinLstPosTop</u>	Selektierte Zeile wird oben in der Liste angezeigt
<u>WinLstPosMiddle</u>	Selektierte Zeile wird in der Mitte der Liste angezeigt
<u>WinLstPosBottom</u>	Selektierte Zeile wird unten in der Liste angezeigt
<u>WinLstPosSelected</u>	Selektierte Zeile behält die Position innerhalb der Liste
<u>WinLstRecDoSelect</u>	In der Liste wird der aktuelle Datensatz selektiert
<u>WinLstReorderColumns</u>	Nummeriert alle sichtbaren Spalten neu (siehe <u>ClmOrder</u>)
<u>WinLstRecEvtSkip</u>	Unterbinden des Ereignisses <u>EvtLstSelect</u>
<u>WinLstIgnoreHidden</u>	Die Liste wird auch aktualisiert, wenn sie unsichtbar (<u>Visible</u> = <u>false</u>) ist.
<u>WinLstActiveOnly</u>	Nur das aktive View der <u>RecList</u> oder <u>DataList</u> aktualisieren.

Konstanten aus den Bereichen WinLst*From*, WinLstPos*, WinLstReorderColumns und WinLstRec* können miteinander kombiniert werden. Konstanten aus gleichen Bereichen können nicht kombiniert werden.

Optionen für Frame-Objekte (in (int1) muss WinUpdState angegeben werden):

<u>WinDialogMaximized</u>	Das Fenster wird maximiert dargestellt.
<u>WinDialogMinimized</u>	Das Fenster wird minimiert dargestellt.
<u>WinDialogNormal</u>	Das Fenster wird weder maximiert noch minimiert dargestellt.
<u>WinDialogPrimary</u>	Befindet sich das Fenster außerhalb des darstellbaren Bereichs, wird es in der Bildschirmmitte des primären Bildschirms angezeigt.

Die Option WinDialogPrimary kann mit den anderen Konstanten kombiniert werden.

Optionen für GanttGraph-Objekte:

<u>WinGntRefresh</u>	Gantt-Graphen neu zeichnen
<u>WinGntRecalc</u>	Gantt-Graphen neu berechnen

Änderungen in Eingabeobjekten werden in den angegebenen Feldpuffer (Eigenschaft DbFieldName) übertragen, wenn der Fokus in ein anderes Objekt gesetzt wird. Diese Übertragung kann ebenfalls prozedural mit dem Parameter WinUpdObj2Fld durchgeführt werden.

Die Optionen WinLstFrom... beeinflussen nur die Anzeige der Datensätze. Der selektierte Datensatz bleibt unverändert. Die Selektion kann mit der Option WinLstRecDoSelect neu gesetzt werden. Ein Update mit dieser Option löst dann auch

Kontakt

das Ereignis EvtLstSelect der betreffenden RecList aus.

Die Position innerhalb der dargestellten Datensätze wird durch die Option _WinLstPos... bestimmt. Diese Option lässt sich mit den anderen Optionen kombinieren. Sollte eine Darstellung nicht möglich sein, weil zum Beispiel der letzte Datensatz in der Mitte dargestellt werden soll, erfolgt die Darstellung als vollständig gefüllte Liste.

Soll ein Fenster maximiert oder minimiert werden ist in (int1) die Konstante _WinUpdState anzugeben.

Als Rückgabewerte wird der Fehlerwert _ErrHdlInvalid zurückgegeben, wenn als (obj) kein gültiger Deskriptor angegeben wurde, sonst _ErrOk.

Beispiele:

```
sub EvtLstSelect( aEvt    : event;    // Ereignis  aID      : bigint;    // Record-ID des Datensatze
```

obj -> WinUserEvent(word1[, int2[,
alpha3]])



Benutzerdefiniertes Ereignis auslösen

obj Fenster-Objekt

word1 Ereignis-Id

Argument des

int2 Benutzers

(optional)

Argument des

alpha3 Benutzers

(optional)

Verwandte

Siehe Befehle,

EvtUser

Mit dieser Anweisung wird ein Ereignis in die Ereigniswarteschlange des Betriebssystems eingefügt. Das Ereignis wird zu einem späteren Zeitpunkt aufgerufen. Zu welchem Zeitpunkt das Ereignis ausgelöst wird, kann nicht genau bestimmt werden, es wird aber nach dem Ereignis, in dem die Anweisung durchgeführt wurde, ausgelöst.

Als Objekt (obj) muss ein Fenster-Objekt übergeben werden, bei dem das Ereignis EvtUser angegeben ist. Als (int1) wird eine Benutzerdefinierte Ereignis-ID übergeben. Die ID muss im Bereich 1 - 65535 liegen und kann vom Programmierer frei gewählt werden. Die ID wird dem Ereignis übergeben und kann zur Unterscheidung mehrerer Ereignisse verwendet werden.

Die Parameter (int2) und (alpha3) können vom Benutzer frei gewählt werden. Der Inhalt wird an das Ereignis weitergereicht.

Beispiel:

```
define{ _EventUser : 10000}...tHdlFrame->WinUserEvent(_EventUser, 42, 'Answer to the ultimate qu
```

obj -> WinWbnExecCommand(int1)



Kommando an ein WebNavigator-Objektes senden

obj Objekt

(WebNavigator-Objekt)

int1 Kommando

Siehe Verwandte Befehle

Mit diesem Befehl wird das Kommando (int1) an ein WebNavigator-Objekt gesendet. Der Deskriptor des WebNavigator-Objektes wird in (obj) übergeben.

Im Parameter (int1) können folgende Optionen übergeben werden:

- **_WinWbnCommandExecPrint (41)**

Zeigt den Druckerauswahldialog an. Bei Klick auf "OK" wird die Seite gedruckt.

- **_WinWbnCommandNavHome (5)**

Navigiert zur Startseite, die in den Internet-Optionen eingestellt ist.

- **_WinWbnCommandNavBack (6)**

Navigiert zur vorherigen Seite.

- **_WinWbnCommandNavForward (7)**

Navigiert zur nächsten Seite.

- **_WinWbnCommandNavSearch (8)**

Navigiert zur Suchseite, die im Internet-Explorer eingestellt ist.

- **_WinWbnCommandSelCopy (1)**

Kopiert den aktuell selektierten HTML-Inhalt bzw. den Inhalt des aktuell fokussierten Objektes in die Zwischenablage.

- **_WinWbnCommandSelCut (2)**

Schneidet die aktuelle Selektion (z. B. in einem Eingabefeld) aus und überträgt ihn in die Zwischenablage.

- **_WinWbnCommandSelDelete (31)**

Löscht den selektierten Text oder das Zeichen an der Cursor-Position (z. B. in einem Eingabefeld).

- **_WinWbnCommandSelPaste (3)**

Fügt den Inhalt der Zwischenablage in das fokussierte Objekt ein.

- **_WinWbnCommandSelSelectAll (4)**

Selektiert den gesamten HTML-Inhalt bzw. den Inhalt des fokussierten Objektes.

- **_WinWbnCommandUpdNoCache (10)**

Seite erneut anfordern (entspricht  +  im Internet Explorer).

- **_WinWbnCommandUpdNormal (9)**

Seite erneut laden (entspricht  im Internet Explorer).

- **_WinWbnCommandZoomIn (11)**

Seiten-Zoom um 25% vergrößern (entspricht  +  im Internet Explorer).

- **_WinWbnCommandZoomOut (12)**

Seiten-Zoom um 25% verkleinern (entspricht  +  im Internet Explorer).

- **_WinWbnCommandZoomReset (13)**

Seiten-Zoom auf 100% setzen (entspricht  +  im Internet Explorer).



Die Kommandos werden unter folgenden Bedingungen ignoriert:

- **Es wird kein HTML-Inhalt angezeigt** - Der WebNavigator ist leer oder aber es wird z. B. der Inhalt eines Verzeichnisses angezeigt.
- **Das Kommando ist nicht verfügbar** - z. B. bei _WinWbnCommandSelCut / _WinWbnCommandSelDelete, wenn kein Ausschneiden oder Löschen möglich ist oder bei _WinWbnCommandNavForward, wenn noch keine nachfolgende Seite existiert.

Mögliche Laufzeitfehler

_ErrHdlInvalid Bei (obj) handelt es sich nicht um ein WebNavigator-Objekt.

_ErrValueInvalid In Argument (int1) wurde keines der gültigen Kommandos angegeben.

with



Überprüfung der Objektnamen bei der Übersetzung

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

Das with-Konstrukt ist ein Sprachelement, welches in Verbindung mit dem Operator \$: (nicht zu verwechseln mit \$) das Vorhandensein von Oberflächen-Objekten bereits zur Übersetzungszeit sicherstellt.

Wird ein Deskriptor mit dem \$-Zeichen referenziert, erfolgt die Überprüfung erst zur Laufzeit der Prozedur. Kann kein Objekt mit dem angegebenen Namen ermittelt werden, erfolgt der Laufzeitfehler "Deskriptor ungültig" und die Prozedur wird abgebrochen.

Mit dem with-Konstrukt und dem \$:-Operator erfolgt die Überprüfung bereits zum Zeitpunkt der Übersetzung.

Syntax:

```
with <Dialogname> $:<Objektname>->...
```

Existiert das Objekt oder der Dialog nicht, wird die Übersetzung mit dem Fehler "Name unbekannt" abgebrochen. Als Dialog können folgende Objekte angegeben werden:

- **Frame**
- **AppFrame**
- **MdiFrame**
- **PrintForm**
- **PrintFormList**
- **PrintDoc**
- **PrintDocRecord**

Sollen mehrere Anweisungen innerhalb des with-Konstrukts verarbeitet werden, müssen diese durch die geschweiften Klammern, in der üblichen Weise geklammert werden.

Beispiele:

```
// Überprüfung eines Objekts with FrmKunden $:edKndNummer->wpCaptionInt # 0; // Überprüfung bei meh
```

with-Konstrukte können geschachtelt werden:

```
with FrmKunden{ ... with FrmSelectionDialog { ... $:rlsSelection->wp... ... } ... }
```

Ist das Objekt rlsSelection im Dialog FrmSelectionDialog nicht vorhanden, wird dieses im Dialog FrmKunden gesucht. Erst wenn es auch dort nicht vorhanden ist, erfolgt ein Übersetzungsfehler. Die Suche erfolgt immer vom inneren zum äußeren with-Block.

Mit dem with-Konstrukt werden auch die Namensreferenzen auf Dialog-Objekte schneller aufgelöst. Wird während der Übersetzung eine \$:-Referenz gefunden, wird der Pfad zu dem entsprechenden Objekt gespeichert. Zur Laufzeit muss nicht mehr

Kontakt

das ganze geladene Objekt durchsucht werden, statt dessen erfolgt die Ermittlung des Deskriptors direkt über den gespeicherten Pfad.

Wurde die Datenbank von einem Stand vor Version 5.0 konvertiert, muss ein Dialog mit der aktuellen Version geöffnet und wieder gespeichert werden, damit die with-Anweisung verwendet werden kann.

Konstanten für Sonstige Befehle
Konstanten für Sonstige Befehle
Siehe Sonstige
Befehle

- WinBeepDefault
- WinBeepError
- WinBeepInformation
- WinBeepQuestion
- WinBeepWarning
- WinDialogMaximized
- WinDialogMinimized
- WinDialogNormal
- WinDialogPrimary
- WinFlashCaption
- WinFlashDefault
- WinFlashStop
- WinFlashTray
- WinFlashUntilFore
- WinFlashUntilStop
- WinGntRecalc
- WinGntRefresh
- WinImageFormatBmp
- WinImageFormatJpg
- WinImageFormatPng
- WinImageFormatTif
- WinLayerDarken
- WinLayerEnd
- WinLayerStart
- WinLayerUpdate
- WinLstActiveOnly
- WinLstEditClearChanged
- WinLstEditLst
- WinLstEditLstAlpha
- WinLstFromFirst
- WinLstFromLast
- WinLstFromSelected
- WinLstFromTop
- WinLstIgnoreHidden
- WinLstPosBottom
- WinLstPosMiddle
- WinLstPosSelected
- WinLstPosTop
- WinLstRecDoSelect
- WinLstRecEvtSkip
- WinLstRecFromBuffer
- WinLstRecFromRecId
- WinLstReorderColumns
- WinMsgAll
- WinMsgNoInput
- WinMsgNoKeyboardInput

- WinMsgNoMouseInput
- WinPicSignReset
- WinPicSignSaveAsExt
- WinPicSignSaveAsJpg
- WinPicSignSaveAsPng
- WinPicSignSaveAsTif
- WinPicSignSavePic
- WinPicSignSaveSign
- WinRvwEditAbortEditing
- WinRvwUpdateDoKeepSelect
- WinRvwUpdateDoSelect
- WinRvwUpdateFromFirst
- WinRvwUpdateFromLast
- WinRvwUpdateFromRecBuf
- WinRvwUpdateFromSelected
- WinRvwUpdateFromTop
- WinRvwUpdateFromTopLocation
- WinRvwUpdateOptClearCache
- WinRvwUpdateOptClearSelected
- WinUpdActivate
- WinUpdBuf2Obj
- WinUpdFld2Buf
- WinUpdFld2Obj
- WinUpdObj2Buf
- WinUpdObj2Fld
- WinUpdOff
- WinUpdOn
- WinUpdSort
- WinUpdState
- WinWbnCommandExecPrint
- WinWbnCommandNavBack
- WinWbnCommandNavForward
- WinWbnCommandNavHome
- WinWbnCommandNavSearch
- WinWbnCommandSelCopy
- WinWbnCommandSelCut
- WinWbnCommandSelDelete
- WinWbnCommandSelPaste
- WinWbnCommandSelSelectAll
- WinWbnCommandUpdNoCache
- WinWbnCommandUpdNormal
- WinWbnCommandZoomIn
- WinWbnCommandZoomOut
- WinWbnCommandZoomReset

_WinBeepDefault

Systemklang für "Standardton Warnsignal" wiedergeben

Wert 0

Siehe WinBeep()

Option bei WinBeep() durch die der Systemklang für "Standardton Warnsignal" wiedergegeben werden kann.

_WinBeepError

Systemklang für "Kritischer Fehler" wiedergeben

Wert 3

Siehe WinBeep()

Option bei WinBeep() durch die der Systemklang für "Kritischer Fehler" wiedergegeben werden kann.

_WinBeepInformation

Systemklang für "Sternchen" wiedergeben

Wert 2

Siehe WinBeep()

Option bei WinBeep() durch die der Systemklang für "Sternchen" wiedergegeben werden kann.

_WinBeepQuestion

Systemklang für "Frage" wiedergeben

Wert 5

Siehe WinBeep()

Option bei WinBeep() durch die der Systemklang für "Frage" wiedergegeben werden kann.

_WinBeepWarning

Systemklang für "Hinweis" wiedergeben

Wert 4

Siehe WinBeep()

Option bei WinBeep() durch die der Systemklang für "Hinweis" wiedergegeben werden kann.

_WinDialogMaximized
Fenster wird maximiert dargestellt
Wert 16 / 0x00000010

Verwandte

Befehle,

WinDialog(),

Siehe WinDialogRun(),

WinAdd(),

WinAddByName(),

WinUpdate(),

WinInfo()

Je nach Befehl hat _WinDialogMaximized folgende Bedeutung:

- WinDialog(), WinDialogRun(), WinAdd() und WinAddByName()

Dialog wird maximiert geladen.

- WinUpdate()

Dialog wird zur Laufzeit maximiert. Die Option kann nur angegeben werden, wenn _WinUpdState als ersten Parameter übergeben wird.

- WinInfo()

Liefert WinInfo(_WinState) den Wert _WinDialogMaximized zurück, ist der Dialog maximiert.

_WinDialogMinimized
Fenster wird minimiert dargestellt
Wert 32 / 0x00000020

Verwandte
Befehle,
WinDialog(),
Siehe WinDialogRun(),
WinAdd(),
WinAddByName(),
WinUpdate(),
WinInfo()

Je nach Befehl hat _WinDialogMinimized folgende Bedeutung:

- WinDialog(), WinDialogRun(), WinAdd() und WinAddByName()

Dialog wird minimiert geladen.

- WinUpdate()

Dialog wird zur Laufzeit minimiert. Die Option kann nur angegeben werden, wenn _WinUpdState als ersten Parameter übergeben wird.

- WinInfo()

Liefert WinInfo(_WinState) den Wert _WinDialogMinimized zurück ist der Dialog minimiert.

_WinDialogNormal
Fenster weder maximiert noch minimiert

Wert 64 /
0x00000040

Siehe WinUpdate(),
WinInfo()

Je nach Befehl hat _WinDialogNormal folgende Bedeutung:

- WinDialog() und WinDialogRun()

Der Dialog wird weder minimiert, noch maximiert angezeigt. Ist durch die Darstellung das Fenster in keinem Bildschirm sichtbar, wird es im primären Bildschirm zentriert dargestellt.

- WinUpdate()

Dialog wird zur Laufzeit von einem minimierten oder maximierten Status zurückgesetzt. Die Option kann nur angegeben werden, wenn _WinUpdState als ersten Parameter übergeben wird.

- WinInfo()

Liefert WinInfo(_WinState) den Wert _WinDialogNormal zurück ist der Dialog weder maximiert noch minimiert.

_WinDialogPrimary

Fenster wird auf dem Primär-Bildschirm zentriert

Wert 128 /
0x00000080

Verwandte

Siehe Befehle,

WinUpdate()

Ist diese Konstante bei WinUpdate() angegeben und befindet sich das AppFrame- oder Frame-Objekt in einem nicht angezeigten Bereich, wird das Fenster auf der Bildschirmmitte des Primärbildschirms dargestellt.

Die Anweisung wird ignoriert, wenn sich das Fenster (auch nur teilweise) im darstellbaren Bereich eines Bildschirms befindet.

WinFlashCaption
Titelzeile blinken lassen
Wert 1 / 0x0001

WinFlash(),
Siehe WinFlashTray,
WinFlashDefault

Option bei WinFlash(), mit der die Titelzeile des angegebenen Fensters blinkt.

WinFlashDefault
Titelzeile und Taskleisten-Icon blinken lassen
Wert 3 / 0x0003

WinFlash(),
Siehe WinFlashCaption,
WinFlashTray

Option bei WinFlash(), mit der die Titelzeile und das Icon in der Taskleise des angegebenen Fensters blinken.

_WinFlashStop

Blinken des Fensters stoppen

Wert 0 / 0x0000

WinFlash(),

Siehe _WinFlashUntilStop,

_WinFlashUntilFore

Option bei WinFlash(), mit der das Blinken des Fenster gestoppt wird, unabhängig davon mit welcher Option WinFlash() initiiert wurde.

WinFlashTray
Taskleisten-Icon blinken lassen
Wert 2 / 0x0002

WinFlash(),
Siehe WinFlashCaption,
WinFlashDefault

Option bei WinFlash(), mit der das Icon in der Taskleise des angegebenen Fensters blinkt.

WinFlashUntilFore

Fenster blinken lassen, bis es in den Vordergrund geholt wird

Wert 12 / 0x000C

WinFlash(),

Siehe WinFlashUntilStop,

WinFlashStop

Option bei WinFlash(), mit der das Fenster so lange blinkt, bis es in den Vordergrund geholt wird.

_WinFlashUntilStop

Fenster blinken lassen, bis WinFlash() mit _WinFlashStop aufgerufen oder das Fenster beendet wird

Wert 4 / 0x0004

WinFlash(),

Siehe _WinFlashStop,

_WinFlashUntilFore

Option bei WinFlash(), mit der das Fenster so lange blinkt, bis WinFlash() mit der Option _WinFlashStop aufgerufen oder das Fenster beendet wird.

_WinGntRecalc
GanttGraph-Objekt neu berechnen
Wert 1 / 0x00000001

WinUpdate(),
Siehe _WinUpdState,
_WinGntRefresh

Mit der Anweisung \$GanttGraph->WinUpdate(_WinUpdState, _WinGntRecalc) wird das GanttGraph-Objekt neu berechnet und anschließend die Achsen und die View-Bereiche neu gezeichnet.

Eine neue Berechnung des GanttGraph-Objektes wird notwendig, wenn eine der folgenden Eigenschaften der Achse geändert wurden:

- Size
- SizeScala
- TitleText

Die Aktualisierung mit der Option _WinGntRecalc schließt die Aktualisierung mit der Option _WinGntRefresh mit ein.

Die Anweisung wird nicht durchgeführt, wenn die Eigenschaft AutoUpdate auf false gesetzt ist.

_WinGntRefresh
GanttGraph-Objekt neu zeichnen
Wert 0 / 0x00000000

WinUpdate(),
Siehe _WinUpdState,
_WinGntRecalc

Mit der Anweisung \$GanttGraph->WinUpdate(_WinUpdState, _WinGntRefresh) wird das GanttGraph-Objekt neu gezeichnet.

Eine Aktualisierung des Objektes wird notwendig, wenn eine Eigenschaft einer Achse verändert wurde. Findet eine Änderung der Größe der Achse statt, muss der Parameter _WinGntRecalc verwendet werden. Die Größe wird beim Setzen der Eigenschaften Size, SizeScala und TitleText verändert.

Die Anweisung wird nicht durchgeführt, wenn die Eigenschaft AutoUpdate auf false gesetzt ist.

WinImageFormatBmp
Windows-Bitmap-Datei erstellen
Wert 2

WinBarcodeSaveImage(),
Siehe WinImageFormatJpg,
WinImageFormatPng,
WinImageFormatTif

Option bei WinBarcodeSaveImage(), durch die der Inhalt eines Barcode-Objektes als Windows-Bitmap-Datei gespeichert wird.

WinImageFormatJpg

JPEG-Datei erstellen

Wert 3

WinBarcodeSaveImage(),

Siehe WinImageFormatBmp,

WinImageFormatPng,

WinImageFormatTif

Option bei WinBarcodeSaveImage(), durch die der Inhalt eines Barcode-Objektes als JPEG-Datei gespeichert wird.

WinImageFormatPng

PNG-Datei erstellen

Wert 5

WinBarcodeSaveImage(),

Siehe WinImageFormatBmp,

WinImageFormatJpg,

WinImageFormatTif

Option bei WinBarcodeSaveImage(), durch die der Inhalt eines Barcode-Objektes als PNG-Datei gespeichert wird.

WinImageFormatTif

TIFF-Datei erstellen

Wert 4

WinBarcodeSaveImage(),

Siehe WinImageFormatBmp,

WinImageFormatJpg,

WinImageFormatPng

Option bei WinBarcodeSaveImage(), durch die der Inhalt eines Barcode-Objektes als TIFF-Datei gespeichert wird.

_WinLayerDarken
Fenster abdunkeln
Wert 1 / 0x00000001

Verwandte

Siehe Befehle,
WinLayer(),
_WinLayerDCross

Option bei WinLayer() um beim Layer-Modus eines Fensters den Anwendungsbildschirm etwas abzudunkeln.

WinLayerDCross
Fenster mit diagonalen Kreuz-Schraffur überdecken
Wert 2 / 0x00000002

Verwandte

Siehe Befehle,
WinLayer(),
WinLayerDarken

Option bei WinLayer() um beim Layer-Modus das Fenster mit einer diagonalen Kreuz-Schraffur zu überdecken.

_WinLayerEnd
Layer-Modus für Fenster beenden

Wert 1 /
0x00000001

Verwandte

Siehe Befehle,

WinLayer()

Option bei WinLayer() um den Layer-Modus eines Fensters zu beenden.

WinLayerStart
Layer-Modus für Fenster aktivieren
Wert 0 /
0x00000000

Verwandte

Siehe Befehle,

WinLayer()

Option bei WinLayer() um den Layer-Modus eines Fensters zu aktivieren.

_WinLayerUpdate
Fenster im Layer-Modus aktualisieren
Wert 2 /
0x00000002

Verwandte

Siehe Befehle,

WinLayer()

Option bei WinLayer() um ein Fenster im Layer-Modus zu aktualisieren.

WinLstActiveOnly
Nur das aktive View aktualisieren
Wert -2.147.483.648
/ 0x80000000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Aktualisiert bei RecList- und DataList-Objekten mit mehreren Views nur das aktive View.

WinLstEditLst

Liste mit einer Spalte im Eingabe-Objekt erzeugen

Wert 1 / 0x00000001

Verwandte

Befehle,

Siehe

WinLstEdit(),

EvtLstEditStart

Option bei WinLstEdit() durch die im Eingabe-Objekt der Liste eine Liste mit einer Spalte erzeugt werden kann.

Die Liste kann im Ereignis EvtLstEditStart mit Einträgen gefüllt werden.

WinLstEditClearChanged
Änderungs-Flag zurücksetzen

Wert 16.777.216 /
0x01000000

Vewandte

Siehe Befehle,

WinLstEdit()

Option bei WinLstEdit() durch die das Änderungs-Flag zurückgesetzt werden kann.

Das Setzen des Flags erfolgt nach dem Ausführen des Ereignisses EvtLstEditCommit, wenn dieses Ereignis den Wert true zurückgibt. Der Inhalt des Flags wird im Ereignis EvtLstEditFinished übergeben und kann dort ausgewertet werden.

WinLstEditLstAlpha

Liste mit zwei Spalten in Eingabe-Objekt erzeugen

Wert 2 / 0x00000002

Verwandte

Siehe Befehle,

WinLstEdit(),

EvtLstEditStart

Option bei WinLstEdit() durch das im Eingabe-Objekt der Liste eine Liste mit zwei Spalten erzeugt werden kann.

Die Liste kann im Ereignis EvtLstEditStart mit Einträgen gefüllt werden.

WinLstFromFirst
Ab dem ersten Datensatz
Wert 1 /
0x00000001

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Liest ab dem ersten Datensatz der Datei die Datensätze der RecList neu ein. Es werden nur soviele Sätze gelesen, wie in der Liste dargestellt werden.

WinLstFromLast
Ab dem letzten Datensatz
Wert 2 /
0x00000002

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Liest ab dem letzten Datensatz der Datei die Datensätze der RecList neu ein. Es werden nur soviele Sätze gelesen, wie in der Liste dargestellt werden.

_WinLstFromSelected
Ausgehend vom selektierten Datensatz
Wert 32 /
0x00000020

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Der Datensatz, der den Fokus hat, bestimmt die Datensätze, die angezeigt werden. Über die Option _WinLstPos... wird die Position angegeben, die der Datensatz in der Liste einnimmt.

_WinLstFromTop
Ab dem ersten sichtbaren Datensatz
Wert 64 /
0x00000040

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Der erste angezeigte Datensatz in der RecList bestimmt die Datensätze, die angezeigt werden. Über die Option _WinLstPos... wird die Position angegeben, die der Datensatz einnimmt.

WinLstIgnoreHidden
Update, auch wenn die Liste nicht sichtbar ist
Wert 1.073.741.824
/ 0x40000000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Das Update wird auch durchgeführt, wenn die Liste oder eines der Elternobjekte nicht sichtbar (Visible = false) ist. In der Regel muss diese Option nicht angegeben werden.

WinLstPosBottom
Letzter sichtbarer Satz in der RecList
Wert 16.384 /
0x00004000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Der in den Feldpuffern befindliche Datensatz erscheint als letzter in der RecList.

_WinLstPosMiddle
Mittelposition in der RecList
Wert 8.192 /
0x00002000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Der in den Feldpuffern befindliche Datensatz erscheint in der Mitte der RecList.

_WinLstPosSelected
Datensatz behält die Position innerhalb der Liste
Wert 32.768 /
0x00008000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Die Option bewirkt einen Neuaufbau der Liste, wobei die Position des selektierten Satzes innerhalb der Liste erhalten bleibt.

_WinLstPosSelected ist in Kombination mit _WinLstFromSelected und _WinLstRecDoSelect zu verwenden.

Beispiel:

```
$RecList->WinUpdate(_WinUpdOn, _WinLstFromSelected | _WinLstPosSelected | _WinLstRecDoSelect);
```

WinLstPosTop
Erster sichtbarer Satz in der RecList
Wert 4.096 /
0x00001000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Der in den Feldpuffern befindliche Datensatz erscheint als erster in der RecList.

_WinLstRecDoSelect
Datensatz selektieren

Wert 256 /
0x00000100

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). In der RecList bekommt der Datensatz den Fokus, der in den Feldpuffern eingetragen ist. Die Position wird über die Option _WinLstPos... bestimmt.

WinLstRecEvtSkip

Unterbinden des Ereignisses EvtLstSelect

Wert 131.072 /
0x00020000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Mit dieser Option wird das Auslösen des Ereignisses EvtLstSelect bei einem Update auf die RecList unterbunden.

_WinLstRecFromBuffer

Ausgehend von den Werten des Feldpuffers

Wert 16 /
0x00000010

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Der Inhalt der Feldpuffer bestimmt die Datensätze, die angezeigt werden. Über die Option _WinLstPos... wird die Position angegeben, die der Datensatz einnimmt.

Existiert der Datensatz der in dem Feldpuffer steht nicht mehr, oder kann er durch einen gesetzten Filter nicht mehr angezeigt werden, wird der nächste darstellbare Datensatz an der entsprechenden Position angezeigt.

_WinLstRecFromRecId
Ausgehend von der Datensatz-ID
Wert 128 /
0x00000080

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Die Datensatz-ID bestimmt, von welchem Satz aus die Liste aufgebaut wird. Über die Option _WinLstPos... wird die Position angegeben, die der Datensatz einnimmt.

_WinLstReorderColumns
Spalten neu nummerieren

Wert 268.435.456 /
0x10000000

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Alle sichtbaren Spalten einer Liste werden neu nummeriert. Anschließend haben alle sichtbaren Spalten eine fortlaufende ClmOrder beginnend mit 1.

_WinMsgAll
Alle Windows Nachrichten verarbeiten
Wert 0

WinSleep(),
Siehe _WinMsgNoKeyboardInput
_WinMsgNoMouseInput
_WinMsgNoInput

Wird diese Konstante bei WinSleep() oder keine der anderen _WinMsg...-Konstanten angegeben, werden alle Windows-Nachrichten aus der Nachrichtenschleife verarbeitet. Führt die Verarbeitung einer Nachricht zur Ausführung eines Ereignisses, welches bereits ausgeführt wird, wird die Nachricht verworfen.

_WinMsgNoInput

Keine Tastatur- oder Mausnachrichten verarbeiten

Wert 3

WinSleep(), _WinMsgAll

Siehe _WinMsgNoKeyboardInput

_WinMsgNoMouseInput

Wird diese Konstante bei WinSleep() angegeben, werden keine Tastatur- oder Mausnachrichten aus der Nachrichtenschleife verarbeitet. Sie verbleiben in der Nachrichtenschleife und werden nach dem WinSleep() ausgeführt.

_WinMsgNoKeyboardInput
Keine Tastaturnachrichten verarbeiten
Wert 1

WinSleep(), _WinMsgAll
Siehe _WinMsgNoMouseInput
_WinMsgNoInput

Wird diese Konstante bei WinSleep() angegeben, werden keine Tastaturnachrichten aus der Nachrichtenschleife verarbeitet. Sie verbleiben in der Nachrichtenschleife und werden nach dem WinSleep() ausgeführt.

_WinMsgNoMouseInput
Keine Mausnachrichten verarbeiten
Wert 2

WinSleep(), _WinMsgAll
Siehe _WinMsgNoKeyboardInput
_WinMsgNoInput

Wird diese Konstante bei WinSleep() angegeben, werden keine Mausnachrichten aus der Nachrichtenschleife verarbeitet. Sie verbleiben in der Nachrichtenschleife und werden nach dem WinSleep() ausgeführt.

_WinPicSignSaveAsExt

Speicherformat aus Dateiendung ermitteln

Wert 1 / 0x0001

Siehe WinPicSaveImage()

Option bei WinPicSaveImage(), durch die das Dateiformat beim Speichern aus der Dateiendung ermittelt wird.

_WinPicSignSaveAsJpg

Zeichnung im JPEG-Format speichern

Wert 2 / 0x0002

Siehe WinPicSaveImage()

Option bei WinPicSaveImage(), durch die die Zeichnung im JPEG-Format gespeichert wird.

_WinPicSignSaveAsPng

Zeichnung im PNG-Format speichern

Wert 4 / 0x0004

Siehe WinPicSaveImage()

Option bei WinPicSaveImage(), durch die die Zeichnung im PNG-Format gespeichert wird.

_WinPicSignSaveAsTif

Zeichnung im TIFF-Format speichern

Wert 8 / 0x0008

Siehe WinPicSaveImage()

Option bei WinPicSaveImage(), durch die die Zeichnung im TIFF-Format gespeichert wird.

_WinPicSignSavePic

Inhalt des Picture-Objektes speichern

Wert 256 / 0x0100

Siehe WinPicSaveImage()

Option bei WinPicSaveImage(), durch die der Inhalt des Picture-Objektes gespeichert wird.

_WinPicSignReset

Zeichnung zurücksetzen

Wert 1.024 / 0x0400

Siehe WinPicSaveImage()

Option bei WinPicSaveImage(), durch die die Zeichnung zurückgesetzt und nur das originale Bild im Picture-Objekt angezeigt wird.

_WinPicSignSaveSign

Zeichnung speichern

Wert 512 / 0x0200

Siehe WinPicSaveImage()

Option bei WinPicSaveImage(), durch die die Zeichnung gespeichert wird.

_WinRvwEditAbortEditing
aktiven Editiervorgang abbrechen
Wert 1 / 0x01

Siehe WinRvwEdit()

Bei dem Befehl WinRvwEdit() kann diese Konstante im Parameter (int4) angegeben werden, um einen aktiven Editiervorgang abubrechen, bevor das ausgewählte Item oder SubItem bearbeitet wird.

_WinRvwUpdateDoKeepSelect
Selektierten Datensatz weiterhin selektieren
Wert 512 / 0x00000200

Verwandte

Siehe Befehle,

WinRvwUpdate()

Option bei WinRvwUpdate(). In dem RecView behält der aktuell selektierte Datensatz den Fokus.



Pro RecView-Objekt gibt es nur einen selektierten Datensatz.

WinRvwUpdateDoSelect

Aktuellen Datensatz aus den Feldpuffern selektieren

Wert 256 /
0x00000100

Verwandte

Siehe Befehle,

WinRvwUpdate(),

EvtLstViewInit

Option bei WinRvwUpdate(). In dem RecView bekommt der Datensatz den Fokus, der in den Feldpuffern eingetragen ist.



Pro RecView-Objekt gibt es nur einen selektierten Datensatz.

_WinRvwUpdateFromFirst
Ab dem ersten Datensatz
Wert 1 / 0x00000001

Verwandte

Siehe Befehle,
WinRvwUpdate(),
EvtLstViewInit

Option bei WinRvwUpdate(). Liest ab dem ersten Datensatz der Datei die Datensätze des RecView neu ein. Es werden nur soviele Sätze gelesen, wie in dem View dargestellt werden.

_WinRvwUpdateFromLast
Ab dem letzten Datensatz
Wert 2 / 0x00000002

Verwandte

Siehe Befehle,
WinRvwUpdate(),
EvtLstViewInit

Option bei WinRvwUpdate(). Liest ab dem letzten Datensatz der Datei die Datensätze des RecView neu ein. Es werden nur soviele Sätze gelesen, wie in dem View dargestellt werden.

_WinRvwUpdateFromRecBuf
Ausgehend von den Werten des Feldpuffers.
Wert 5 / 0x00000005

Verwandte

Siehe Befehle,
WinRvwUpdate(),
EvtLstViewInit

Option bei WinRvwUpdate(). Der Inhalt der Feldpuffer bestimmt die Datensätze, die angezeigt werden.

Existiert der Datensatz der in dem Feldpuffer steht nicht mehr, oder kann er durch einen gesetzten Filter nicht mehr angezeigt werden, wird der nächste darstellbare Datensatz angezeigt.

_WinRvwUpdateFromSelected

Ab dem ersten sichtbaren Datensatz

Wert 4 / 0x00000004

Verwandte

Siehe Befehle,

WinRvwUpdate()

Option bei WinRvwUpdate(). Der erste angezeigte Datensatz in dem RecView ist die aktuell selektierte Gruppe.

_WinRvwUpdateFromTop
Ab dem ersten sichtbaren Datensatz
Wert 3 / 0x00000003

Verwandte

Siehe Befehle,
WinRvwUpdate(),
EvtLstViewInit

Option bei WinRvwUpdate(). Der erste angezeigte Datensatz in dem RecView bestimmt die Datensätze, die angezeigt werden.

_WinRvwUpdateFromTopLocation

Angezeigte Datensätze aktualisieren und Position der ersten angezeigten Gruppe beibehalten

Wert 6 / 0x00000006

Verwandte

Befehle,

Siehe WinRvwUpdate(),

EvtLstViewInit

Option bei WinRvwUpdate(). Der Befehl mit dieser Option führt einen Neuaufbau der angezeigten Datensätze des RecView-Objektes durch, ohne die Position der ersten angezeigten Gruppe zu ändern.

_WinRvwUpdateOptClearCache
Cache des RecView-Objektes leeren
Wert 65.536 / 0x00010000

Verwandte Befehle,

Siehe WinRvwUpdate(),

_WinRvwUpdateOptClearSelected

Option bei WinRvwUpdate(). Alle Gruppen werden aus dem Cache des RecView-Objektes entfernt. Dies hat zur Folge, dass das Ereignis EvtLstGroupInit für alle anzuzeigenden Gruppen durchgeführt wird.

WinRvwUpdateOptClearSelected

Wert 524.288 / 0x00080000

Verwandte Befehle,

Siehe WinRvwUpdate(),

WinRvwUpdateOptClearCache

Option bei WinRvwUpdate(). Die aktuell selektierte Gruppe wird aus dem Cache des RecView-Objektes entfernt. Dies hat zur Folge, dass das Ereignis EvtLstGroupInit für diese Gruppe durchgeführt wird.

_WinUpdActivate
Frame-Objekt in den Vordergrund holen
Wert 30

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Holt das Fenster-Objekt in den Vordergrund.



Die Aktivierung lässt sich nicht erzwingen. Liegt eine andere Anwendung über dem Fenster der CONZEPT 16-Applikation, verhindert Windows, dass diese Anwendung den Fokus verliert. Dies führt zum Blinken des Fensters, welches aktiviert werden soll.

WinUpdBuf2Obj
Übertragen des Textpuffers
Wert 13

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Überträgt den Inhalt des Textpuffers in das Objekt TextEdit.

WinUpdFld2Buf

Übertragen der Feldpuffer in den Datensatzpuffer

Wert 11

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate() - Überträgt den Feldpuffer in den Datensatzpuffer der Objekte RecList und MdiFrame.

- **RecList**

Ist die Eigenschaft DbRecBuf einer RecList gesetzt, werden Änderungen der Feldpuffer die durch die RecList verursacht werden, auf den ursprünglichen Feldpufferinhalt zurückgesetzt. Dies ist auch der Fall, wenn in dem Ereignis EvtLstSelect das Lesen eines Datensatzes stattfindet. Soll der Feldpuffer des in diesem Ereignis gelesenen Datensatzes erhalten bleiben, muss der Feldpuffer in den Puffer der RecList übertragen werden.

Beispiel:

```
sub EvtLstSelect( aEvt : event; // Ereignis aID : bigint; // Datensatz-ID) : l
```

- **MdiFrame**

Bei einem MdiFrame überträgt der Befehl die Inhalte der Feldpuffer in den Datensatzpuffer (DbRecBuf) des Objektes.

Beispiel:

```
$MdiFrame->WinUpdate(_WinUpdFld2Buf);
```

Wird der Befehl mit einem Objekt benutzt, welches kein Fenster-Objekt und kein RecList-Objekt ist, wird das Fenster-Objekt selbstständig ermittelt.

Beispiel:

```
$button->WinUpdate(_WinUpdFld2Buf);
```

Es wird zunächst das übergeordnete Fenster-Objekt des Buttons ermittelt. Dann werden die Feldpuffer in die Datensatzpuffer des Fenster-Objektes übertragen.

WinUpdFld2Obj
Übertragen der Feldpuffer
Wert 10

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Überträgt den Inhalt der Feldpuffer in das Objekt, sofern in der Eigenschaft DbFieldName des Objektes ein Datenbankfeld angegeben ist.

WinUpdObj2Buf

Übertragen des Inhaltes des Objektes in den Textpuffer

Wert 12

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Überträgt den Inhalt des Objektes TextEdit in den Textpuffer.

WinUpdObj2Fld
Übertragen der Objektinhalte
Wert 14

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Überträgt den Inhalt des Eingabeobjektes in den Feldpuffer, sofern in der Eigenschaft DbFieldName des Objektes ein Datenbankfeld angegeben ist.

_WinUpdOff

Eigenschaft AutoUpdate auf false setzen

Wert 21

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Setzt die Eigenschaft AutoUpdate eines Objektes auf false. Der Parameter kann mit der Option _WinUpdScrollPos kombiniert werden, um bei Objekten mit einem Scrollbar die Scrollposition zu speichern. Die Scrollposition muss kann später mit der Kombination _WinUpdOn | _WinUpdScrollPos wieder hergestellt werden.

_WinUpdOn

Eigenschaft AutoUpdate auf true setzen

Wert 20

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Setzt die Eigenschaft AutoUpdate eines Objektes auf true. Der Parameter kann mit der Option _WinUpdScrollPos kombiniert werden, um bei Objekten mit einem Scrollbar die Scrollposition wieder herzustellen. Die Scrollposition muss zuvor mit der Kombination _WinUpdOff | _WinUpdScrollPos gesichert worden sein.

_WinUpdScrollPos

Speichern und Wiederherstellen der Scrollposition

Wert -2.147.483.648

/ 0x80000000

Verwandte

Siehe Befehle,

WinUpdate()

Bei der Anweisung WinUpdate() können die Konstanten _WinUpdOff und _WinUpdOn mit dieser Konstante kombiniert werden.

Die Anweisung tObj->WinUpdate(_WinUpdOff | _WinUpdScrollPos) speichert die derzeitige Scrollposition und setzt die Eigenschaft AutoUpdate auf false. Zu einem späteren Zeitpunkt kann mit der Anweisung tObj->WinUpdate(_WinUpdOn | _WinUpdScrollPos) die ursprüngliche Scrollposition wieder hergestellt werden.

Die Kombination kann bei allen Objekten verwendet werden, die eine normale Scrollbar (Listen-Objekte, TextEdit usw.) besitzen. Besitzt das Objekt keine normale Scrollbar, zum Beispiel bei einem RecList-Objekt mit dem Ereignis EvtLstRecControl, kann die Scrollposition nicht gesichert werden.

Wird die Scrollposition wieder hergestellt, ohne die Position zuvor zu sichern, erfolgt der Laufzeitfehler _ErrValueInvalid.

Beispiele:

```
aEvt:Obj->WinUpdate(_WinUpdOff | _WinUpdScrollPos);...aEvt:Obj->WinUpdate(_WinUpdOn | _WinUpdScro
```


WinUpdSort
Sortierung aktualisieren
Wert 32

Verwandte

Siehe Befehle,

WinUpdate()

Option bei WinUpdate(). Der Inhalt eines DataList-Objekts wird neu sortiert. Die Anweisung muss aufgerufen werden, wenn Änderungen am Inhalt von einer oder mehreren Zeilen vorgenommen wurden.

_WinUpdState
Status setzen
Wert 31

Verwandte
Siehe Befehle,
WinUpdate()
Option bei WinUpdate().

Bei Angabe dieser Option muss in (int2) eine der folgenden Konstanten verwendet werden:

<u>_WinDialogMaximized</u>	Dialog maximieren
<u>_WinDialogMinimized</u>	Dialog minimieren
<u>_WinDialogNormal</u>	Dialog weder maximieren noch minimieren
<u>_WinGntRefresh</u>	<u>GanttGraph</u> -Objekt neu zeichnen
<u>_WinGntRecalc</u>	<u>GanttGraph</u> -Objekt neu berechnen
<u>_WinStateAttachClosed</u>	<u>GroupTile</u> -Objekt unsichtbar setzen
<u>_WinStateAttachMaximized</u>	<u>GroupTile</u> -Objekt maximieren. Ist das <u>GroupTile</u> -Objekt vor der Ausführung unsichtbar, wird es automatisch auf sichtbar gesetzt.
<u>_WinStateAttachNormal</u>	<u>GroupTile</u> -Objekt normal darstellen. Ist das <u>GroupTile</u> vor der Ausführung unsichtbar, wird es automatisch auf sichtbar gesetzt.

Beispiele:

```
// Dialog maximieren$Frame->WinUpdate(_WinUpdState, _WinDialogMaximized);// Dialog minimieren$Fra
```

Befehle der Mehrfachselektion

Befehle zur Programmierung einer Mehrfachselektion


Siehe Objekte der Mehrfachselektion,
Mehrfachselektions-Ereignisse

Befehle:

- WinMsdDelete
- WinMsdDeleteName
- WinMsdInsert
- WinMsdInsertName
- WinMsdRead
- WinMsdReadName
- WinMsdUpdate

Konstanten:

- WinMsdNoTreeSync
- WinMsdRecId

obj -> WinMsdDelete(bigint1[, int2]) 

: int

Element aus der Selektion entfernen

obj Deskriptor des Objekts, in dem selektiert wird

bigint1 Element, das aus der Selektion entfernt werden soll

Optionen (optional)

int2 _WinMsdNoTreeSync TreeView-Objekt nicht neu aufbauen

_WinMsdRecId Datensatz-ID entfernen

Fehlerwert

Resultat int _ErrOk Element entfernt.

_ErrMsdNotFound Das Element wurde nicht gefunden.

Siehe Verwandte Befehle, WinMsdInsert(), WinMsdRead(), WinMsdDeleteName()

Mit diesem Befehl wird das Element (bigint1) aus der Menge der selektierten Elemente entfernt.

Das Objekt, in dem die Selektion stattfindet wird als (obj) der Anweisung übergeben. In (bigint1) wird das Element übergeben, dass aus der Selektion entfernt werden soll. In Abhängigkeit vom Objekt muss in (bigint1) eine Datensatz-ID (RecList), eine Zeilennummer (DataList), der Deskriptor eines TreeNode-Objekts (TreeView), der Deskriptor eines CanvasGraphic-Objekts (Canvas) oder der Deskriptor eines CteItems (SelectionData) übergeben werden.

Beim RecList-Objekt kann auch die Option _WinMsdRecId verwendet werden. In (bigint1) muss dann die Dateinummer angegeben werden. Dadurch wird automatisch die Datensatz-ID des aktuell geladenen Datensatzes der angegebenen Datei ermittelt.

Bei den Objekten RecList und DataList wird das neu hinzugefügte Selektions-Element sichtbar, nachdem das Objekt durch den Befehl WinUpdate() neu gezeichnet wurde. Beim Canvas-Objekt muss zusätzlich die Option _WinUpdState angegeben werden. Beim TreeView-Objekt müssen die Selektions-Elemente explizit übertragen werden, damit die Selektion im TreeView-Objekt sichtbar wird. Bei Verwendung von WinMsdDelete() ist dies standardmäßig der Fall. Durch _WinMsdNoTreeSync kann die Übertragung des neuen Elementes in den TreeView verhindert werden. Um alle Elemente einer Mehrfachselektion in das TreeView-Objekt zu übertragen kann der Befehl WinMsdUpdate() benutzt werden.

Natürlich kann zum Entfernen eines Selektions-Elementes auch der Befehl CteDelete() verwendet werden. Beim TreeView-Objekt wird anschließend eine Übertragung der Elemente in der Selektionsmenge in das TreeView-Objekt notwendig. Hierzu kann der Befehl WinMsdUpdate() verwendet werden.

Beispiele:

```
// Entfernen eines Datensatzes$RecList->WinMsdDelete(tblCstCustomer, _WinMsdRecId);// Identisch m
```

Mögliche Laufzeitfehler:

- __ErrHdlInvalid Der als (obj) übergebene Deskriptor ist ungültig.
- __ErrValueInvalid Die in (int2) übergebenen Optionen sind nicht zulässig.
- __ErrFileInvalid Die angegebene Dateinummer in (int1) existiert nicht.

obj -> WinMsdDeleteName(alpha1) : handle



Element aus der Selektion einer StoList entfernen

obj Deskriptor des Objekts, in dem selektiert wird

alpha1 Element, das aus der Selektion entfernt werden soll

Fehlerwert

Resultat int ErrOk Element entfernt.
ErrMsdNotFound Das Element wurde nicht
gefunden.

Siehe Verwandte Befehle, WinMsdInsertName(),
WinMsdReadName(), WinMsdDelete()

Mit diesem Befehl wird das Element (alpha1) aus der Menge der selektierten
Elemente entfernt.

Das Objekt, in dem die Selektion stattfindet wird als (obj) der Anweisung übergeben.
In (alpha1) wird das Element übergeben, dass aus der Selektion entfernt werden soll.

Natürlich kann zum Entfernen eines Selektions-Elementes auch der Befehl CteDelete()
verwendet werden.



Dieser Befehl kann nur mit StoList-Objekten verwendet werden.

Beispiele:


```
$StoList->WinMsdDeleteName('StoDir');
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der als (obj) übergebene Deskriptor ist ungültig.

ErrValueInvalid Die in (alpha2) übergebene Zeichenkette ist leer.

obj ->

WinMsdInsert(bigint1 ,
int2[, alpha3]) : int

Selektion erweitern

obj Deskriptor des Objekts, in
dem selektiert wird

bigint1 Element, das in die Selektion
aufgenommen werden soll

Optionen (optional)

_WinMsdNoTreeSync TreeView-Objekt

int2 nicht neu
aufbauen

_WinMsdRecId Datensatz-ID
einfügen

alpha3 Custom-Eigenschaft

Fehlerwert

_ErrOk Element
aufgenommen.

Resultat int _ErrMsdExists Das Element
ist bereits
selektiert.

Verwandte Befehle,

Siehe WinMsdDelete(),
WinMsdRead(),
WinMsdInsertName()

Mit diesem Befehl wird das Element (bigint1) in die Menge der selektierten Elemente hinzugefügt.

Das Objekt, in dem die Selektion stattfindet wird als (obj) der Anweisung übergeben. In (bigint1) wird das Element übergeben, das in die Selektion aufgenommen werden soll. In Abhängigkeit vom Objekt muss in (bigint1) eine Datensatz-ID (RecList), eine Zeilennummer (DataList), der Deskriptor eines TreeNode-Objekts (TreeView), der Deskriptor eines CanvasGraphic-Objektes (Canvas) oder der Deskriptor eines Cte-Items (SelectionData) übergeben werden.

Beim RecList-Objekt kann auch die Option _WinMsdRecId verwendet werden. In (bigint1) muss dann die Dateinummer angegeben werden. Dadurch wird automatisch die Datensatz-ID des aktuell geladenen Datensatzes der angegebenen Datei ermittelt.

In (alpha3) kann eine Zeichenkette angegeben werden, die in der Custom-Eigenschaft des Cte-Items gespeichert wird.

Bei den Objekten RecList und DataList wird das neu hinzugefügte Selektions-Element sichtbar, nachdem das Objekt durch den Befehl WinUpdate() neu gezeichnet wurde. Beim Canvas-Objekt muss zusätzlich die Option _WinUpdState angegeben werden. Beim TreeView-Objekt muss das neu hinzugefügte Selektions-Element explizit übertragen werden, damit es im TreeView-Objekt sichtbar wird. Bei Verwendung von WinMsdInsert ist dies standardmäßig der Fall. Durch _WinMsdNoTreeSync kann die Übertragung des neuen Elementes in den TreeView verhindert werden. Um alle

Elemente einer Mehrfachselektion in das TreeView-Objekt zu übertragen kann der Befehl WinMsdUpdate() benutzt werden.

Natürlich kann für die Erzeugung eines neuen Selektions-Elements auch der Befehl CteInsert() verwendet werden, jedoch müssen dann die Eigenschaften Name und ID konform mit den Bedingungen der Mehrfachselektion gesetzt werden. Beim TreeView-Objekt wird zudem eine Übertragung der Elemente in der Selektionsmenge in das TreeView-Objekt notwendig. Hierzu kann der Befehl WinMsdUpdate() verwendet werden.

Beispiele:

```
// Einfügen eines Datensatzes$RecList->WinMsdInsert(tblCstCustomer, _WinMsdRecId);// Identisch mit
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der als (obj) übergebene Deskriptor ist ungültig.

_ErrValueInvalid Die in (int2) übergebenen Optionen sind nicht zulässig.

_ErrFileInvalid Die angegebene Dateinummer in (int1) existiert nicht.

obj ->

WinMsdInsertName(alpha1[,
alpha2]) : int



Selektion einer StoList erweitern

obj Deskriptor des Objekts, in dem
 selektiert wird

alpha1 Zeichenkette, um die die Selektion
 erweitert werden soll

alpha2 Custom-Eigenschaft (optional)

 Fehlerwert

Resultat int ErrOk Element
 aufgenommen.
 ErrMsdExists Das Element ist
 bereits selektiert.

Verwandte Befehle,

Siehe WinMsdDeleteName(),
 WinMsdReadName(), WinMsdInsert()

Mit diesem Befehl wird die Zeichenkette (alpha1) in die Menge der selektierten Elemente hinzugefügt.

Das Objekt, in dem die Selektion stattfindet wird als (obj) der Anweisung übergeben. In (alpha1) wird das Element übergeben, dass in die Selektion aufgenommen werden soll.

In (alpha2) kann eine Zeichenkette angegeben werden, die in der Custom-Eigenschaft des Cte-Items gespeichert wird.

Natürlich kann für die Erzeugung eines neuen Selektions-Elements auch der Befehl CteInsert() verwendet werden, jedoch müssen dann die Eigenschaften Name und ID konform mit den Bedingungen der Mehrfachselektion gesetzt werden.



Dieser Befehl kann nur mit StoList-Objekten verwendet werden.

Beispiele:

```
$StoList->WinMsdInsertName('StoDir');
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der als (obj) übergebene Deskriptor ist ungültig.

_ErrValueInvalid Die in (alpha2) übergebene Zeichenkette ist leer.

obj -> WinMsdRead(bigint1[, ,
int2]) : handle

Element einer Selektion lesen

obj Deskriptor des Objekts, in dem
 selektiert wird

bigint1 Element, das gelesen werden soll
 oder Dateinummer
 Optionen (optional)

int2 _WinMsdRecId In (bigint1) wurde
 eine Dateinummer
 übergeben

Resultat handle Deskriptor des
 CteItem-Objektes

Verwandte Befehle,

Siehe WinMsdDelete(), WinMsdInsert(),
 WinMsdReadName()

Mit diesem Befehl wird das Element (bigint1) in der Menge der selektierten Elemente gelesen. Mit dem Befehl kann geprüft werden, ob ein bestimmtes Element in der Selektion enthalten ist. Zum Lesen aller selektierten Elemente sollte der Cte-Tree oder die Cte-Liste verwendet werden (siehe SelectionData).

Das Objekt, bei dem die Selektion gelesen werden soll, wird als (obj) der Anweisung übergeben. In (bigint1) wird das Element übergeben, dass gelesen werden soll. In Abhängigkeit vom Objekt muss in (bigint1) eine Datensatz-ID (RecList), eine Zeilennummer (DataList), der Deskriptor eines TreeNode-Objekts (TreeView) oder der Deskriptor eines CanvasGraphic-Objekts (Canvas) übergeben werden.

Die Übergabe einer Datensatz-ID kann auch durch die Angabe der Dateinummer in (bigint1) erfolgen, wenn in (int2) die Konstante _WinMsdRecId übergeben wird.

Zurückgegeben wird der Deskriptor auf das CteItem-Objekt, das das selektierte Element enthält (siehe auch SelectionData).

Beispiele:

```
// Überprüfen eines Datensatzes if ($RecList->WinMsdRead(tblCstCustomer, _WinMsdRecId) > 0){ // D
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der als (obj) übergebene Deskriptor ist ungültig.

_ErrValueInvalid Die in (int2) übergebenen Optionen sind nicht zulässig.

_ErrFileInvalid Die angegebene Dateinummer in (int1) existiert nicht.

obj -> WinMsdReadName(alpha1) : handle 

Element einer Selektion einer StoList lesen

obj Deskriptor des Objekts, in dem
 selektiert wird

alpha1 Element, das gelesen werden
 soll

Resultat handle Deskriptor des Cte-Items

Siehe Verwandte Befehle,
 WinMsdDeleteName(),
 WinMsdInsertName(),
 WinMsdRead()

Mit diesem Befehl wird das Element (alpha1) in der Menge der selektierten Elemente gelesen. Mit dem Befehl kann geprüft werden, ob ein bestimmtes Element in der Selektion enthalten ist. Zum Lesen aller selektierten Elemente sollte der Cte-Tree oder die Cte-Liste verwendet werden (siehe SelectionData).

Das Objekt, bei dem die Selektion gelesen werden soll, wird als (obj) der Anweisung übergeben. In (alpha1) wird das Element übergeben, dass gelesen werden soll, angegeben.

Zurückgegeben wird der Deskriptor auf das Cte-Item, das das selektierte Element enthält (siehe auch SelectionData).




Dieser Befehl kann nur mit StoList-Objekten verwendet werden.

Beispiele:

```
// Überprüfen, ob das Storage-Objekt StoDir selektiert ist if ($StoList->WinMsdReadName('StoDir'))
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der als (obj) übergebene Deskriptor ist ungültig.

obj ->
 WinMsdUpdate([int1]) 
 : int
 Selektion aktualisieren
 obj Deskriptor des Objekts, in
 dem selektiert wird
 int1 Optionen (reserviert /
 optional)
 Fehlerwert
 Resultat int _ErrOk Element
 aufgenommen.

Siehe Verwandte Befehle

Der Befehl überträgt alle Selektions-Elemente einer Mehrfachselektion in die interne Selektionsliste des TreeView-Objekts. Nach erfolgreicher Durchführung des Befehls sind alle TreeNode-Objekte, die in der Mehrfachselektion vorhanden sind auch im TreeView selektiert. Ein Neuzeichnen des TreeView-Objektes findet nur statt, wenn die Eigenschaft AutoUpdate des TreeView-Objekts gesetzt (true) ist.

Als Objekt muss ein TreeView-Objekt oder SelectionData-Objekt, das zu einem TreeView-Objekt gehört, übergeben werden.

Zur Zeit können keine Optionen angegeben werden.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der als (obj) übergebene Deskriptor ist kein TreeView-Objekt oder kein SelectionData-Objekt, das zu einem TreeView-Objekt gehört.
_ErrValueInvalid Die in (int1) übergebenen Optionen sind nicht zulässig.

Befehle für dynamische Strukturen

Liste der Befehle und Konstanten zur Bearbeitung von dynamischen Strukturen

Befehlsgruppen,

Befehlsliste,

Siehe Dynamische

Strukturen,


Beispiel

Befehle

- CteClear
- CteClose
- CteDelete
- CteInfo
- CteInsert
- CteInsertItem
- CteInsertNode
- CteNodeValueAlpha
- CteOpen
- CteRead

Konstanten

- _CteAfter
- _CteAttribList
- _CteAttribTree
- _CteBefore
- _CteChildList
- _CteChildTree
- _CteCmpE
- _CteCmpG
- _CteCmpGE
- _CteCmpL
- _CteCmpLE
- _CteCount
- _CteCustom
- _CteFirst
- _CteItem
- _CteLast
- _CteList
- _CteNext
- _CteNode
- _CteNodePath
- _CteNodePathCI
- _CteNodeValueRead
- _CteNodeValueWrite
- _CtePrev
- _CteSearch
- _CteSearchCI
- _CteTree
- _CteTreeCI

obj ->
CteClear(logic1[, 
int2])

Liste/Knoten leeren

obj Liste/Knoten

logic1 Enthaltene Elemente
löschen

Knotenart bei Knoten
(optional)

int2 CteChild Untergeordnete
Knoten löschen

CteAttrib Attributknoten
löschen

Siehe Verwandte Befehle,
CteDelete(), Beispiel

Mit dieser Funktion können alle Elemente aus einer Liste oder einem Knoten entfernt werden. Wird in (logic1) true angegeben, werden die Elemente gelöscht, bei false bleiben sie nach dem Entfernen erhalten.



Bei einem CteNode-Objekt werden die untergeordneten Knoten bzw. die Attributknoten unabhängig von (logic1) immer gelöscht.

Beispiele:

```
// Liste leeren und Elemente löschtList->CteClear(true); // Liste leeren und Elemente beibehalten
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Liste/Baum (obj) ungültig

ErrValueInvalid Knotenart (int2) unbekannt



obj -> CteClose()

Element, Liste oder Knoten löschen

Element,

obj Liste oder

Knoten

Verwandte

Befehle,

Siehe CteOpen(),

CteClear(),

Beispiel

Mit dieser Funktion wird ein mit CteOpen() erzeugtes Objekt gelöscht. Beim Löschen einer Liste oder eines Baums bleiben alle darin enthaltenen Elemente erhalten und können wieder in eine andere dynamische Struktur eingefügt werden. Um alle enthaltenen Elemente auf einmal zu löschen, kann CteClear() benutzt werden.

Beim Löschen eines Elementes wird dieses automatisch aus allen Listen entfernt.

Beim Löschen eines Knotens wird dieser automatisch aus dem übergeordneten Knoten entfernt.

Es ist darauf zu achten, dass jedes nicht mehr verwendete Objekt auch gelöscht wird, da sonst der verfügbare Arbeitsspeicher mit der Zeit immer geringer wird.




Bei einem CteNode-Objekte werden die untergeordneten Knoten bzw. die Attributknoten ebenfalls gelöscht.

Beispiel:

```
// Objekt löschentItem->CteClose();
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Element, Liste oder Knoten (obj) ungültig

obj -> CteDelete(handle1[, int2]) : 

logic

Element aus Liste/Knoten entfernen

obj Liste/Knoten

handle1 Element

Knotenart bei Knoten

(optional)

_CteChild Untergeordneten
int2 Knoten
entfernen

_CteAttrib Attributknoten
entfernen

Resultat logic Entfernungserfolg

Siehe Verwandte Befehle,
CteInsert()

Mit dieser Funktion wird das Element (handle1) aus der Liste oder dem Knoten (obj) entfernt. Das Resultat ist false, wenn sich das Element nicht in der Liste/dem Baum befindet. Das Element selbst bleibt dabei erhalten.

Beispiele:

```
// Element aus Liste entfernenList->CteDelete(tItem);// Untergeordneten Knoten aus Knoten entfer
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Struktur (obj) oder Element (handle1) ungültig



obj -> CteInfo(int1) : int

Listen- / Element- / Knoteninformation ermitteln

obj Liste / Element / Knoten

Informationstyp

CteCount Elementanzahl von
Liste ermitteln

int1 CteList Verkettete Liste von
Element ermitteln

CteTree Sortierte Liste von
Element ermitteln

Resultat int Struktur-/Elementinformation

Siehe Verwandte Befehle

Mit dieser Funktion können verschiedene Informationen von einem Knoten, einer verketteten Liste, einer sortierten Liste oder einem Element ermittelt werden.

Bei einer Liste wird bei CteCount die Anzahl der enthaltenen Elemente zurückgeliefert.



Bei CteNode-Objekten wird die Anzahl aller Elemente in der Struktur gezählt. Sollen nur die direkten Kinder / Attribute gezählt werden, können die Eigenschaften ChildCount und AttribCount verwendet werden.

Für ein Element kann damit ermittelt werden, in welchen Listen es enthalten ist. Das Resultat bei CteList bzw. CteTree ist 0, wenn das Element in keiner Liste enthalten ist.

Beispiel:

```
// Elementanzahl von Liste ermitteltCount # tList->CteInfo(_CteCount);// Verkettete Liste von E
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Liste/Element (obj) ungültig

ErrValueInvalid Informationstyp (int1) unbekannt oder (obj) vom falschen Typ

2365

Das Resultat ist false, wenn das Element bereits in einer Liste enthalten ist oder sich das Referenzelement nicht in der Liste (obj) befindet (bei Verwendung von _CteBefore oder _CteAfter).

Beispiele:

```
// Element an Listenanfang einfügentList->CteInsert(tItem, _CteFirst); // Element an Listenende einfügen
```

Einfügen in eine sortierte Liste

Das Element wird anhand seiner Eigenschaft Name in die Liste einsortiert. Die Eigenschaft darf nicht leer sein.

Das Resultat ist false, wenn das Element bereits in einer sortierten Liste enthalten ist oder sich bereits ein Element mit gleichem Namen in der Liste befindet.

Beispiel:

```
// Element in Liste einsortierentList->CteInsert(tItem);
```

Einfügen in einen Knoten

Der Knoten wird entweder mit _CteChild als untergeordneter Knoten oder mit _CteAttrib als Attributknoten eingefügt. Standardmäßig wird _CteChild verwendet

Verfügt der Knoten über eine verkettete Liste, können auch die Optionen zur Positionierung verwendet werden.

Das Resultat ist false, wenn der Knoten bereits in einem Knoten enthalten ist oder, wenn der Knoten über eine sortierte Liste verfügt, sich bereits ein Knoten mit gleichem Namen in dem Knoten befindet.

Beispiele:

```
// Untergeordneten Knoten in Knoten einfügentNode->CteInsert(tNode, _CteChild); // Untergeordneter Knoten
```

Mögliche Laufzeitfehler:

<u>_ErrHdlInvalid</u>	Liste/Knoten (obj), Element (handle1) oder Referenzelement (handle3) ungültig
<u>_ErrValueInvalid</u>	Einfügeposition (int2) unbekannt

Kontakt

obj -> CteInsertItem(alpha1, bigint2,
alpha3[, int4[, int5]]) : handle



Element erzeugen und in Liste einfügen

obj Liste

alpha1 Name-Eigenschaft

bigint2 ID-Eigenschaft

alpha3 Custom-Eigenschaft

Einfügeposition bei verketteter Liste

(optional)

CteFirst Element am Anfang einfügen

CteLast Element am Ende einfügen

int4 CteBefore Element vor Referenzelement
einfügen

CteAfter Element nach Referenzelement
einfügen

int5 Referenzelement (optional)

ErrExists Element mit dem
Namen ist bereits
vorhanden

Resultat handle ErrNameInvalid Der Name ist leer
> 0 Deskriptor des
Elements

Siehe Verwandte Befehle, CteOpen(), CteInsert(),
CteDelete()

Mit diesem Befehl wird ein Element mit den Eigenschaften Name (alpha1), ID (bigint2) und Custom (alpha3) erzeugt und in die Liste (obj) eingefügt.

Das Resultat ist ErrExists, wenn ein Element mit dem Namen bereits in einer sortierten Liste vorhanden ist. Ist der Name leer, ist das Resultat ErrNameInvalid. Andernfalls wird der Deskriptor des eingefügten Elementes zurückgegeben.

Mögliche Laufzeitfehler:

ErrHdlInvalid Liste/Knoten (obj), oder Referenzelement (int5) ungültig.

ErrValueInvalid Einfügeposition (int4) unbekannt

Kontakt

obj -> CteInsertNode(alpha1, bigint2, var3[, int4[, int5[, alpha6]]) : handle

Knoten erzeugen und in Knoten einfügen

obj Knoten

alpha1 Name-Eigenschaft

bigint2 ID-Eigenschaft

var3 Value-Eigenschaft

Einfügeposition bei verketteter Liste (optional)

CteFirst Element am Anfang einfügen

CteLast Element am Ende einfügen

CteBefore Element vor Referenzelement einfügen

int4 CteAfter Element nach Referenzelement einfügen

Listenart (optional)

CteChild Untergeordneten Knoten einfügen

CteAttrib Attributknoten einfügen

int5 Referenzknoten (optional)

alpha6 Custom-Eigenschaft (optional)

ErrExists Knoten mit dem Namen ist bereits vorhanden

Resultat handle ErrNameInvalid Der Name ist leer

> 0 Deskriptor des Knoten

Siehe Verwandte Befehle, CteOpen(), CteInsert(), CteDelete()

Mit diesem Befehl wird ein Knoten mit den Eigenschaften Name (alpha1), ID (bigint2), Value (var3) und Custom (alpha6) erzeugt und in den Knoten (obj) eingefügt.

Die Eigenschaft Flags des neuen Knotens wird vom Knoten (obj) übernommen.

Der Knoten wird entweder mit CteChild als untergeordneter Knoten oder mit CteAttrib als Attributknoten eingefügt. Standardmäßig wird CteChild verwendet.

Verfügt der Knoten über eine verkettete Liste, können auch die Optionen zur Positionierung verwendet werden.

Das Resultat ist ErrExists, wenn ein Element mit dem Namen bereits in einer sortierten Liste vorhanden ist. Ist der Name leer, ist das Resultat ErrNameInvalid. Andernfalls wird der Deskriptor des eingefügten Knoten zurückgegeben.

Mögliche Laufzeitfehler:

ErrHdlInvalid Knoten (obj), oder Referenzknoten (int5) ungültig

ErrValueInvalid Einfügeposition (int4) unbekannt

obj -> CteNodeValueAlpha(handle1, int2) : int 

Zeichenkette des Knoten lesen oder schreiben


obj CteNode-Objekt

handle1 Memory-Objekt

Durchzuführende Operation:

int2 CteNodeValueWrite Inhalt in Knoten
übertragen

CteNodeValueRead Inhalt aus Knoten lesen

Resultat int ErrOK Befehl erfolgreich durchgeführt 

Siehe Verwandte Befehle, CteOpen(), CteInsert(),
CteDelete()

Die Länge eines CteNode-Objektes mit alphanumerischem Inhalt (Type = TypeAlpha) ist nicht längenlimitiert. Mit der Eigenschaft ValueAlpha lassen sich jedoch maximal 65520 Byte schreiben bzw. lesen. CteNodeValueAlpha() schreibt und liest alphanumerische Inhalte mit mehr als 65520 Byte.

Der Deskriptor des CteNode-Objektes wird im Argument (obj) angegeben. Das Argument (handle1) enthält den Deskriptor des Memory-Objektes. Das Argument (int2) definiert die durchzuführende Operation:

CteNodeValueWrite Diese Option kopiert den Inhalt des Memory-Objektes in das CteNode-Objekt. Nach dem Aufruf kann auch die Eigenschaft ValueAlpha verwendet werden, um den neuen Inhalt auszulesen. Hier können jedoch maximal 65520 Byte verarbeitet werden. Der Datentyp (Eigenschaft Type) des CteNode-Objektes wird auf TypeAlpha modifiziert.

CteNodeValueRead Diese Option kopiert den Inhalt des CteNode-Objektes in das Memory-Objekt. Der Datentyp (Eigenschaft Type) des CteNode-Objektes muss TypeAlpha sein.

 Der Zeichensatz des Memory-Objektes (Eigenschaft Charset) wird ignoriert.

Der Rückgabewert liefert immer ErrOk. Konnte die Verarbeitung nicht durchgeführt werden, wird ein Laufzeitfehler generiert.

Mögliche Laufzeitfehler:

<u>ErrHdlInvalid</u>	<u>CteNode</u> -Objekt (obj) oder <u>Memory</u> -Objekt (handle1) enthalten keine gültigen Deskriptoren.
<u>ErrValueInvalid</u>	Der Modus (int2) enthält einen ungültigen Wert. Der Buffer des <u>Memory</u> -Objektes ist nicht groß genug, um den Inhalt komplett speichern zu können. Bei Verwendung der Option <u>MemAutoSize</u> kann der Fehler auftreten, wenn das <u>Memory</u> -Objekt mehr als 512 MB an Daten benötigt (nur 32-bit-Client). Die Fehler treten beim Lesen des Inhaltes mit <u>CteNodeValueRead</u> auf.
<u>ErrValueRange</u>	Der Datentyp des <u>CteNode</u> -Objektes ist nicht <u>TypeAlpha</u> . Der Fehler tritt beim Lesen des Inhaltes mit <u>CteNodeValueRead</u> auf.
<u>ErrFldType</u>	
<u>ErrMemExhausted</u>	

Kontakt

Nicht genügend freier Arbeitsspeicher zum Durchführen des Befehls.



CteOpen(int1[, int2]) : handle
Element, Liste oder Knoten erzeugen

	Objekttyp	
	<u>CteItem</u>	Element erzeugen
	<u>CteList</u>	Verkettete Liste erzeugen
	<u>CteTree</u>	Sortierte Liste erzeugen
int1	<u>CteTreeCI</u>	Sortierte Liste ohne Unterscheidung von Groß-/Kleinschreibung erzeugen
	<u>CteNode</u>	Knoten erzeugen
	Optionen bei Knoten (optional)	
	<u>CteChildList</u>	Verkettete Liste der untergeordneten Knoten
	<u>CteChildTree</u>	Sortierte Liste der untergeordneten Knoten
	<u>CteChildTreeCI</u>	Sortierte Liste der untergeordneten Knoten (ohne Unterscheidung der Groß-/Kleinschreibung)
int2	<u>CteAttribList</u>	Verkettete Liste der Attributknoten
	<u>CteAttribTree</u>	Sortierte Liste der Attributknoten
	<u>CteAttribTreeCI</u>	Sortierte Liste der Attributknoten (ohne Unterscheidung der Groß-/Kleinschreibung)

Resultat handle Element, Liste oder Knoten

Siehe Verwandte Befehle, CteClose(),
CteInsertItem(), CteInsertNode(), Beispiel

Mit dieser Funktion wird ein neues Objekt einer dynamischen Struktur erzeugt. Der Typ des erzeugten Objekts wird in (int1) angegeben.

Die Eigenschaften eines neuen Objektes sind leer bzw. NULL.

Beispiele:

```
// Element erzeugtItem # CteOpen(_CteItem);// Verkettete Liste erzeugtList # CteOpen(_CteList
```

Bei Knoten können Optionen (int2) übergeben werden. Werden keine Optionen angegeben, werden die Optionen CteChildList | CteAttribList | CteAttribTree verwendet. Um einen Knoten ohne Listen zu erzeugen muss 0 übergeben werden.

Die Optionen werden in die Eigenschaft Flags übernommen.

Beispiele:

Kontakt

// Knoten mit verkettete Liste für untergeordnete Knoten// und verketteter und sortierter Liste

Die Funktion gibt den Deskriptor des erzeugten Objekts zurück. Konnte das Objekt nicht angelegt werden, gibt der Befehl 0 zurück.

Mögliche Laufzeitfehler:

_ErrValueInvalid Objekttyp (int1) oder Optionen (int2) unbekannt

obj -> CteRead(int1[,
handle2[, alpha3]]) :



handle

Element/Knoten lesen

obj Liste/Knoten

int1 Suchmodus (siehe Text)

handle2 Referenzelement (optional)

alpha3 Suchname (optional)

Resultat handle Element-Deskriptor

Verwandte Befehle,

Siehe CteInsert(), CteDelete(),

Beispiel

Mit diesem Befehl können Elemente in einer dynamischen Struktur gelesen werden.

Folgende Suchmodi (int1) sind sowohl für Listen (CteList und CteTree) als auch für Knoten (CteNode) möglich:

- CteFirst

Erstes Element lesen

- CteLast

Letztes Element lesen

- CtePrev

Element vor Referenzelement lesen

- CteNext

Element nach Referenzelement lesen

Soll eine Liste eines CteNode-Objekts durchsucht werden, muss der Suchmodus mit einer der folgenden Konstanten kombiniert werden:

- CteChildList

verkettete Liste der untergeordneten Knoten durchsuchen

- CteChildTree

Sortierte Liste der untergeordneten Knoten durchsuchen

- CteAttribList

verkettete Liste der Attribute durchsuchen

- CteAttribTree

Sortierte Liste der Attribute durchsuchen

Ist bei CtePrev oder CteNext das Referenzelement nicht angegeben oder nicht in der Liste/dem Knoten (obj) enthalten, ist das Resultat 0.

Beispiel:

```
// Alle Elemente einer Liste durchlaufenfor tItem # tList->CteRead(_CteFirst);loop tItem # tLI
```

Diese Operationen können mit _CteSearch oder _CteSearchCI erweitert werden. Dabei wird der Name des Elements zusätzlich auf Ähnlichkeit mit (alpha3) überprüft und dadurch nach dem ersten Element gesucht, das dem Kriterium entspricht. Durch die zusätzliche Angabe von _CteCustom kann anstatt der Eigenschaft Name die Eigenschaft Custom überprüft werden.

Beispiele:

```
// Erstes Element mit Namen 'Test' lesenItem # tList->CteRead(_CteFirst | _CteSearch, 0, 'test')
```

Der direkte Zugriff über den Namen ist nur bei sortierten Listen (CteTree) und Knoten (CteNode) möglich. Dabei wird entweder der Name im Referenzelement (int2) oder der Name in (alpha3) als Suchwert verwendet, wobei (alpha3) Vorrang hat.

Folgende Suchmodi (int1) sind für sortierte Listen und Knoten mit sortierten Listen möglich:

- _CteCmpL

Element mit nächstkleinerem Namen lesen

- _CteCmpLE

Element mit gleichen oder nächstkleinerem Namen lesen

- _CteCmpE

Element mit gleichen Namen lesen

- _CteCmpGE

Element mit gleichen oder nächstgrößerem Namen lesen

- _CteCmpG

Element mit nächstgrößerem Namen lesen



Diese Modi können nicht mit _CteSearch, _CteSearchCI und _CteCustom kombiniert werden.

Beispiele:

```
// Element mit Namen von tRefItem lesenItem # tList->CteRead(_CteCmpE, tRefItem);// Erstes Element
```

Bei Knoten muss neben dem Suchmodus auch die zu durchsuchende Liste angegeben werden:

- _CteChildList

Suche in verketteter Liste der untergeordneten Knoten

- _CteChildTree

Suche in sortierter Liste der untergeordneten Knoten

- _CteAttribList

Kontakt

Suche in verketteter Liste der Attributknoten

- CteAttribTree

Suche in sortierter Liste der Attributknoten

Beispiele:

```
// Ersten Knoten in verketteter Liste der untergeordneten Knoten suchen.tNode # tNodeRoot->CteRea
```

Für Bäume besteht zusätzlich die Möglichkeit mit CteNodePath oder CteNodePathCI einen Knoten über mehrere Ebenen zu suchen. Dabei wird immer die sortierte Liste des Knotens verwendet. Falls keine sortierte Liste vorhanden ist, wird die verkettete Liste durchsucht. Die Suchoptionen werden in der letzten Ebene angewendet.

Die Separatoren für die Suche mit CteNodePath und CteNodePathCI können über die Eigenschaften CteNodeSepPath und CteNodeSepAttrib des Sys-Objekts eingestellt werden.



Diese Modi können nicht mit CteCmpL, CteCmpLE, CteCmpE, CteCmpGE und CteCmpG kombiniert werden.

Beispiele:

```
// Untergeordneten Knoten mit Namen "child" im untergeordneten Knoten mit Namen "parent" suchen.t
```

Das Resultat bei allen Leseoperationen ist 0, wenn kein Element gefunden wurde, sonst wird der Deskriptor des gefundenen Elements zurückgegeben.

Mögliche Laufzeitfehler:

ErrHdlInvalid Liste/Knoten (obj) oder Element (int2) ungültig

ErrValueInvalid Suchmodus (int1) unbekannt

Konstanten für dynamische Strukturen

Konstanten für dynamische Strukturen

Befehle für

Siehe dynamische
Strukturen

- _CteAfter
- _CteAttribList
- _CteAttribTree
- _CteBefore
- _CteChildList
- _CteChildTree
- _CteCmpE
- _CteCmpG
- _CteCmpGE
- _CteCmpL
- _CteCmpLE
- _CteCount
- _CteCustom
- _CteFirst
- _CteItem
- _CteLast
- _CteList
- _CteNext
- _CteNode
- _CteNodePath
- _CteNodePathCI
- _CteNodeValueRead
- _CteNodeValueWrite
- _CtePrev
- _CteSearch
- _CteSearchCI
- _CteTree
- _CteTreeCI

_CteAfter

Element nach Referenzelement einfügen

Wert 6 / 0x0006

Verwandte

Siehe Befehle,

CteInsert(),

_CteBefore

Option bei CteInsert(), CteInsertItem() und CteInsertNode() durch die ein Element nach dem Referenzelement in eine Liste/Knoten eingefügt werden kann.

_CteAttrib
Attributknoten
Wert 2.097.152 /
0x00200000

Verwandte
Befehle,

Siehe CteInsertNode(),
CteDelete(),
CteClear()

Wird bei einer Anweisung diese Option angegeben, betrifft die Anweisung die Attributknoten.

_CteAttribList
Verkettete Liste der Attributknoten

Wert 65.536 /
0x00010000

Verwandte
Siehe Befehle,
CteOpen(),
CteRead()

Wird diese Option bei der Anweisung CteOpen() angegeben, wird ein Knoten-Objekt mit einer verketteten Liste für Attributknoten erzeugt.

Bei der Übergabe bei der Anweisung CteRead(), wird die verkettete Liste der Attributknoten durchsucht.

_CteAttribTree
Sortierte Liste der Attributknoten
Wert 131.072 /
0x00020000

Verwandte
Siehe Befehle,
CteOpen(),
CteRead()

Wird diese Option bei der Anweisung CteOpen() angegeben, wird ein Knoten-Objekt mit einer sortierten Liste für Attributknoten erzeugt.

Bei der Übergabe bei der Anweisung CteRead(), wird die sortierte Liste der Attributknoten durchsucht.

_CteAttribTreeCI

Sortierte Liste der Attributknoten ohne Unterscheidung der Groß-/Kleinschreibung

Wert 393.216 /
0x00060000

Verwandte

Siehe Befehle,

CteOpen()

Option bei CteOpen(). Der Knoten wird mit einer sortierten Liste für Attributknoten erzeugt. In der Liste findet keine Unterscheidung der Groß-/Kleinschreibung statt.

_CteBefore
Element vor Referenzelement einfügen
Wert 5 / 0x0005

Verwandte
Siehe Befehle,
CteInsert(),
_CteAfter

Option bei CteInsert(), CteInsertItem() und CteInsertNode() durch die ein Element vor dem Referenzelement in eine Liste/Knoten eingefügt werden kann.

_CteChild
Untergeordnete Knoten
Wert 1.048.576 /
0x00100000

Verwandte
Befehle,

Siehe CteInsertNode(),
CteDelete(),
CteClear()

Wird bei einer Anweisung diese Option angegeben, betrifft die Anweisung die untergeordneten Knoten.

_CteChildList
Verkettete Liste der untergeordneten Knoten

Wert 4.096 /
0x00001000

Verwandte

Siehe Befehle,
CteOpen(),
CteRead()

Wird diese Option bei der Anweisung CteOpen() angegeben, wird ein Knoten-Objekt mit einer verketteten Liste für untergeordnete Knoten erzeugt.

Bei der Übergabe bei der Anweisung CteRead(), wird die verkettete Liste der untergeordneten Knoten durchsucht.

_CteChildTree

Sortierte Liste der untergeordneten Knoten

Wert 8.192 /
0x00002000

Verwandte

Siehe Befehle,
CteOpen(),
CteRead()

Wird diese Option bei der Anweisung CteOpen() angegeben, wird ein Knoten-Objekt mit einer sortierten Liste für untergeordnete Knoten erzeugt.

Bei der Übergabe bei der Anweisung CteRead(), wird die sortierte Liste der untergeordneten Knoten durchsucht.

_CteChildTreeCI

Sortierte Liste der untergeordneten Knoten ohne Unterscheidung der Groß-/Kleinschreibung

Wert 24.576 /
0x00006000

Verwandte

Siehe Befehle,

CteOpen()

Option bei CteOpen(). Der Knoten wird mit einer sortierten Liste für untergeordnete Knoten erzeugt. In der Liste findet keine Unterscheidung der Groß-/Kleinschreibung statt.

_CteCmpE

Element mit gleichem Namen lesen

Wert 13 /
0x000D

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die das Element mit dem gleichen Namen gelesen werden kann.

_CteCmpG

Element mit nächstgrößerem Namen lesen

Wert 15 /
0x000F

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die das Element mit dem nächstgrößerem Namen gelesen werden kann.

_CteCmpGE

Element mit gleichem oder nächstgrößerem Namen lesen

Wert 14 /
0x000E

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die das Element mit dem gleichen oder nächstgrößeren Namen gelesen werden kann.

_CteCmpL

Element mit nächstkleinerem Namen lesen

Wert 11 /
0x000B

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die das Element mit dem nächstkleineren Namen gelesen werden kann.

_CteCmpLE

Element mit gleichem oder nächstkleinerem Namen lesen

Wert 12 /
0x000C

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die das Element mit dem gleichen oder nächstkleineren Namen gelesen werden kann.

CteCount

Elementanzahl einer Liste oder eines Knotens ermitteln

Wert 16 /
0x0010

Verwandte

Siehe Befehle,

CteInfo()

Option bei CteInfo() durch die die Anzahl der Elemente einer Liste ermittelt werden kann.



Bei CteNode-Objekten wird die Anzahl aller Elemente in der Struktur gezählt. Sollen nur die direkten Kinder / Attribute gezählt werden, können die Eigenschaften ChildCount und AttribCount verwendet werden.

_CteCustom
Element über Custom-Eigenschaft suchen
Wert 1.024 /
0x00000400

Verwandte
Siehe Befehle,
CteRead(),
_CteSearch

Option bei CteRead() durch die ein Element über die Eigenschaft Custom gesucht werden kann.

_CteFirst

Erstes Element

Wert 1 / 0x0001

Verwandte

Befehle,

Siehe CteInsert(),

CteRead(),

_CteBefore

Option bei CteInsert() und CteRead() durch die ein Element am Anfang der Liste eingefügt bzw. das erste Element einer Struktur gelesen werden kann.

_CteItem
Element erzeugen
Wert 0 / 0x0000

Verwandte

Siehe Befehle,
CteOpen()

Option bei CteOpen() durch die ein neues Element erzeugt wird. Das Objekt besitzt folgende Eigenschaften:

- Name
- ID
- Custom

Das Objekt kann ebenfalls mit den Anweisungen CteInsertItem() erzeugt werden.

_CteLast

Letztes Element

Wert 2 / 0x0002

Verwandte

Befehle,

Siehe CteInsert(),

CteRead(),

_CteAfter

Option bei CteInsert() und CteRead() durch die ein Element am Ende der Liste eingefügt bzw. das letzte Element einer Struktur gelesen werden kann.

_CteList

Verkettete Liste erzeugen/ermitteln

Wert 1 / 0x0001

Verwandte

Siehe Befehle,
CteOpen(),
CteInfo()

Option bei CteOpen() durch die eine neue verkettete Liste erzeugt wird. Das Objekt besitzt folgende Eigenschaften:

- Name
- ID
- Custom

_CteNext

Element nach Referenzelement lesen

Wert 4 / 0x0004

Verwandte

Siehe Befehle,
CteRead(),
_CtePrev

Option bei CteRead() durch die das Element nach dem Referenzelement gelesen werden kann.

_CteNode
Knoten erzeugen
Wert 4 / 0x0004

Verwandte

Siehe Befehle,
CteOpen()

Option bei CteOpen() durch die ein neuer Knoten erzeugt wird. Der Knoten besitzt folgende Eigenschaften:

- Name
- ID
- Custom
- Flags
- Parent
- Type
- Value...
- AttribCount
- ChildCount

Nähere Informationen befinden sich im Abschnitt Dynamische Strukturen.

_CteNodePath

Knoten über Pfad aus Name-Eigenschaft suchen

Wert 7 / 0x0007

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die ein Knoten über den Pfad aus der Eigenschaft Name gesucht werden kann.

Die Separatoren für Pfad und Attribut können über die Eigenschaften CteNodeSepPath und CteNodeSepAttrib des _Sys-Objekts gesetzt werden.

_CteNodePathCI

Knoten über Pfad aus Name-Eigenschaft ohne Unterscheidung der Groß-/Kleinschreibung suchen

Wert 8 / 0x0008

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die ein Knoten über den Pfad aus der Eigenschaft Name ohne Unterscheidung der Groß-/Kleinschreibung gesucht werden kann.

Die Separatoren für Pfad und Attribut können über die Eigenschaften CteNodeSepPath und CteNodeSepAttrib des _Sys-Objekts gesetzt werden.

_CteNodeValueRead

Daten in Memory-Objekte einlesen

Wert 0 / 0x0000

Verwandte Befehle,

Siehe CteNodeValueAlpha(),

_CteNodeValueWrite

Option bei CteNodeValueAlpha() durch die Daten eines CteNode-Objektes in ein Memory-Objekt kopiert werden können.

_CteNodeValueWrite

Daten des Memory-Objektes schreiben

Wert 1 / 0x0001

Verwandte Befehle,

Siehe CteNodeValueAlpha(),

_CteNodeValueRead

Option bei CteNodeValueAlpha() durch die Daten eines Memory-Objektes in ein CteNode-Objekt kopiert werden können.

_CtePrev

Element vor Referenzelement lesen

Wert 3 / 0x0003

Verwandte

Siehe Befehle,

CteRead(),

_CteNext

Option bei CteRead() durch die das Element vor dem Referenzelement gelesen werden kann.

_CteSearch

Element über Name- oder Custom-Eigenschaft suchen

Wert 256 /
0x00000100

Verwandte

Siehe Befehle,

CteRead()

Option bei CteRead() durch die ein Element über die Eigenschaft Name gesucht werden kann. Wird zusätzlich die Option _CteCustom angegeben, findet die Suche über die Eigenschaft Custom statt.

_CteSearchCI

Element über Name- oder Custom-Eigenschaft suchen (ohne Unterscheidung der Groß-/Kleinschreibung)

Wert 512 /
0x00000200

Verwandte

Befehle,

Siehe CteRead(),

_CteSearch,

_CteCustom,

Option bei CteRead() durch die ein Element über die Eigenschaft Name, ohne Unterscheidung der Groß-/Kleinschreibung, gesucht werden kann.

_CteTree

Sortierte Liste erzeugen/ermitteln

Wert 2 / 0x0002

Verwandte

Siehe Befehle,
CteOpen(),
CteInfo()

Option bei CteOpen() durch die eine neue sortierte Liste erzeugt wird. Das Objekt besitzt folgende Eigenschaften:

- Name
- ID
- Custom

_CteTreeCI

Baum ohne Unterscheidung von Groß-/Kleinschreibung erzeugen

Wert 3 / 0x0003

Verwandte

Siehe Befehle,

CteOpen()

Option bei CteOpen() durch die eine neue sortierte Liste erzeugt wird. Das Objekt besitzt folgende Eigenschaften:

- Name
- ID
- Custom

In dieser sortierten Liste werden Groß- und Kleinbuchstaben nicht unterschieden.

Befehle für Memory-Objekte

Liste der Befehle und Konstanten zur Bearbeitung von Memory-Objekten

Befehlsgruppen,

Siehe Befehlsliste,

Memory


Befehle

- BinReadMem
- BinWriteMem
- FsiReadMem
- FsiWriteMem
- MemAllocate
- MemCnv
- MemCompress
- MemCopy
- MemDecrypt
- MemEncrypt
- MemFindByte
- MemFindStr
- MemFree
- MemGenKeyPair
- MemHash
- MemHMAC
- MemReadByte
- MemReadStr
- MemResize
- MemSign
- MemUncompress
- MemVerify
- MemWriteByte
- MemWriteStr
- MsxReadMem
- MsxWriteMem
- PdfTextExtractMem
- SckReadMem
- SckWriteMem
- WinRtfPicInsertMem

Konstanten

- _ComprFmtDeflate
- _ComprFmtGzip
- _ComprFmtZlib
- _ComprLvlDefault
- MemAppend
- MemCipherAES128
- MemCipherAES192
- MemCipherAES256
- MemCipherModeCBC
- MemCipherModeCTR
- MemCipherModeGCM

- MemCipherModeOFB
- MemCipherNoPadding
- MemCipherRSA
- MemDataLen
- MemDecBase64
- MemDecHex
- MemEncBase64
- MemEncHex
- MemIVBase64
- MemIVHex
- MemIVMem
- MemKeyAsymPrivate
- MemKeyAsymPublic
- MemKeyBase64
- MemKeyHex
- MemKeyMem
- MemObjSize
- MemPaddingRSAOaep
- MemPaddingRSAPkcs1
- MemSignatureBase64
- MemSignatureHex
- MemSignDSA
- MemSignRSA

obj -> BinReadMem(handle1[, alpha2[,  int3]]) : int

Binäres Objekt in Memory-Objekt lesen

obj Deskriptor eines binären Objekts

handle1 Deskriptor eines Memory-Objekts

alpha2 Verschlüsselungscode (optional)

Optionen (optional)

int3 __BinErrorDecryption Eindeutiger
Fehlerwert wenn
Entschlüsselungscode
falsch

Resultat int Fehlerwert 

Siehe Verwandte Befehle, BinExport(),
BinWriteMem()

Mit dieser Funktion wird der Inhalt des binären Objekts (obj) in das Memory-Objekt (handle1) eingelesen. Falls der Objekthinhalte verschlüsselt gespeichert wurde, muss in (alpha2) der entsprechende Verschlüsselungscode angegeben werden. Bei einem inkorrekten Code ist das Resultat __ErrBinData. Falls das Objekt leer ist, wird __ErrBinNoData zurückgeliefert. In allen anderen Fällen ist das Resultat __ErrOk.


Der Wert der Eigenschaft Len entspricht nach der Operation der unkomprimierten Datengröße des binären Objekts.

Optional kann als Option (int3) __BinErrorDecryption angegeben werden um bei einem falschen Entschlüsselungscode __ErrBinDecryption statt dem allgemeinen Fehlerwert, __ErrBinData, zu erhalten.

Mögliche Laufzeitfehler:

__ErrHdlInvalid Der in (obj) oder (handle1) übergebene Deskriptor ist ungültig.

__ErrStringOverflow Das zu lesenden binäre Objekt ist größer als das Memory-Objekt.

obj -> BinWriteMem(handle1[, int2[, alpha3]]) 

: int


Binäres Objekt aus Memory-Objekt schreiben

obj Deskriptor eines
 binären Objekts

handle1 Deskriptor eines
 Memory-Objekts

int2 Kompressionsstufe
 (optional)

alpha3 Verschlüsselungs-Code
 (optional)

Resultat int Fehlerwert 

Siehe Verwandte Befehle,
BinReadMem()

Mit dieser Funktion wird der komplette Inhalt des Memory-Objekts (handle1) in das binäre Objekt (obj) geschrieben. Ein bereits bestehender Inhalt wird dabei überschrieben. Das Objekt muss dazu exklusiv gesperrt sein (siehe BinLock oder BinSingleLock).

Optional kann der Inhalt durch Übergabe einer der Stufen 1 bis 4 in (int2) komprimiert werden. Eine Kompressionsstufe sollte nicht bei Dateien angegeben werden, die sich nicht weiter komprimieren lassen. Dazu gehören vor allem gepackte Dateiformate (.zip, .rar usw.) und komprimierte Multimedia-Formate (.jpg, .mov, .mp3 usw.).

Optional kann das Objekt mit einer symmetrischen Verschlüsselung gespeichert werden. Dazu wird ein entsprechender Verschlüsselungscode mit bis zu 64 Zeichen in (alpha2) übergeben (siehe StrEncrypt()). Es ist zu beachten, dass ohne diesen Code der Objekthinhalte nicht mehr gelesen werden kann.

Das Resultat ist ErrOk, wenn die Daten korrekt geschrieben werden konnten. Es können folgende Fehlerresultate auftreten:


ErrBinNoLock Das binäre Objekt ist nicht exklusiv gesperrt.

ErrBinNoData Das Memory-Objekt enthält keine Daten

rDeadlock Verklemmung aufgetreten

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) oder (handle1) angegebene Deskriptor ist ungültig.

obj -> FsiReadMem(handle1, )
int2, int3) : int


Datei in Memory-Objekt lesen

obj Datei-Deskriptor

handle1 Deskriptor des Memory-Objekts

int2 Position im Memory-Objekt

int3 Anzahl der Bytes

Resultat int Anzahl der gelesenen Bytes 
oder Fehlerwert

Siehe Verwandte Befehle, FsiWriteMem()

Mit dieser Funktion werden Daten aus der externen Datei (obj) ab der aktuellen Position gelesen (siehe FsiSeek() bzw. FsiSeek64()). In (handle1) muss der Deskriptor eines Memory-Objekts angegeben werden. Aus der Datei werden maximal (int3) Bytes gelesen und ab der Position (int2) in das Memory-Objekt übertragen. Gegebenenfalls wird der Wert der Eigenschaft Len erhöht.


Werden aus der externen Datei Zeichenketten in das Objekt gelesen, muss die Eigenschaft Charset des Memory-Objekts auf den Zeichensatz der externen Datei gesetzt werden, damit die Zeichenketten korrekt verarbeitet werden können. Eine Konvertierung der Zeichenkodierung aufgrund der Angaben bei FsiOpen() findet nicht statt.

Das Resultat gibt die Anzahl der gelesenen Bytes zurück. Ist das Resultat negativ, ist ein Fehler aufgetreten und das Resultat enthält den Fehlerwert (_ErrFsi...). Der Fehlerwert des Betriebssystems kann über die Eigenschaft FsiError abgefragt werden.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der Datei-Deskriptor (obj) oder der Deskriptor des Memory-Objekts ist ungültig.

_ErrValueRange Die übergebenen Werte in (int2) oder (int3) sind ungültig.

obj -> FsiWriteMem(handle1, int2, )
int3) : int


Memory-Objekt in Datei schreiben

obj Datei-Deskriptor

handle1 Deskriptor des Memory-Objekts

int2 Position im Memory-Objekt

int3 Anzahl der zu schreibenden Bytes

Resultat int Anzahl der geschriebenen Bytes 
oder Fehlerwert

Siehe Verwandte Befehle, FsiOpen()

Mit dieser Funktion werden Daten in die externe Datei (obj) ab der aktuellen Position geschrieben (siehe FsiSeek() bzw. FsiSeek64()). In (handle1) muss der Deskriptor eines Memory-Objekts angegeben werden. Aus dem Memory-Objekt werden ab der Position (int2) eine Anzahl von (int3) Bytes in die Datei geschrieben.


Sollen Zeichenketten in die externen Datei geschrieben, muss die Eigenschaft Charset des Memory-Objekts auf den Zeichensatz der externen Datei gesetzt werden, damit die Zeichenketten beim Einfügen in das Memory-Objekt (siehe MemWriteStr()) korrekt verarbeitet werden können. Eine Konvertierung der Zeichenkodierung aufgrund der Angaben bei FsiOpen() findet nicht statt.

Das Resultat gibt die Anzahl der geschriebenen Bytes zurück. Ist das Resultat negativ ist ein Fehler aufgetreten und das Resultat enthält den Fehlerwert (_ErrFsi...). Der Fehlerwert des Betriebssystems kann über die Eigenschaft FsiError abgefragt werden.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Datei-Deskriptor (obj) oder der Deskriptor des Memory-Objekts ist ungültig.

_ErrValueRange Der in (int2) oder (int3) übergebene Wert ist außerhalb des zulässigen Bereichs.

MemAllocate(int1) : 

handle

Memory-Objekt erzeugen

int1 Speichermenge in Byte oder
 _MemAutoSize

Resultat handle Deskriptor auf das
 Objekt

Siehe Verwandte Befehle, Memory,
 MemFree()

Mit dieser Funktion wird ein Memory-Objekt angelegt. Die Größe muss dabei im Bereich von 1 Byte bis 512 MB (bei 32-Bit-Prozessen) bzw. 2 GB (bei 64-Bit-Prozessen) liegen. Der Speicherbereich wird im Adressraum des aktuellen Prozesses angelegt. Der Adressraum ist bei 32-Bit-Prozessen normalerweise 2 GB groß, in speziellen Fällen auch mehr (siehe ProcessMemoryLimitMB). Als Rückgabewert wird ein Deskriptor auf das Objekt zurückgegeben. Konnte das Objekt nicht angelegt werden, weil nicht ausreichend Speicher vorhanden ist, wird _ErrOutOfMemory zurückgegeben.

Bei der Größe des Objekts können anstelle der Anzahl der Bytes auch folgende Konstanten angegeben werden:

<u>_Mem1K</u>	<u>_Mem1M</u>
<u>_Mem2K</u>	<u>_Mem2M</u>
<u>_Mem4K</u>	<u>_Mem4M</u>
<u>_Mem8K</u>	<u>_Mem8M</u>
<u>_Mem16K</u>	<u>_Mem16M</u>
<u>_Mem32K</u>	<u>_Mem32M</u>
<u>_Mem64K</u>	<u>_Mem64M</u>
<u>_Mem128K</u>	<u>_Mem128M</u>
<u>_Mem256K</u>	<u>_Mem256M</u>
<u>_Mem512K</u>	<u>_Mem512M</u>

Anstelle der Größe des Objekts kann die Konstante _MemAutoSize angegeben werden. Die Größe des Objekts wird dann automatisch vergrößert, wenn entsprechende Daten in das Objekt geschrieben werden. Beim Vergrößern des Objekts muss der Speicherinhalt kopiert werden, daher dauert das Schreiben dann länger als bei Objekten mit statischer Größe. Eine Verkleinerung des Objekts findet nicht automatisch statt.


Mit dem Befehl MemResize(), kann die Größe nachträglich verändert werden.

Um das Objekt wieder zu entfernen und den Speicher freizugeben, muss die Anweisung MemFree() verwendet werden.

Folgende Fehlerwerte sind möglich:

_ErrValueInvalid Als Größe (int1) wurde ein Wert ≤ 0 angegeben.
_ErrLimitExceeded Bei einem 32-Bit-Prozess wurde eine Größe (int1) von mehr als 512 MB angegeben.

ErrOutOfMemory Der Speicher konnte nicht angefordert werden.

obj -> MemCnv(handle1, )
int2) : int


Memory-Objekt konvertieren

obj Deskriptor des Quell-Memory-Objekts

handle1 Deskriptor des Ziel-Memory-Objekts


int2 Art der Konvertierung (siehe Text)

Fehlerwert

Resultat int ErrOk Kein Fehler 
ErrData Ungültige Zeichen im
Quell-Memory-Objekt

Siehe Verwandte Befehle

Diese Funktion überträgt und konvertiert den Inhalt des Memory-Objekts (obj) in den Speicherbereich des Memory-Objekts (handle1).

 Das Ziel-Memory-Objekt (handle1) muss mindestens 1024 Byte groß sein. Alternativ kann es auch mit MemAutoSize angelegt werden.

Mit (int2) kann die Art der Konvertierung angegeben werden:

- **Zeichensatzumwandlung**

Wird ein Zielzeichensatz angegeben (siehe Charset), erfolgt eine Zeichensatzumwandlung. Als Quellzeichensatz wird die Eigenschaft Charset des Quell-Objekts (obj) verwendet. Es kann zwischen allen unterstützten Zeichensätzen konvertiert werden. Dabei werden Zeichen, die nicht im Zielzeichensatz darstellbar sind, durch ein Fragezeichen ersetzt. Die Eigenschaft Charset des Ziel-Objekts (handle1) wird auf den übergebenen Wert gesetzt. Das Resultat ist immer ErrOk. Zeichen, die nicht umgewandelt werden können werden als Fragezeichen konvertiert.

- **Kodierung nach Base64**

Durch Angabe der Konstanten MemEncBase64 wird der Inhalt in Base64 kodiert. Die Datenmenge wächst dabei um ein Drittel (33,33%). Es können beliebige binäre Daten kodiert werden.

- **Dekodierung aus Base64**

Mit der Konstanten MemDecBase64 wird der Inhalt aus Base64 dekodiert. Der Inhalt muss aus gültigen Base64-Daten bestehen. Beim Dekodieren werden Whitespace-Zeichen, wie Leerzeichen und Zeilenumbrüche, ignoriert. Bei nicht erlaubten Zeichen wird der Fehlerwert ErrData zurückgegeben. Die Datenmenge schrumpft dabei um ein Viertel (25%).

- **Kodierung in hexadezimale Zeichen**

Durch Angabe der Konstanten MemEncHex wird der Inhalt in hexadezimale Zeichen kodiert. Die Datenmenge wächst dabei auf das doppelte. Es können beliebige binäre Daten kodiert werden.

- **Dekodierung aus hexadezimalen Zeichen**

Mit der Konstanten MemDecHex wird der Inhalt aus hexadezimalen dekodiert. Der Inhalt muss aus gültigen Daten bestehen. Beim Dekodieren werden Whitespace-Zeichen, wie Leerzeichen und Zeilenumbrüche, ignoriert. Bei nicht

Kontakt

erlaubten Zeichen wird der Fehlerwert __ErrData zurückgegeben. Die Datenmenge schrumpft dabei um die Hälfte (50%).

Die Eigenschaft Len des Zielobjekts wird bei allen Operationen neu gesetzt.

Resultat

Als Resultat wird __ErrOk zurückgegeben, wenn die Konvertierung erfolgreich war. Bei Konvertierungen mit nicht erlaubten Zeichen wird der Fehlerwert __ErrData zurückgegeben.

Beispiel:

```
tMemOrg # MemAllocate(_MemAutoSize);tMemOrg->MemWriteStr(1, '...'); tMemCopy # MemAllocate((tMemOrg->Len)*2);
```

Mögliche Laufzeitfehler:

<u>__ErrHdlInvalid</u>	Der in (obj) oder (handle1) übergebene Deskriptor ist ungültig.
<u>__ErrValueInvalid</u>	In (int2) wurde kein zulässiger Wert übergeben. Das Zielobjekt ist bei der Zeichensatzumwandlung nicht groß genug (<u>Size</u>), um das umgewandelte Objekt aufzunehmen. Es muss mindestens 1024 Byte groß sein oder über automatische Vergrößerung (<u>__MemAutoSize</u>) verfügen.
<u>__ErrStringOverflow</u>	

```
obj -> MemCompress(int1[,  
int2[, int3[, int4[, handle5[,  
int6]]]]) : int
```



Speicherbereich komprimieren

obj Quelle / Ziel (Memory-Objekt)

Kompressionsformat

int1 ComprFmtDeflate DEFLATE-Format

ComprFmtGzip GZIP-Format

ComprFmtZlib ZLIB-Format

int2 Kompressionsstufe (optional)

int3 Quellposition (optional)

int4 Quelllänge (optional)

handle5 Ziel (Memory-Objekt, optional)

int6 Zielposition (optional)

Resultat int Fehlerwert



Verwandte Befehle,

Siehe MemUncompress(),

FsiFileCompress()

Dieser Befehl komprimiert den Inhalt des Memory-Objektes (obj).

Es muss eines der folgenden Kompressionsformate (int1) angegeben werden:

ComprFmtDeflate DEFLATE-Format

ComprFmtGzip GZIP-Format

ComprFmtZlib ZLIB-Format

Als Kompressionsstufe (int2) können Werte zwischen 0 (keine Komprimierung) und 9 (maximale Komprimierung) angegeben werden. Alternativ wird mit ComprLvlDefault die Standard-Komprimierungsstufe angegeben.

Im Parameter (int3) kann die Quellposition angegeben werden. Ist dieser Wert nicht angegeben oder 0, werden die Daten ab Beginn des Memory-Objektes komprimiert.

Der Parameter (int4) gibt die zu komprimierende Länge an. Ist dieser Wert nicht angegeben, 0 oder MemDataLen wird der restliche Inhalt (nach der Quellposition) des Memory-Objektes komprimiert.

Optional kann im Parameter (handle5) ein Ziel-Memory-Objekt angegeben werden. Ist dieses nicht angegeben oder ist es identisch mit dem Quellobjekt (obj), wird in das Quellobjekt geschrieben. Hierbei werden alle Daten verworfen, die hinter dem Dateifuß des Kompressionsformats vorhanden waren.

Zusätzlich kann eine Zielposition (int6) angegeben werden, wenn nicht an den Anfang des Ziel-Objektes geschrieben werden soll. Alle vorhandenen Daten ab der Position werden überschrieben.

Beispiele


```
// Inhalt des Memory-Objektes tMemSrc in neues Memory-Objekt im GZIP-Format komprimieren tMemSrc->
```


Fehlerwerte

Folgende Fehlerwerte können von der Funktion zurückgegeben werden:

ErrOk Kein Fehler aufgetreten.
ErrGeneric Interner Fehler aufgetreten.
Mögliche Laufzeitfehler:

<u>ErrHdlInvalid</u>	Einer der übergeben Deskriptoren (obj) oder (handle5) ist ungültig.
<u>ErrMemExhausted</u>	Nicht genug Speicher vorhanden.
<u>ErrValueInvalid</u>	Im Kompressionsformat (int1) oder Kompressionsstufe (int2) wurde ein ungültiger Wert angegeben. Eine der Längen- oder Positionsangaben (int3), (int4) oder (int6) ist ungültig. Der komprimierte Inhalt kann im ungünstigsten Fall länger werden als der unkomprimierte Inhalt. Übersteigt die Länge den zu Verfügung stehenden Platz des Ziel-Memory-Objektes wird der Laufzeitfehler ebenfalls generiert.
<u>ErrValueRange</u>	

obj -> MemCopy(int1, )
int2, int3[, handle4])

Speicherbereich kopieren

obj Deskriptor des
 Memory-Objekts

int1 Position des
 ersten Bytes
 Anzahl der zu
int2 kopierenden
 Bytes

int3 Zielposition

handle4 Deskriptor des
 Ziel-Objekts

Siehe Verwandte
 Befehle

Dieser Befehl kopiert einen ausgewählten Speicherbereich des Memory-Objekts (obj) an eine andere Stelle des Speichers oder in ein anderes Memory-Objekt (handle4). Der Befehl kopiert (int2) Bytes ab der Position (int1) an die Zielposition (int3).

Überlappende Bereiche werden beim Kopieren korrekt behandelt. Nach dem Kopieren wird der Wert der Eigenschaft Len des Zielobjekt gegebenenfalls erhöht.

Bei einer ungültigen Startposition (int1) oder einer zu großen Länge (int2) wird ein Laufzeitfehler erzeugt.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der übergeben Deskriptor (obj) oder (handle4) ist ungültig.

_ErrValueRange Die angegebenen Startposition (int1) oder die Länge des zu kopierenden Bereiches (int2) sind zu groß.

```
obj -> MemDecrypt(int1, alpha2,
alpha3[, int4[, int5[, handle6[, int7[,
handle8[, handle9[, handle10]]]]]]])
```



: int

Speicherbereich entschlüsseln

obj Chiffre / Klartext (Memory-Objekt)

int1 Optionen (siehe Text)

alpha2 Schlüssel

alpha3 Initialisierungsvektor

int4 Chiffreposition

int5 Chiffrelänge

handle6 Klartext (Memory-Objekt)

int7 Klartextposition

handle8 Schlüssel (Memory-Objekt)

handle9 Initialisierungsvektor (Memory-Objekt)

handle10 Authentifikations-Tag für GCM
(Memory-Objekt)

Fehlerwert

ErrOk

Erfolg

ErrMemKeyInvalid Schlüssel ist
ungültig

ErrMemKeyLength Schlüssel ist zu kurz
oder zu lang für den
Algorithmus

Resultat int ErrMemIVInvalid Initialisierungsvektor
ist ungültig

ErrMemIVLength Initialisierungsvektor
ist zu kurz oder zu
lang für den
Algorithmus

ErrMemDecrypt Entschlüsselung ist
fehlgeschlagen

Siehe Verwandte Befehle, MemEncrypt(), Blog

Dieser Befehl entschlüsselt den Inhalt des Memory-Objektes (obj). Es müssen der gleiche Schlüssel und Initialisierungsvektor angegeben werden, der auch beim Verschlüsseln verwendet wurde.

Folgende Optionen (int1) sind möglich:

- **Verschlüsselungsalgorithmus**

MemCipherAES128 128-Bit-Verschlüsselung mit AES

MemCipherAES192 192-Bit-Verschlüsselung mit AES

MemCipherAES256 256-Bit-Verschlüsselung mit AES

MemCipherRSA Verschlüsselung mit RSA

- **Verschlüsselungsmodus (nicht in Verbindung mit MemCipherRSA)**

MemCipherModeCBC Betriebsmodus Cipher-block chaining

MemCipherModeOFB Betriebsmodus Output feedback

MemCipherModeCTR Betriebsmodus Counter Mode

MemCipherModeGCM Betriebsmodus Galois/Counter Mode

- **Kodierung des Schlüssels**

MemKeyHex Schlüssel liegt als hexadezimal kodierte Zeichenkette vor (alpha2)

MemKeyBase64 Schlüssel liegt Base64-kodierte Zeichenkette vor (alpha2)

MemKeyMem Schlüssel liegt binär vor (handle8)

- **Typ des asymmetrischen Schlüssels (nur in Verbindung mit MemCipherRSA)**

MemKeyAsymPrivate Der Schlüssel (alpha2) / (handle8) ist ein privater asymmetrischer Schlüssel

MemKeyAsymPublic Der Schlüssel (alpha2) / (handle8) ist ein öffentlicher asymmetrischer Schlüssel

- **Kodierung des Initialisierungsvektors (nicht in Verbindung mit MemCipherRSA)**

MemIVHex Initialisierungsvektor liegt als hexadezimal kodierte Zeichenkette vor (alpha3)

MemIVBase64 Initialisierungsvektor liegt als Base64-kodierte Zeichenkette vor (alpha3)

MemIVMem Initialisierungsvektor liegt binär vor (handle9)

- **Padding-Optionen**

MemCipherNoPadding Kein Padding voraussetzen (**nicht** in Verbindung mit MemCipherRSA)

MemPaddingRSAPkcs1 PKCS#1 Padding voraussetzen (**nur** in Verbindung mit MemCipherRSA). Dieser Modus ist weit verbreitet.

MemPaddingRSAOaep OAEP Padding voraussetzen (**nur** in Verbindung mit MemCipherRSA und MemKeyAsymPublic). Dieser Modus sollte für neue Projekte verwendet werden.

Der Parameter (int1) setzt sich aus der Kombination je einer Konstanten der Bereiche Verschlüsselungsalgorithmus, Verschlüsselungsmodus, Kodierung des Schlüssels und Kodierung des Initialisierungsvektors zusammen. Diese können zusätzlich mit den Padding-Optionen kombiniert werden.



Wird die Option MemCipherNoPadding verwendet, muss die zu entschlüsselnde Nachricht 16 Byte lang oder ein Vielfaches davon sein.

Bei der Entschlüsselung mit MemCipherRSA muss ein Padding (MemPaddingRSA...) angegeben werden.

Der Schlüssel wird in (alpha2) bzw. (handle8) angegeben. Je nach Verschlüsselungsalgorithmus muss der Schlüssel eine bestimmte Länge haben:



Algorithmus	Schlüssellänge (Rohdaten)
--------------------	----------------------------------

<u>MemCipherAES128</u>	128 Bit (16 Byte)
------------------------	-------------------

<u>MemCipherAES192</u>	192 Bit (24 Byte)
------------------------	-------------------

<u>MemCipherAES256</u>	256 Bit (32 Byte)
------------------------	-------------------

<u>MemCipherRSA</u>	Die Verschlüsselungsstärke hängt von der Schlüssellänge ab.
---------------------	---

-  Im Gegensatz zu MemCipherAES..., wo der Schlüssel aus zufälligen Zeichen besteht, muss bei MemCipherRSA ein gültiger Schlüssel eines Schlüsselpaars angegeben werden. Dieses Schlüsselpaar kann beispielsweise mit MemGenKeyPair() oder OpenSSL erzeugt werden. Die maximale Länge der verschlüsselbaren Zeichenkette hängt hierbei vom gewählten Padding und der Bitlänge des Schlüssels ab.
Der asymmetrische Schlüssel muss im PKCS #1- oder im X.509-Format vorliegen. Die Begin- und Endmarkierung dürfen **nicht** enthalten sein.
-  Die Ver- und Entschlüsselung mit RSA nimmt wesentlich mehr Zeit in Anspruch als mit AES. Um große Datenmengen zu verschlüsseln sollte ein zufälliger AES-Schlüssel generiert werden, der mit RSA verschlüsselt wird. Der eigentliche Klartext wird mit AES verschlüsselt.

Der Initialisierungsvektor wird in (alpha3) bzw. (handle9) angegeben. Die Länge vom Initialisierungsvektor ist abhängig von der Blocklänge des Verschlüsselungsalgorithmus. Bei AES sind dies 128 Bit (16 Byte). Der Initialisierungsvektor muss nicht geheim gehalten werden. Bei MemCipherRSA wird der Initialisierungsvektor nicht verwendet.

In (int4) und (int5) wird die Position und die Länge der zu entschlüsselnden Daten in (obj) angegeben.

In (handle6) kann ein Ziel-Memory-Objekt angegeben werden, in dass der Klartext gespeichert wird. Ist dieses nicht angegeben, erfolgt die Speicherung in (obj).

Die Zielposition des Klartextes in (handle6) bzw. (obj) kann im Argument (int7) angegeben. Ist das Argument nicht gesetzt, wird an den Anfang des Memory-Objektes geschrieben.

In (handle10) muss ein Memory-Objekt angegeben werden, wenn als Option (int1) MemCipherModeGCM angegeben ist. In dieses Memory-Objekt muss zuvor das Authentifikations-Tag geschrieben, welches bei MemEncrypt() ermittelt wurde. Dieses Tag ist 16 Byte lang.

Mögliche Laufzeitfehler:

<u>ErrHdlInvalid</u>	Einer der übergeben Deskriptoren (obj), (handle6), (handle8), (handle9) oder (handle10) ist ungültig.
<u>ErrMemExhausted</u>	Nicht genug Speicher vorhanden.
<u>ErrNoArgument</u>	Bei der Option <u>MemKeyMem</u> wurde kein Schlüssel (handle8), bei <u>MemIVMem</u> kein Initialisierungsvektor (handle9) oder bei <u>MemCipherModeGCM</u> kein Authentifikations-Tag (handle10) angegeben.
<u>ErrValueInvalid</u>	In den Optionen (int1) wurde eine ungültige oder unvollständige Kombination angegeben.
<u>ErrValueRange</u>	Eine der Längen- oder Positionsangaben (int4), (int5) oder (int7) ist ungültig.

```
obj -> MemEncrypt(int1, alpha2,
alpha3[, int4[, int5[, handle6[, int7[,
handle8[, handle9[, handle10]]]]]]))
```



: int

Speicherbereich verschlüsseln

obj Klartext / Chiffre (Memory-Objekt)

int1 Optionen (siehe Text)

alpha2 Schlüssel

alpha3 Initialisierungsvektor

int4 Klartextposition

int5 Klartextlänge

handle6 Chiffre (Memory-Objekt)

int7 Chiffreposition

handle8 Schlüssel (Memory-Objekt)

handle9 Initialisierungsvektor (Memory-Objekt)

handle10 Authentifikations-Tag für GCM
(Memory-Objekt)

Fehlerwert

ErrOk

Erfolg

ErrMemKeyInvalid

Schlüssel ist ungültig oder der falsche Schlüsseltyp bei asymmetrischen Schlüsseln wurde angegeben

Resultat int ErrMemKeyLength Schlüssel ist zu kurz oder zu lang für den Algorithmus

ErrMemIVInvalid

Initialisierungsvektor ist ungültig

ErrMemIVLength

Initialisierungsvektor ist zu kurz oder zu lang für den Algorithmus

Siehe Verwandte Befehle, MemDecrypt(), Blog

Dieser Befehl verschlüsselt den Inhalt des Memory-Objektes (obj).

Folgende Optionen (int1) sind möglich:

• **Verschlüsselungsalgorithmus**

MemCipherAES128 128-Bit-Verschlüsselung mit AES

MemCipherAES192 192-Bit-Verschlüsselung mit AES

MemCipherAES256 256-Bit-Verschlüsselung mit AES

MemCipherRSA Verschlüsselung mit RSA

• **Verschlüsselungsmodus (nicht in Verbindung mit MemCipherRSA)**

MemCipherModeCBC Betriebsmodus Cipher-block chaining

MemCipherModeOFB Betriebsmodus Output feedback

MemCipherModeCTR Betriebsmodus Counter Mode

MemCipherModeGCM Betriebsmodus Galois/Counter Mode

• **Kodierung des Schlüssels**

MemKeyHex Schlüssel liegt als hexadezimal kodierte Zeichenkette vor (alpha2)

MemKeyBase64 Schlüssel liegt Base64-kodierte Zeichenkette vor (alpha2)

MemKeyMem Schlüssel liegt binär vor (handle8)

• **Typ des asymmetrischen Schlüssels (nur in Verbindung mit MemCipherRSA)**

MemKeyAsymPrivate Der Schlüssel (alpha2) / (handle8) ist ein privater asymmetrischer Schlüssel



Bei Verwendung des privaten Schlüssels erzeugt der gleiche Klartext immer das gleiche Chiffre

MemKeyAsymPublic Der Schlüssel (alpha2) / (handle8) ist ein öffentlicher asymmetrischer Schlüssel

• **Kodierung des Initialisierungsvektors (nicht in Verbindung mit MemCipherRSA)**

MemIVHex Initialisierungsvektor liegt als hexadezimal kodierte Zeichenkette vor (alpha3)

MemIVBase64 Initialisierungsvektor liegt als Base64-kodierte Zeichenkette vor (alpha3)

MemIVMem Initialisierungsvektor liegt binär vor (handle9)

• **Padding-Optionen**

MemCipherNoPadding Kein Padding anwenden (**nicht** in Verbindung mit MemCipherRSA)

MemPaddingRSAPkcs1 PKCS#1 Padding anwenden (**nur** in Verbindung mit MemCipherRSA). Dieser Modus ist weit verbreitet.

MemPaddingRSAOaep OAEP Padding anwenden (**nur** in Verbindung mit MemCipherRSA und MemKeyAsymPublic). Dieser Modus sollte für neue Projekte verwendet werden.

Der Parameter (int1) setzt sich aus der Kombination je einer Konstanten der Bereiche Verschlüsselungsalgorithmus, Verschlüsselungsmodus, Kodierung des Schlüssels und Kodierung des Initialisierungsvektors zusammen. Diese können zusätzlich mit den Padding-Optionen kombiniert werden.




Wird die Option MemCipherNoPadding verwendet, muss die zu verschlüsselnde Nachricht 16 Byte lang oder ein Vielfaches davon sein.

Bei der Verschlüsselung mit MemCipherRSA muss ein Padding (MemPaddingRSA...) angegeben werden. Das Padding füllt den Rest der zu verschlüsselnden Nachricht auf die Schlüsselbitlänge auf. Für das Padding werden mindestens die folgende Anzahl an Zeichen benötigt:

MemPaddingRSAPkcs1 11 Zeichen

MemPaddingRSAOaep 42 Zeichen

 Auch wenn der zu verschlüsselnde Klartext im Memory-Objekt nur aus darstellbaren Zeichen besteht, kann das Chiffre aus nichtdarstellbaren Zeichen (binären Daten) bestehen.

Der Schlüssel wird in (alpha2) bzw. (handle8) angegeben. Je nach Verschlüsselungsalgorithmus muss der Schlüssel eine bestimmte Länge haben:


Algorithmus Schlüssellänge (Rohdaten)

MemCipherAES128 128 Bit (16 Byte)


MemCipherAES192 192 Bit (24 Byte)

MemCipherAES256 256 Bit (32 Byte)

MemCipherRSA Die Verschlüsselungsstärke hängt von der Schlüssellänge ab.

 Im Gegensatz zu MemCipherAES..., wo der Schlüssel aus zufälligen Zeichen besteht, muss bei MemCipherRSA ein gültiger Schlüssel eines Schlüsselpaares angegeben werden. Dieses Schlüsselpaar kann beispielsweise mit MemGenKeyPair() oder OpenSSL erzeugt werden. Die maximale Länge der verschlüsselbaren Zeichenkette hängt hierbei vom gewählten Padding und der Bitlänge des Schlüssels ab.

Der asymmetrische Schlüssel muss im PKCS #1- oder im X.509-Format vorliegen. Die Begin- und Endmarkierung dürfen **nicht** enthalten sein.

 Die Ver- und Entschlüsselung mit RSA nimmt wesentlich mehr Zeit in Anspruch als mit AES. Um große Datenmengen zu verschlüsseln sollte ein zufälliger AES-Schlüssel generiert werden, der mit RSA verschlüsselt wird. Der eigentliche Klartext wird mit AES verschlüsselt.

Der Initialisierungsvektor wird in (alpha3) bzw. (handle9) angegeben. Dieser dient dazu, dass die gleiche Nachricht mit dem gleichen Schlüssel nicht das gleiche Ergebnis liefert. Der Initialisierungsvektor sollte zufällig generiert und nicht mehrfach genutzt werden. Die Länge vom Initialisierungsvektor ist abhängig von der Blocklänge des Verschlüsselungsalgorithmus. Bei AES sind dies 128 Bit (16 Byte). Der Initialisierungsvektor muss nicht geheim gehalten werden. Bei MemCipherRSA wird der Initialisierungsvektor nicht verwendet.

In (int4) und (int5) wird die Position und die Länge der zu verschlüsselnden Daten in (obj) angegeben.

In (handle6) kann ein Ziel-Memory-Objekt angegeben werden, in dass das Chiffre gespeichert wird. Ist dieses nicht angegeben, erfolgt die Speicherung in (obj).

Die Zielposition des Chiffres in (handle6) bzw. (obj) kann im Argument (int7) angegeben werden. Ist das Argument nicht gesetzt, wird an den Anfang des Memory-Objektes geschrieben.

In (handle10) muss ein Memory-Objekt angegeben werden, wenn als Option (int1) MemCipherModeGCM angegeben ist. In dieses Memory-Objekt wird das Authentifikations-Tag geschrieben, welches bei MemDecrypt() zur Authentifizierung der verschlüsselten Nachricht angegeben werden muss. Dieses Tag ist 16 Byte lang.

Mögliche Laufzeitfehler:

ErrHdlInvalid

Kontakt

Einer der übergeben Deskriptoren (obj), (handle6), (handle8), (handle9) oder (handle10) ist ungültig.

ErrMemExhausted Nicht genug Speicher vorhanden.

ErrNoArgument Bei der Option MemKeyMem wurde kein Schlüssel (handle8), bei MemIVMem kein Initialisierungsvektor (handle9) oder bei MemCipherModeGCM kein Memory-Objekt für das Authentifikations-Tag (handle10) angegeben.

ErrStringOverflow Der Plaintext (obj) ist bei der RSA-Verschlüsselung zusammen mit dem Padding länger als der Schlüssel (alpha2) bzw. (handle8).

ErrValueInvalid In den Optionen (int1) wurde eine ungültige oder unvollständige Kombination angegeben.

ErrValueRange Eine der Längen- oder Positionsangaben (int4), (int5) oder (int7) ist ungültig.

obj ->

MemFindByte(int1, int2, byte3) : int

Bytewert suchen

obj Deskriptor des
Memory-Objekts

int1 Startposition

int2 Länge des zu
durchsuchenden
Bereichs

byte3 Suchwert

Resultat int Position des
gesuchten Wertes

Siehe Verwandte Befehle,
MemFindStr()

Der Befehl sucht im Speicherbereich des Memory-Objekts (obj) ab der Startposition (int1) in einem Abschnitt von (int2) Bytes nach dem ersten Vorkommen des Byte-Werts (byte3). Das Resultat ist 0, wenn der Bytewert in diesem Abschnitt nicht vorkommt, ansonsten wird die Position des gefundenen Bytes zurückgeliefert.

Soll das gesamte Objekt nach einem Bytewert durchsucht werden, kann in (int2) die Datenmenge des Memory-Objekts (siehe Eigenschaft Len) angegeben werden. Soll zu Beginn des Objekts einige Bytes übersprungen werden, müssen diese von der Anzahl der zu durchsuchenden Bytes abgezogen werden.

Beispiel:

```
// Suche ab Anfang des ObjektstStartByte # 1;tSearchBytes # tMemory->spLen;tRes # tMemory->MemFindByte(1, tMemory->spLen, 10);
```


Folgendes Beispiel ermittelt die Anzahl der Zeilenwechsel in einem Memory-Objekt:

```
for tPos # tMemory->MemFindByte(1, tMemory->spLen, 10);loop tPos # tMemory->MemFindByte(tPos + 1, tMemory->spLen, 10);
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

ErrValueRange Die angegebene Startposition (int1) oder zu durchsuchende Bereich (int2) sind außerhalb des zulässigen Bereichs oder in (byte3) wurde kein byte-Wert übergeben.

obj -> MemFindStr(int1, , int2, alpha3[, int4]) : int

Zeichenkette suchen

obj Deskriptor des Memory-Objekts

int1 Startposition

int2 Anzahl der zu durchsuchenden
Bytes

alpha3 zu suchende Zeichenkette

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 ersten Zeichens

Siehe Verwandte Befehle, MemFindByte()

Diese Funktion sucht im Speicherbereich des Memory-Objekts (obj) ab der Position (int1) in (int2) Bytes nach dem ersten Vorkommen der Zeichenkette (alpha3). Das Resultat ist 0, wenn der Wert nicht gefunden wurde. Ansonsten wird die Position des ersten gefundenen Werts zurückgeliefert.

Soll das gesamte Objekt nach einer Zeichenkette durchsucht werden, kann in (int2) die Datenmenge des Memory-Objekts (siehe Eigenschaft Len) angegeben werden. Soll zu Beginn des Objekts einige Bytes übersprungen werden, müssen diese von der Anzahl der zu durchsuchenden Bytes abgezogen werden.

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.

Beispiel:

```
// Suche ab Anfang des ObjektstStartByte # 1; tSearchBytes # tMemory->spLen; tRes # tMemory->MemFindStr(1, tMemory->spLen, StrChar(13) + StrChar(10));
```

Folgendes Beispiel ermittelt die Anzahl der Zeilenwechsel in einem Memory-Objekt:

```
for    tPos # tMemory->MemFindStr(1, tMemory->spLen, StrChar(13) + StrChar(10)); loop    tPos # tMemory->MemFindStr(tPos, tMemory->spLen, StrChar(13) + StrChar(10));
```

Kontakt

Abhängig vom verwendeten Zeichensatz entspricht ein Byte nicht unbedingt einem Zeichen. Bei der Verwendung zum Beispiel von UTF-8 ist ein Zeichen zwischen einem und vier Bytes lang. Die Suche kann nur am Beginn eines Zeichens gestartet werden, sonst wird der Laufzeitfehler __ErrValueRange erzeugt.

Mögliche Laufzeitfehler:

<u>__ErrHdlInvalid</u>	Der in (obj) übergebene Deskriptor ist ungültig.
<u>__ErrValueRange</u>	Die in (int1) oder (int2) übergebenen Werte liegen außerhalb des zulässigen Bereichs oder die Suche beginnt nicht an einer Zeichengrenze.

obj -> MemFree()



Memory-Objekt entfernen

obj Deskriptor eines
 Memory-Objekts

Verwandte


Siehe Befehle,

MemAllocate()

Mit dieser Funktion wird ein mittels MemAllocate() angelegtes Memory-Objekt wieder entfernt und der Speicherbereich freigegeben. In (obj) wird der Deskriptor des Objekts übergeben.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der übergeben Deskriptor ist ungültig.

obj -> MemGenKeyPair(handle1, int2) : 

int

Asymmetrisches Schlüsselpaar erzeugen

obj Privater
asymmetrischer
Schlüssel
(Memory-Objekt)
Öffentlicher
handle1 asymmetrischer
Schlüssel
(Memory-Objekt)
int2 Schlüssellänge in
Bit

Resultat int Fehlerwert
ErrOk Erfolg

Siehe Verwandte
Befehle,
MemEncrypt(),
MemDecrypt()

Dieser Befehl erzeugt ein RSA-Schlüsselpaar für asymmetrische Verschlüsselung (siehe MemEncrypt() und MemDecrypt() mit MemCipherRSA) bzw. Signierung (siehe MemSign() und MemVerify() mit MemSignRSA).



Die Bitlänge sollte eine Zweierpotenz sein ≥ 2048 sein (z. B. 2048, 4096)

Je höher die Schlüssellänge ist, desto länger dauert die Ver- bzw. Entschlüsselung. Dennoch sollten Schlüssellängen ≤ 1024 Bit nicht mehr verwendet werden.

Beispiel:

```
tMemPrivate->MemGenKeyPair(tMemPublic, 2048);
```

Mögliche Laufzeitfehler:

<u>ErrHdlInvalid</u>	Einer der übergeben Deskriptoren (obj) oder (handle1) ist ungültig.
<u>ErrMemExhausted</u>	Nicht genug Speicher vorhanden.
<u>ErrValueInvalid</u>	Es wurde eine ungültige Schlüssellänge (int2) angegeben.

obj -> MemHash(int1[, int2[,
int3]]) : alpha



Authentifizierungs-Code ermitteln

obj Deskriptor des Memory-Objekts
 Hash-Verfahren und Kodierungen
 MemHashMD5 MD5-Hash
 (Message-Digest
 Algorithm 5)
 MemHashRMD160 RIPEMD
 160-Hash (RACE
 Integrity
 Primitives
 Evaluation
 Message Digest)
 int1 MemHashSHA1 SHA-1 Hash
 (Secure Hash
 Algorithm)
 MemHashSHA256 SHA-256 Hash
 MemHashSHA384 SHA-384 Hash
 MemHashSHA512 SHA-512 Hash
 MemResultHex Hexadezimaless
 Ergebnis
 MemResultBase64 Base64-kodiertes
 Ergebnis
 int2 Startposition für die Berechnung
 (optional)
 int3 Länge (optional)
 Resultat alpha Hash-Wert
 Siehe Verwandte Befehle, MemHMAC()
 Dieser Befehl bildet für eine Nachricht einen Authentifizierungs-Code. Der Code wird
 aus dem Inhalt des übergebenen Memory-Objekts (obj) und dem Verfahren (int1)
 errechnet. Im Parameter (int1) wird auch die Kodierung des Ergebnisses angegeben.
 Folgende Konstanten können übergeben werden:

- **Hash-Verfahren**

MemHashMD5 MD5-Hash (Message-Digest Algorithm 5)
MemHashRMD160 RIPEMD 160-Hash (RACE Integrity Primitives Evaluation
 Message Digest)
MemHashSHA1 SHA-1 Hash (Secure Hash Algorithm)
MemHashSHA256 SHA-256 Hash
MemHashSHA384 SHA-386 Hash
MemHashSHA512 SHA-512 Hash

- **Kodierung des Ergebnisses**

MemResultHex Hexadezimaless Ergebnis
MemResultBase64 Base64-kodiertes Ergebnis



Der Parameter (int1) setzt sich aus der Kombination je einer Konstanten aus den Bereichen Verfahren und Ergebnis-Kodierung zusammen.


Beispiel:

```
tHashValue # tMem->MemHash(_MemHashSHA256 | _MemResultHex);
```

Der Hash-Wert kann auch aus einem Teil des Memory-Objekts gebildet werden. Dazu wird die Startposition (int2) und die zu berücksichtigende Länge (int3) angegeben. Wird eine ungültigen Startposition (int2) oder eine zu große Länge (int3) angegeben, wird ein Laufzeitfehler erzeugt.

Mögliche Laufzeitfehler:

- _ErrHdlInvalid** Der übergeben Deskriptor (obj) ist ungültig.
- _ErrValueInvalid** Als Hash- und Kodierungsverfahren (int1) wurde eine ungültige / unvollständige Kombination angegeben. Aus jeder Gruppe muss eine Konstante angegeben werden.
- _ErrValueRange** Die angegeben Startposition (int2) oder die Länge des Bereiches (int3) ist zu groß.

obj -> MemHMAC(int1, alpha2[, 
 int3[, int4[, handle5]]]) : alpha
 Authentifizierungs-Code ermitteln

obj Deskriptor des Memory-Objekts
 Hash-Verfahren und Kodierungen

<u>MemHashMD5</u>	MD5-Hash (Message-Digest Algorithm 5)
<u>MemHashRMD160</u>	RIPEMD 160-Hash (RACE Integrity Primitives Evaluation Message Digest)
<u>MemHashSHA1</u>	SHA-1 Hash (Secure Hash Algorithm)
<u>MemHashSHA256</u>	SHA-256 Hash
<u>MemHashSHA384</u>	SHA-384 Hash
int1 <u>MemHashSHA512</u>	SHA-512 Hash
<u>MemKeyHex</u>	Schlüssel liegt hexadezimal vor
<u>MemKeyBase64</u>	Schlüssel liegt Base64-kodiert vor
<u>MemKeyMem</u>	Schlüssel liegt als <u>Memory</u> -Objekt (handle5) vor
<u>MemResultHex</u>	Hexadezimales Ergebnis
<u>MemResultBase64</u>	Base64-kodiertes Ergebnis

alpha2 Schlüssel des Hash-Verfahrens
 int3 Startposition für die Berechnung
 (optional)
 int4 Länge (optional)
 handle5 Memory-Objekt mit Schlüssel des
 Hash-Verfahrens (optional)

Resultat alpha Authentifizierungscode

Siehe Verwandte Befehle, MemHash()

Dieser Befehl bildet für eine Nachricht einen Authentifizierungs-Code. Der Code wird aus dem Inhalt des übergebenen Memory-Objekts (obj), dem Verfahren (int1) und dem Schlüssel (alpha2) errechnet. Der Schlüssel kann alternativ in einem Memory-Objekt vorliegen, welches in (alpha5) angegeben wird. Dazu muss in (int1) MemKeyMem

angegeben sein. Es werden maximal die ersten 1024 Byte des Schlüssels aus dem Memory-Objekt berücksichtigt.

Im Parameter (int1) wird auch die Kodierung des Ergebnisses angegeben. Folgende Konstanten können übergeben werden:

- **Hash-Verfahren**

- MemHashMD5 MD5-Hash (Message-Digest Algorithm 5)
- MemHashRMD160 RIPEMD 160-Hash (RACE Integrity Primitives Evaluation Message Digest)
- MemHashSHA1 SHA-1 Hash (Secure Hash Algorithm)
- MemHashSHA256 SHA-256 Hash
- MemHashSHA384 SHA-384 Hash
- MemHashSHA512 SHA-512 Hash

- **Kodierung des Schlüssels**

- MemKeyHex Schlüssel liegt hexadezimal vor
- MemKeyBase64 Schlüssel liegt Base64-kodiert vor
- MemKeyMem Schlüssel liegt Memory-Objekt vor

- **Kodierung des Ergebnisses**

- MemResultHex Hexadezimaales Ergebnis
- MemResultBase64 Base64-kodiertes Ergebnis



Der Parameter (int1) setzt sich aus der Kombination je einer Konstanten aus den Bereichen Verfahren, Schlüssel-Kodierung und Ergebnis-Kodierung zusammen.

Beispiele:

```
tHashValue # tMem->MemHMAC(_MemHashSHA256 | _MemKeyBase64 | _MemResultHex, aKeyHMAC);tHashValue #
```

Der Hash-Wert kann auch aus einem Teil des Memory-Objekts gebildet werden. Dazu wird die Startposition (int3) und die zu berücksichtigende Länge (int4) angegeben. Wird eine ungültigen Startposition (int3) oder eine zu große Länge (int4) angegeben, wird ein Laufzeitfehler erzeugt.

Mögliche Laufzeitfehler:

- ErrHdlInvalid Der übergeben Deskriptor (obj) oder (handle5) ist ungültig.
- ErrValueInvalid Als Hash- und Kodierungsverfahren (int1) wurde eine ungültige / unvollständige Kombination angegeben. Aus jeder Gruppe muss eine Konstante angegeben werden.
- ErrValueRange Die angegeben Startposition (int3) oder die Länge des Bereiches (int4) ist zu groß.

obj ->

MemReadByte()

: int

Bytewert lesen

obj Deskriptor des
 Memory-Objekts

int1 Position des zu
 lesenden Bytes

Resultat int gelesenes Byte 

Verwandte Befehle,

Siehe MemWriteByte(),

MemReadStr()


Der Befehl ermittelt den Bytewert im Speicherbereich des Memory-Objekts (obj) an der Position (int1).

Mögliche Laufzeitfehler:

__ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

__ErrValueRange Die in (int1) angegebene Position ist nicht gültig.

obj ->

MemReadStr(int1, )

int2[, int3]) : alpha


Zeichenkette lesen

obj Deskriptor des
 Memory-Objekts

int1 Erstes zu lesendes Byte


int2 Anzahl, der zu lesenden
 Bytes


int3 Zielzeichensatz
 (optional)

Resultat alpha gelesene
 Zeichenkette 

Siehe Verwandte Befehle,
 MemWriteStr(),
 MemReadByte()

Dieser Befehl liest aus dem Speicherbereich des Memory-Objekts (obj) ab der Position (int1) eine Menge von (int2) Bytes und überträgt sie in den Resultatwert. Dabei wird eine Zeichensatzwandlung durchgeführt, wenn in (int3) ein Zielzeichensatz (siehe Charset) angegeben wird. Als Quellzeichensatz wird die Eigenschaft Charset des Memory-Objekts verwendet. Es kann zwischen allen unterstützten Zeichensätzen konvertiert werden. Dabei werden Zeichen, die nicht im Zielzeichensatz darstellbar sind, durch ein Fragezeichen ersetzt.

 In einigen Zeichensätze werden mehrere Bytes zur Repräsentation eines Zeichens verwendet. Wird eine Anzahl von Bytes angegeben, die nicht auf eine Zeichengrenze fällt, kann das letzte Zeichen nicht konvertiert werden.

 Da Zeichenketten null-terminiert sind, kann die Länge der resultierenden Zeichenkette kleiner als die zu lesende Länge (int2) sein, sofern das Memory-Objekt im ausgelesenen Bereich ein Null-Byte enthält.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) übergebene Desriptor ist ungültig.

_ErrValueRange Die Position (int1) oder die Länge (int2) liegen außerhalb des zulässigen Bereichs.

_ErrValueInvalid In (int3) wurde ein undefinierter Zeichensatz angegeben.




obj -> MemResize(int1) : int

Größe des Memory-Objekts verändern

obj Deskriptor des
 Memory-Objekts

int1 Neue Größe des
 Objekts

Resultat int Fehlerwert 

Siehe Verwandte
 Befehle, Size

Mit dieser Funktion wird die Größe des Speicherbereichs des Memory-Objekts (obj) verändert. In (int1) wird die neue Größe in Bytes angegeben. Die neue Größe muss dabei im Bereich von 1 Byte bis 512 MB (bei 32-Bit-Prozessen) bzw. 2 GB (bei 64-Bit-Prozessen) liegen. Bei einer Verkleinerung werden eventuell überstehende Daten verworfen.

Konnte das Objekt verändert werden, wird _ErrOk zurückgegeben. Steht keine ausreichende Menge an Hauptspeicher zur Verfügung, ist das Resultat _ErrOutOfMemory.

Folgende Fehlerwerte sind möglich:

_ErrValueInvalid Als Größe (int1) wurde ein Wert ≤ 0 angegeben.

_ErrLimitExceeded Bei einem 32-Bit-Prozess wurde eine Größe (int1) von mehr als 512 MB angegeben.

_ErrOutOfMemory Der Speicher konnte nicht angefordert werden.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) übergebene Desriptor ist ungültig.

obj -> MemSign(int1,
alpha2, var alpha3[,
int4[, int5]]) : int
Signatur erstellen



obj Nachricht (Memory-Objekt)

Optionen

<u>MemSignRSA</u>	RSA-Signatur erstellen
<u>MemSignDSA</u>	DSA-Signatur erstellen
<u>MemHashMD5</u>	MD5-Hash verwenden
<u>MemHashRMD160</u>	RIPEMD-160-Hash verwenden
<u>MemHashSHA1</u>	SHA-1-Hash verwenden
<u>MemHashSHA256</u>	SHA-256-Hash verwenden
<u>MemHashSHA384</u>	SHA-384-Hash verwenden
<u>MemHashSHA512</u>	SHA-512-Hash verwenden
<u>MemKeyHex</u>	Schlüssel liegt als hexadezimal kodierte Zeichenkette vor
<u>MemKeyBase64</u>	Schlüssel liegt Base64-kodierte Zeichenkette vor
<u>MemResultHex</u>	Signatur soll Hex-Codiert sein
<u>MemResultBase64</u>	Signatur soll Base64-Codiert sein

alpha2 Schlüssel

var
alpha3 Signatur

int4 Nachrichtenposition (optional)

int5 Nachrichtenlänge (optional)

Resultat int Fehlerwert

<u>ErrOk</u>	Erfolg
<u>ErrMemKeyInvalid</u>	Schlüssel ungültig

ErrMemKeyLength Schlüssel
zu kurz für
Nachricht

Siehe [Verwandte Befehle](#), [MemVerify\(\)](#),
[Blog](#)

Dieser Befehl signiert eine Nachricht nach dem in (int1) angegebenen Verfahren. Die Signatur wird aus dem Inhalt des übergebenen Memory-Objektes (obj), dem Verfahren (int1) und dem privaten Schlüssel (alpha2) gebildet. Im Parameter (int1) wird ebenfalls der zu verwendete Hash-Algorithmus, die Codierung des privaten Schlüssels und die Codierung der Signatur angegeben.

Folgende Optionen (int1) sind möglich:

- **Typ der Signatur**

- MemSignRSA RSA-Signatur erstellen

- MemSignDSA DSA-Signatur erstellen

- **Hash-Verfahren**

- MemHashMD5 MD5-Hash verwenden

- MemHashRMD160 RIPEMD-160-Hash verwenden

- MemHashSHA1 SHA-1-Hash verwenden

- MemHashSHA256 SHA-256-Hash verwenden

- MemHashSHA384 SHA-384-Hash verwenden

- MemHashSHA512 SHA-512-Hash verwenden

- **Kodierung des Schlüssels**

- MemKeyHex Schlüssel liegt als hexadezimal kodierte Zeichenkette vor

- MemKeyBase64 Schlüssel liegt Base64-kodierte Zeichenkette vor

- **Kodierung des Ergebnisses**

- MemResultHex Signatur soll Hex-Codiert sein

- MemResultBase64 Signatur soll Base64-Codiert sein

Der Paramter (int1) setzt sich aus der Kombination je einer Konstanten der Bereiche zusammen.

Die ermittelte Signatur wird in (alpha3) zurückgegeben.

Der private Schlüssel (alpha2) muss im PKCS #1- (RSA oder DSA) oder im PKCS #8-Format (RSA oder DSA) vorliegen. Die Begin- und Endmarkierung dürfen **nicht** enthalten sein.

Ein RSA-Schlüsselpaar mit 2048 Bit Länge kann beispielsweise wie folgt angelegt werden:

```
tMemPrivate->MemGenKeyPair(tMemPublic, 2048);
```

Alternativ ist dies mit OpenSSL wie folgt möglich:

```
openssl genrsa -out rsa_priv_key.pem 2048openssl rsa -in rsa_priv_key.pem -pubout -out rsa_pub_ke
```

Kontakt

Mit der folgenden Funktion kann der Schlüssel eingelesen werden:

```
sub ReadKey( aKeyFile      : alpha(250); var aKey      : alpha; opt aPublic : logic;): int; local
```

Optional kann mit den Argumenten (int4) die Position und (int5) die Länge der zu verwendenden Nachricht angegeben werden.

Mögliche Laufzeitfehler:

<u>_ErrHdlInvalid</u>	Der übergeben Deskriptor (obj) ist ungültig.
<u>_ErrMemExhausted</u>	Nicht genug Speicher für kodierte Signatur (vor der Rückgabe) vorhanden.
<u>_ErrStringOverflow</u>	Der Schlüssel (alpha2) ist länger als 8192 Zeichen oder die in (alpha3) übergebene Variable ist nicht lang genug zum Speichern der Signatur.
<u>_ErrValueInvalid</u>	In den Optionen (int1) wurde eine ungültige oder unvollständige Kombination angegeben.
<u>_ErrValueRange</u>	Die angegeben Startposition (int4) oder die Länge des Bereiches (int5) ist zu groß.

obj -> MemUncompress([int1[,
int2[, handle3[, int4]]]]) : int
Speicherbereich dekomprimieren



obj Quelle / Ziel
 (Memory-Objekt)
int1 Quellposition
 (optional)
int2 Quelllänge (optional)
handle3 Ziel (Memory-Objekt,
 optional)
int4 Zielposition (optional)
Resultat int Fehlerwert



Verwandte Befehle,
Siehe MemCompress(),
 FsiFileUncompress()

Dieser Befehl dekomprimiert den Inhalt des Memory-Objektes (obj).

Im Parameter (int1) kann die Quellposition angegeben werden. Ist dieser Wert nicht angegeben oder 0, werden die Daten ab Beginn des Memory-Objektes dekomprimiert.

Der Parameter (int2) gibt die zu dekomprimierende Länge an. Ist dieser Wert nicht angegeben, 0 oder MemDataLen wird der restliche Inhalt (nach der Quellposition) des Memory-Objektes dekomprimiert.

Optional kann im Parameter (handle3) ein Ziel-Memory-Objekt angegeben werden. Ist dieses nicht angegeben oder ist es identisch mit dem Quellobjekt (obj), wird in das Quellobjekt geschrieben. Hierbei werden alle Daten verworfen, die hinter dem letzten dekomprimierten Zeichen vorhanden waren.

Zusätzlich kann eine Zielposition (int4) angegeben werden, wenn nicht an den Anfang des Ziel-Objektes geschrieben werden soll. Alle vorhandenen Daten ab der Position werden überschrieben.

Beispiele

```
// Inhalt des Memory-Objektes tMemSrc in neues Memory-Objekt dekomprimieren tMemSrc->MemUncompress
```

Fehlerwerte

Folgende Fehlerwerte können von der Funktion zurückgegeben werden:

ErrOk Kein Fehler aufgetreten.
ErrData Komprimierte Daten sind inkonsistent oder Quellobjekt (obj) ist leer.
ErrGeneric Interner Fehler aufgetreten.
Mögliche Laufzeitfehler:

ErrHdlInvalid Einer der übergeben Deskriptoren (obj) oder (handle3) ist ungültig.
ErrMemExhausted Nicht genug Speicher vorhanden.

ErrValueRange Eine der Längen- oder Positionsangaben (int1), (int2) oder (int4) ist ungültig.

obj -> MemVerify(int1,
alpha2, alpha3[, int4[,
int5]]) : int



Signatur überprüfen

obj Nachricht (Memory-Objekt)

Optionen

MemSignRSA

RSA-Signatur
verifizieren

MemSignDSA

DSA-Signatur
verifizieren

MemHashMD5

MD5-Hash
verwenden

MemHashRMD160

RIPEMD-160-Hash
verwenden

MemHashSHA1

SHA-1-Hash
verwenden

MemHashSHA256

SHA-256-Hash
verwenden

MemHashSHA384

SHA-384-Hash
verwenden

int1

MemHashSHA512

SHA-512-Hash
verwenden

MemKeyHex

Schlüssel liegt als
hexadezimal
kodierte
Zeichenkette vor

MemKeyBase64

Schlüssel liegt
Base64-kodierte
Zeichenkette vor

MemSignatureHex

Signatur liegt
Hex-Codiert vor

MemSignatureBase64

Signatur liegt
Base64-Codiert
vor

alpha2 Schlüssel

alpha3 Signatur

int4 Nachrichtenposition (optional)

int5 Nachrichtenlänge (optional)

Resultat int Fehlerwert

ErrOk

Signatur
passt zur
Nachricht

ErrMemKeyInvalid Schlüssel

ungültig
_ErrMemSgnInvalid Signatur
ungültig
_ErrMemMsgVerify Signatur
passt nicht
zur
Nachricht

Siehe Verwandte Befehle, MemSign(),
Blog

Dieser Befehl prüft, ob die in (alpha3) angegebene Signatur zum öffentlichen Schlüssel (alpha2) und der im Memory-Objekt (obj) liegenden Nachricht passt. Im Parameter (int1) wird neben dem, verwendeten Verfahren, der Hash-Algorithmus sowie die Codierung des öffentlichen Schlüssels und die der Signatur angegeben.

Folgende Optionen (int1) sind möglich:

- **Typ der Signatur**

- _MemSignRSA RSA-Signatur verifizieren

- _MemSignDSA DSA-Signatur verifizieren

- **Hash-Verfahren**

- _MemHashMD5 MD5-Hash verwenden

- _MemHashRMD160 RIPEMD-160-Hash verwenden

- _MemHashSHA1 SHA-1-Hash verwenden

- _MemHashSHA256 SHA-256-Hash verwenden

- _MemHashSHA384 SHA-384-Hash verwenden

- _MemHashSHA512 SHA-512-Hash verwenden

- **Kodierung des Schlüssels**

- _MemKeyHex Schlüssel liegt als hexadezimal kodierte Zeichenkette vor

- _MemKeyBase64 Schlüssel liegt Base64-kodierte Zeichenkette vor

- **Kodierung der Signatur**

- _MemSignatureHex Signatur liegt Hex-Codiert vor

- _MemSignatureBase64 Signatur liegt Base64-Codiert vor

Der Parameter (int1) setzt sich aus der Kombination je einer Konstanten der Bereiche zusammen.


Der öffentliche Schlüssel (alpha2) muss im PKCS #1- (nur RSA) oder im X.509-Format (RSA oder DSA) vorliegen. Die Begin- und Endmarkierung dürfen **nicht** enthalten sein.

Optional kann mit den Argumenten (int4) die Position und (int5) die Länge der zu verwendenden Nachricht angegeben werden.

Über den Rückgabewert kann ermittelt werden, ob die Signatur zum öffentlichen Schlüssel (alpha2) und der Nachricht (obj) passt (_ErrOk) oder nicht (_ErrMemMsgVerify).

Mögliche Laufzeitfehler:

<u>ErrHdlInvalid</u>	Der übergeben Deskriptor (obj) ist ungültig.
<u>ErrStringOverflow</u>	Der Schlüssel (alpha2) ist länger als 8192 Zeichen.
<u>ErrValueInvalid</u>	In den Optionen (int1) wurde eine ungültige oder unvollständige Kombination angegeben.
<u>ErrValueRange</u>	Die angegeben Startposition (int4) oder die Länge des Bereiches (int5) ist zu groß.

obj ->
 MemWriteByte(int1 
 int2[, int3])

Bytewert schreiben

obj Deskriptor des
 Memory-Objekts
 int1 zu beschreibende
 Position

int2 zu schreibender
 Wert
 Anzahl der Bytes,
 int3 die geschrieben
 werden sollen
 (optional)

Verwandte
Befehle,
 Siehe MemReadByte(),
 MemWriteStr()

Mit diesem Befehl wird der Bytewert (int2) in den Speicherbereich des
Memory-Objekts (obj) an die Position (int1) geschrieben. Durch die Angabe einer
 Länge (int3) kann ein kompletter Speicherabschnitt mit diesem Wert gefüllt werden.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

_ErrValueRange Die Position (int1) oder die Länge (int3) liegen außerhalb des
 zulässigen Bereichs.

obj ->

MemWriteStr(int1,
alpha2[, int3])



Zeichenkette schreiben

obj Deskriptor des
 Memory-Objekts

 Position des
int1 ersten zu
 schreibenden
 Bytes

alpha2 zu schreibende
 Zeichenkette

int3 Zeichensatz der
 Zeichenkette
 (optional)

Verwandte

Siehe Befehle,

MemReadStr(),

MemWriteByte()


Dieser Befehl schreibt die Daten des alphanumerischen Werts (alpha2) in den Speicherbereich des Memory-Objekts (obj) ab der Position (int1). Dabei wird eine Zeichensatzwandlung durchgeführt, wenn in (int3) ein Quellzeichensatz (siehe Charset) angegeben wird. Als Zielzeichensatz wird die Eigenschaft Charset des Memory-Objekts verwendet. Es kann zwischen allen unterstützten Zeichensätzen konvertiert werden. Dabei werden Zeichen, die nicht im Zielzeichensatz darstellbar sind, durch ein Fragezeichen ersetzt.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) übergebene Desriptor ist ungültig.

_ErrValueRange Die Position (int1) liegt außerhalb des zulässigen Bereichs.

_ErrValueInvalid In (int3) wurde ein undefinierter Zeichensatz angegeben.

obj -> MsxReadMem(handle1, int2, int3) : 

int


Nachrichtenkanal in Memory-Objekt lesen

obj Deskriptor eines
Nachrichtenkanals

handle1 Deskriptor eines
Memory-Objekts

int2 Zielposition im
Memory-Objekt

int3 Anzahl der zu
lesenden Bytes

Resultat int Fehlerwert 

Siehe Verwandte Befehle,
MsxWriteMem()

Dieser Befehl liest binäre Daten aus dem Nachrichtenkanal (obj) in das Memory-Objekt (handle1) ein. Die Funktion entspricht damit MsxRead(_MsxData, ...). Vor dem Aufruf von MsxReadMem() muss ein Nachrichtenelement bereits mit MsxRead(_MsxItem, ...) geöffnet sein. In (int2) wird die Zielposition im Memory-Objekt und in (int3) die Datenlänge angegeben. Die Datenlänge muss identisch mit der beim Schreiben angegebenen Länge sein (siehe MsxWriteMem()).

Das Resultat enthält den Fehlerwert oder __ErrOk, wenn kein Fehler aufgetreten ist.

Mögliche Laufzeitfehler:

__ErrHdlInvalid Der in (obj) oder (handle1) angegebene Deskriptor ist ungültig.

__ErrValueRange Der in (int2) oder (int3) übergebene Wert ist außerhalb des zulässigen Bereichs.



obj -> MsxWriteMem(handle1, int2, int3) : int

Nachrichtenkanal schreiben aus einem Memory-Objekt

obj Deskriptor eines Nachrichtenkanals

handle1 Deskriptor eines Memory-Objekts

int2 Startposition im Memory-Objekt

int3 Anzahl der zu schreibenden Bytes

Resultat int Anzahl der geschriebenen Bytes oder Fehlerwert

Siehe Verwandte Befehle, MsxReadMem()

Dieser Befehl schreibt binäre Daten aus dem Memory-Objekt (handle1) in den Nachrichtenkanal (obj). Die Funktion entspricht damit MsxWrite(_MsxData, ...). Vor dem Aufruf von MsxWriteMem() muss ein Nachrichtenelement bereits mit MsxWrite(_MsxItem, ...) geöffnet sein. In (int2) wird die Startposition im Memory-Objekt und in (int3) die Datenlänge angegeben. Die Funktion erzeugt im Nachrichten-Stream ein binäres Feld mit der Länge (int3). Zum Einlesen der Daten dieses binären Felds muss exakt die gleiche Länge bei MsxReadMem() angegeben werden.

Das Resultat enthält den Fehlerwert oder _ErrOk, wenn kein Fehler aufgetreten ist.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) oder (handle1) angegebene Deskriptor ist ungültig.

_ErrValueRange Der in (int2) oder (int3) angegebene wert ist außerhalb des zulässigen Bereichs.

obj -> PdfTextExtractMem(handle1) : int
Textinformationen einer PDF-Seite ermitteln



obj Deskriptor eines
 PDF-Objekts

handle1 Deskriptor eines
 Memory-Objekts

Resultat int Fehlerwert

Verwandte


Siehe Befehle,
 PdfPageOpen()

Mit dieser Anweisung wird die Textinformation einer zuvor mit PdfPageOpen() geöffneten Seite eines PDF-Dokuments ermittelt und in das Memory-Objekt (handle1) übertragen.

Ist keine Seite geöffnet, gibt die Anweisung den Fehlerwert _ErrPdfPageClosed zurück. Der ermittelte Text wird an das Memory-Objekt angehängt (siehe Len). Der Zeichensatz des Memory-Objekts wird entsprechend der Eigenschaft Charset berücksichtigt.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.

obj -> SckReadMem(int1, handle2, )
int3, int4) : int

Vom Socket in Memory-Objekt lesen

obj Socket-Deskriptor

Optionen

SckLine komplette Zeile lesen
int1 SckReadMax Lesen bis maximale Länge /
Lesen aller empfangener
Zeichen

handle2 Deskriptor des Memory-Objekts

int3 Startposition im Memory-Objekt

int4 Anzahl der zu lesenden Bytes

Resultat int Anzahl der gelesenen Bytes oder
Fehlerwert


Siehe Verwandte Befehle, SckWriteMem()

Mit dieser Funktion werden Daten vom Socket (obj) gelesen. Durch die Angabe einer Option in (int1) kann die Anzahl der Zeichen auf eine Zeile oder auf die Menge der verfügbaren Daten begrenzt werden. In (handle2) muss der Deskriptor eines Memory-Objekts angegeben werden. In (int3) wird die Startposition und (int4) die Anzahl der maximal zu lesenden Bytes übergeben. Gegebenenfalls wird der Wert der Eigenschaft Len erhöht.

Das Resultat gibt die Anzahl der gelesenen Bytes zurück. Ist das Resultat negativ ist ein Fehler aufgetreten und das Resultat enthält den Fehlerwert.

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) oder (handle2) übergebene Deskriptor ist ungültig.
ErrMemExhausted Der Speicher konnte nicht angefordert werden.
ErrValueRange Der in (int3) oder (int4) übergebene Wert ist außerhalb des zulässigen Bereichs.

obj -> SckWriteMem(int1, handle2, int3, int4) : 

Vom Memory-Objekt auf den Socket schreiben

obj Socket-Deskriptor

Optionen

int1 SckBuffered Daten puffern

SckLine nach den Daten automatisch ein
CR/LF senden

handle2 Deskriptor eines Memory-Objekts

int3 Startposition im Memory-Objekt

int4 Anzahl der zu schreibenden Bytes

Resultat int Anzahl geschriebener Bytes oder Fehlerwert

Siehe Verwandte Befehle, SckReadMem()

Mit dieser Funktion werden Daten auf den Socket (obj) geschrieben. Mit der Option SckLine in (int1) wird nach den Daten automatisch ein CR/LF gesendet. Mit der Option SckBuffered werden die Daten zunächst zwischengespeichert und entweder beim Erreichen der Puffergrenze (4 KB) oder durch den Aufruf von SckWrite() mit einer Länge von 0 geschrieben. Dadurch werden die Daten von mehreren SckWriteMem()-Aufrufen für das Versenden zusammengefasst.

In (handle2) muss der Deskriptor eines Memory-Objekts angegeben werden. In (int3) wird die Startposition und (int4) die Anzahl der zu schreibenden Bytes übergeben.

Das Resultat gibt die Anzahl der geschriebenen Bytes zurück. Ist das Resultat negativ, ist ein Fehler aufgetreten und das Resultat enthält den Fehlerwert.

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) oder (handle2) übergeben Deskriptor ist ungültig.

ErrValueRange Der in (int3) oder (int4) übergebene Wert ist außerhalb des
zulässigen Bereichs.

obj -> WinRtfPicInsertMem(handle1[,
int2[, int3]]) : int



Einfügen von Bildern in ein RtfEdit-Objekt

obj Deskriptor des RtfEdit-Objekts

handle1 Deskriptor des Memory-Objekts


int2 Seitennummer (optional)

Optionen (optional)

WinRtfPicModeSpeed Performantes
Einfügen

WinRtfPicModeQuality Qualitatives
Einfügen

int3 WinRtfPicModeAuto Modus
abhängig von
der Farbtiefe
und Größe
des Bildes
auswählen

Resultat int Einfügeresultat (siehe Text) 

Siehe Verwandte Befehle,
WinRtfPicInsertName()

Diese Funktion fügt ein Bild aus einem Memory-Objekt (handle1) an der aktuellen Cursorposition eines RtfEdit-Objekts (obj) ein. Ist eine Selektion vorhanden, wird diese durch das Bild ersetzt. Das Argument Seitennummer (int2) bestimmt bei einem Multipage-TIFF, die Seite, die das einzufügende Bild enthält. Die Seitenzählung beginnt mit 1. Wird (int2) nicht angegeben oder ist der Wert Null, dann wird immer die erste Seite gewählt. Bei Formaten außer TIFF wird das Argument ignoriert.

Optional können in (int3) folgende Optionen angegeben werden:

WinRtfPicModeSpeed Performantes Einfügen

WinRtfPicModeQuality Qualitatives Einfügen

WinRtfPicModeAuto Modus abhängig von der Farbtiefe und Größe des Bildes
auswählen

Wird keine der Optionen angegeben, wird automatisch WinRtfPicModeAuto verwendet.

Die anzeigbaren Formate sind GIF, TIFF, JPEG, PNG und BMP.



Transparente oder semitransparente Pixel bei PNG bzw. 32bpp BMP-Dateien werden zu weißer Farbe gemischt.

Wird das Bildformat nicht unterstützt, gibt die Funktion den Fehlerwert ErrGeneric zurück. Ist der Arbeitsspeicher nicht ausreichend, wird ErrOutOfMemory zurückgegeben.

Beispiel:

```
// Bild ladentMem # MemAllocate(_MemAutoSize);tFsi # FsiOpen(_Sys->spPathMyPictures + '\MyPicture
```

Mögliche Laufzeitfehler:

Kontakt

ErrHdlInvalid Bei (obj) handelt es sich nicht um ein RtfEdit-Objekt

Befehle für HTTP-Objekte

Liste der Befehle und Konstanten zur Bearbeitung von HTTP-Objekten

Befehlsgruppen,

Siehe Befehlsliste,

HTTP

Befehle

- HttpClose
- HttpGetData
- HttpOpen

Konstanten

- _HttpCloseConnection
- _HttpDiscard
- _HttpParamsToData
- _HttpParamsToURI
- _HttpProxyAuthTypeBasic
- _HttpProxyAuthTypeNTLM
- _HttpRecvRequest
- _HttpRecvResponse
- _HttpSendRequest
- _HttpSendResponse
- _HttpSkipChunked
- _HttpUseWebProxy
- _HttpUseWebProxyTLS

HttpOpen(int1,
handle2) : handle



HTTP-Objekt erzeugen

Objekt-Typ

int1	<u>_HttpRequest</u>	Anfrage senden
	<u>_HttpRecvRequest</u>	Anfrage empfangen
	<u>_HttpSendResponse</u>	Antwort senden
	<u>_HttpRecvResponse</u>	Antwort empfangen

handle2 Deskriptor einer
Socket-Verbindung

Resultat handle HTTP-Objekt oder
Fehlerwert

Siehe Verwandte Befehle, HTTP,
HttpClose(), Beispiel

Der Befehle erzeugt ein neues HTTP-Objekt vom Typ (int1). In (handle2) muss der Deskriptor der aktiven Socket-Verbindung (SckConnect()) angegeben werden. Das Resultat ist entweder ein Fehlerwert oder der Deskriptor des neuen HTTP-Objekts. Für den Objekttyp gibt es vier Varianten, für die auch unterschiedliche Aktionen durchgeführt werden:

- **_HttpRequest - Objekt zum Erstellen und Versenden einer Anfrage**

Das Objekt wird mit den Standardwerten für Method, URI und Protocol definiert. Die Listen für Header-Einträge und Parameter sind leer. Das Resultat ist ErrOk.

- **_HttpSendResponse - Objekt zum Erstellen und Versenden einer Antwort**

Das Objekt wird mit den Standardwerten für StatusCode und Protocol definiert. Die Listen für Header-Einträge und Parameter sind leer. Das Resultat ist ErrOk.

- **_HttpRecvRequest - Objekt zum Empfangen und Auswerten einer Anfrage**

Das Objekt wird angelegt und der komplette Request-Header über den Socket (handle2) in die Liste der Header-Einträge eingelesen. Falls die URI Parameter enthält, werden diese in die Parameterliste übertragen. Das Resultat ist ErrOk, wenn ein gültiger HTTP-Header eingelesen werden konnte. Falls kein gültiger Header ermittelt werden konnte, ist das Resultat ErrData. Als Resultat können auch Socket-Fehler zurückgeliefert werden, wenn beim Lesen der Header-Daten ein solcher Fehler auftritt.

- **_HttpRecvResponse - Objekt zum Empfangen und Auswerten einer Antwort**

Das Objekt wird angelegt und der komplette Response-Header über den Socket (handle2) in die Liste der Header-Einträge eingelesen. Das Resultat ist ErrOk, wenn ein gültiger HTTP-Header eingelesen werden konnte. Falls kein gültiger

Kontakt

Header ermittelt werden konnte, ist das Resultat ErrData. Als Resultat können auch Socket-Fehler zurückgeliefert werden, wenn beim Lesen der Header-Daten ein solcher Fehler auftritt.

Mögliche Laufzeitfehler:

ErrValueInvalid Der in (int1) übergebene Wert ist nicht gültig.

ErrHdlInvalid Der in (handle2) übergebene Deskriptor ist ungültig.

obj -> HttpGetData(handle1) : 
 int
 Daten des HTTP-Objekts lesen
 obj Deskriptor des
 HTTP-Objekts
 handle1 Deskriptor des
 Ziel-Objekts
 Resultat int Fehlerwert 
 Verwandte
 Siehe Befehle, HTTP,
 HttpClose(),
 Beispiel

Mit diesem Befehl werden die Daten des HTTP-Bodys eingelesen. Soll der Inhalt des HTTP-Bodys gesetzt werden, erfolgt das mit der Anweisung HttpClose(). In (obj) wird der Deskriptor des HTTP-Objekts übergeben. Das Objekt muss vom Typ HttpRecvRequest oder HttpRecvResponse sein. In (handle1) wird das Objekt angegeben, in das die Daten eingelesen werden sollen. Dafür können drei verschiedene Objekttypen verwendet werden:

- **Datei**

Der in (handle1) übergeben Deskriptor muss vom Typ HdlFile sein. Die Daten werden direkt in die angegebene Datei geschrieben.

- **Memory-Objekt**

Der in (handle1) übergeben Deskriptor muss vom Typ HdlMem sein. Die Daten werden in das angegebene Memory-Objekt geschrieben.

- **Cte-Liste**

Der in (handle1) übergeben Deskriptor muss vom Typ HdlCteList sein. In diesem Fall enthält der HTTP-Body Parameter, die in die angegebene Cte-Liste eingelesen werden.



Die Cte-Liste enthält nur dann Werte, wenn die Daten URL-kodiert (x-www-form-urlencoded) übermittelt werden. Ansonsten ist das Ergebnis ErrData.

Neben den Objekttypen müssen noch beim Einlesen von Daten drei Fälle unterschieden werden:

- Im einfachsten Fall hat der HTTP-Header einen Content-Length-Eintrag, der die Größe des HTTP-Bodys enthält. Dadurch ist die Datenmenge bereits vor dem Aufruf der Anweisung bekannt.
- Im zweiten Fall hat der HTTP-Header einen Transfer-Encoding: chunked-Eintrag, der die Übertragung der Daten in mehreren Blöcken anzeigt. Es werden dann soviele Blöcke gelesen, wie die Gegenstelle liefert. In einigen Fällen wird zwischen den Daten und der nächsten Blocklänge kein Zeilenumbruch gesendet. Dies wird ignoriert und der nächste Block gelesen.
- Im dritten Fall werden solange Daten vom Socket gelesen, bis ein Timeout eintritt. Hierbei sollte der Timeout des Sockets vor dem Lesen der Daten auf


Kontakt

einen niedrigen Wert (0,25 bis 3 Sekunden) gesetzt werden, da HttpGetData() auf jeden Fall bis zum Timeout wartet.

Als Rückgabewert wird ein Fehlerwert zurückgegeben. Bei ErrOk ist kein Fehler aufgetreten. Im Fehlerfall können Fehlerkonstanten aus den Bereichen der externen Dateien, Socketfehler oder Fehler bei der Verarbeitung von Memory-Objekten zurückgegeben werden.

Mögliche Laufzeitfehler:

<u>ErrHdlInvalid</u>	Der in (obj) oder in (handle1) übergebene Deskriptor ist ungültig
<u>ErrStringOverflow</u>	Das in (handle1) angegebene <u>Memory</u> -Objekt ist zu klein für die Datenmenge.

obj -> HttpClose(int1[, ,
handle2]) : int

HTTP-Objekt schließen

obj Deskriptor des HTTP-Objekts

Optionen:

	<u>_HttpDiscard</u>	Daten nicht versenden
	<u>_HttpCloseConnection</u>	Nach versenden Verbindung trennen
	<u>_HttpParamsToURI</u>	Parameter in URI schreiben
int1	<u>_HttpParamsToData</u>	Parameter in Body schreiben
	<u>_HttpSkipChunked</u>	Chunked-Daten überlesen
	<u>_HttpUseWebProxy</u>	Senden über einen HTTP-Proxy
	<u>_HttpUseWebProxyTLS</u>	Senden über einen HTTPS-Proxy

handle2 Deskriptor auf HTTP-Body

Resultat int Fehlerwert 

Siehe Verwandte Befehle, Http,
HttpOpen(), Beispiel

Dieser Befehl schließt das Objekt (obj) und leert die Header- und Parameter-Liste. Alle Objekte, die in den Listen enthalten sind, werden gelöscht.

Bei den Typen _HttpRecvRequest und _HttpRecvResponse werden eventuell noch ausstehende Daten des HTTP-Body eingelesen und verworfen. Dazu muss der Header über einen Content-Length-Eintrag verfügen.

Bei den Typen _HttpSendRequest und _HttpSendResponse wird die Anfrage bzw. die Antwort über den bei HttpOpen() angegebenen Socket versendet. Als HTTP-Body kann der Inhalt einer externen Datei, der Inhalt eines Memory-Objekts oder die Parameter-Liste gesendet werden. Der Deskriptor der externen Datei oder des Memory-Objekts wird in (handle2) übergeben. In (int1) können eine oder mehrere der folgenden Optionen angegeben werden:

- **_HttpDiscard**

Die Daten werden nicht versendet.

- **_HttpCloseConnection**

Im HTTP-Header wird der Eintrag Connection: close vorgenommen, der bei HTTP/1.1 dem Empfänger signalisiert, dass die Verbindung nach dem Erhalt der HTTP-Response geschlossen werden soll.

- **_HttpParamsToURI**

Der Inhalt der Parameter-Liste wird in die URI eingefügt. Diese Option ist nur beim Typ _HttpSendRequest möglich.

- **_HttpParamsToData**

Der Inhalt der Parameter-Liste wird als HTTP-Body aufbereitet. In (handle2) darf kein Deskriptor angegeben werden.

- **_HttpSkipChunked**

Auf dem Socket vorhandene und noch nicht gelesene Chunked-Daten werden überlesen. Die Option wird nur bei _HttpRecvRequest und _HttpRecvResponse ausgewertet.

- **_HttpUseWebProxy / _HttpUseWebProxyTLS**

Falls das Senden über einen HTTP-Proxy (_HttpUseWebProxy) bzw. HTTPS-Proxy (_HttpUseWebProxyTLS) erfolgt, muss eine dieser Option verwendet werden. Dadurch wird ein modifizierter HTTP-Header verwendet. Die Eigenschaft ProxyAuthorization ist ebenfalls nur bei diesen Optionen wirksam.

Der Rückgabewert ist der _ErrOk, wenn der Versand erfolgreich war. Zu den möglichen Fehlerwerten gehören Socket-Fehler und _ErrFsiReadFault, wenn die Datei in (handle2) nicht vollständig gelesen werden kann.

Bei _HttpSendRequest werden die folgenden Header-Einträge automatisch gesetzt, wenn sie nicht bereits per Prozedur definiert wurden:

- Host: (falls HostName definiert ist)
- Date:
- User-Agent:

Bei _HttpSendResponse werden die folgenden Header-Einträge automatisch gesetzt, wenn sie nicht bereits per Prozedur definiert wurden:

- Date:
- Server:

Der Header-Eintrag Content-Length: wird automatisch gesetzt, wenn ein HTTP-Body vorhanden ist. Der angegebene Deskriptor muss, sofern er nicht weiter benötigt wird, entfernt werden (siehe MemFree() bzw. FsiClose()).

Mögliche Laufzeitfehler:

<u>_ErrHdlInvalid</u>	Der in (obj) oder in (handle2) übergebene Deskriptor ist ungültig.
<u>_ErrStringOverflow</u>	Bei der Option <u>_HttpParamsToURI</u> wird die URI länger als 8192 Bytes.

Befehle für Job-Verarbeitung

Liste der Befehle und Konstanten zur Verarbeitung von Jobs

Siehe Befehlsgruppen,
Befehlsliste, Job

Befehle

- JobClose
- JobControl
- JobEvent
- JobOpen
- JobSleep
- JobStart

obj -> JobClose()



Kontrollobjekt entfernen

obj Deskriptor eines
 Kontroll-Objekts

Verwandte

Siehe Befehle,


JobOpen()

Die Anweisung kann innerhalb einer Ereignisfunktion des SOA-Service, sowie im Standard- oder Advanced-Client ausgeführt werden.

Die Funktion schließt das JobControl-Objekt mit dem Deskriptor (obj). Der Job kann in der selben Prozedur oder in einer anderen Ereignisprozedur des Tasks-Prozesses erneut mit JobOpen() geöffnet werden, sofern der Job noch aktiv ist oder die beim Starten des Jobs angegeben Zeitspanne nach Jobende noch nicht verstrichen ist.

Mögliche Laufzeitfehler:

ErrHdlInvalid Der übergebene Deskriptor ist ungültig.

obj -> JobControl(int1[, 
int2]) : int

Job kontrollieren

obj	Deskriptor des JobControl-Objekts	
	Durchzuführende Funktion	
	<u>_JobWakeup</u>	Suspendierten Job aktivieren
	<u>_JobStop</u>	Job beenden, Ende nicht abwarten
int1	<u>_JobTerminate</u>	Job beenden, Ende abwarten
	<u>_JobMsxTimeoutRead</u>	Timeout abfragen / setzen
	<u>_JobEventReceiver</u>	Frame-Deskriptor für Job-Events setzen bzw. entfernen

int2 neuer Wert des Timeouts /
Frame-Deskriptor

Resultat int Ergebnis der Abfrage (siehe
Text)

Siehe Verwandte Befehle, JobOpen(),
Inter-Thread-Kommunikation
(Blog)

Die Anweisung kann innerhalb einer Ereignisfunktion des SOA-Service, sowie im
Standard- oder Advanced-Client ausgeführt werden.

Mit dieser Funktion kann ein JobControl- oder Job-Objekt verschiedene Funktionen
durchführen. In (obj) wird der Deskriptor des Objektes (siehe JobOpen()) übergeben.
In (int1) steht der Typ der durchzuführenden Funktion.

Für JobControl-Objekte sind folgende Optionen verwendbar:

- **_JobWakeup**

Mit dieser Option wird der Job aktiviert, wenn er sich in der eigenen
Ereignisfunktion mit der Anweisung JobSleep() suspendiert hat. Der
Rückgabewert ist _ErrTerminated, wenn der Job bereits beendet ist. Ansonsten
ist das Resultat _ErrOk.

- **_JobStop**

Diese Funktion setzt die Eigenschaft StopRequest für den Job ohne auf das
Ende des Jobs zu warten. Der Rückgabewert ist immer _ErrOk.

- **_JobTerminate**

Diese Funktion setzt die Eigenschaft StopRequest für den Job und wartet
darauf, dass sich der Job beendet. Der Rückgabewert ist immer _ErrOk.

- **JobMsxTimeoutRead**

Mit dieser Option kann der Timeout für MsxRead() auf die Message-Pipeline abgefragt (zwei Argumente) oder gesetzt werden (drei Argumente, der neue Wert steht in (int2)). Das Resultat ist der aktuelle bzw. neue Wert des Timeouts in Millisekunden (siehe Verarbeitungshinweise zum SOA-Service).

Für Job-Objekte sind folgende Optionen verwendbar:

- **JobMsxTimeoutRead**

Mit dieser Option kann der Timeout für MsxRead() auf die Message-Pipeline abgefragt (zwei Argumente) oder gesetzt werden (drei Argumente, der neue Wert steht in (int2)). Das Resultat ist der aktuelle bzw. neue Wert des Timeouts in Millisekunden (siehe Verarbeitungshinweise zum SOA-Service).


- **JobEventReceiver**

Mit dieser Option kann ein Frame-Deskriptor gesetzt werden (drei Argumente, der Deskriptor steht in (int2)), der durch JobEvent() ausgelöste EvtJob-Ereignisse empfängt. Bei Übergabe von nur zwei Argumenten wird ein zuvor gesetzter Deskriptor wieder entfernt.

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.

ErrValueInvalid Der in (int1) oder (int2) übergebene Wert ist nicht gültig.

obj -> 
 JobEvent([int1])
 Job-Ereignis auslösen
 obj Job-Objekt oder
 JobControl-Objekt
 Minimale
 int1 Zeitspanne seit
 letztem Aufruf in
 Millisekunden
 Resultat int Fehlerwert
 Siehe Verwandte Befehle,
 EvtJob

Mit dieser Anweisung wird ein Ereignis in die Ereigniswarteschlange des Betriebssystems eingefügt. Das Ereignis wird im Client aufgerufen, wenn die Ereignisse, die vor diesem eingefügt wurden, abgearbeitet sind. Zu welchem Zeitpunkt das Ereignis ausgelöst wird, kann nicht genau bestimmt werden.

Es sind zwei verschiedene Szenarien möglich:

1. Der Aufruf von JobEvent() erfolgt in einem Job, das Ereignis wird dann in dem Dialog ausgelöst, welcher auf Seite des Jobstarters bei JobOpen() angegeben wurde. In (obj) wird das Job-Objekt des Jobs angegeben.
2. Der Aufruf von JobEvent() erfolgt auf der Seite des Jobstarters, das Ereignis wird dann in dem Dialog ausgelöst, welcher auf Seite des Jobs mittels JobControl() und der Option JobEventReceiver angegeben wurde. In (obj) wird das JobControl-Objekt des Jobstarters angegeben.

Im Parameter (int1) kann optional die minimale Zeitspanne in Millisekunden angegeben werden, die seit dem letzten Aufruf von JobEvent() vergangen sein muss. Ist diese noch nicht vergangen, wird der Fehlerwert ErrLocked zurückgegeben.

Das Resultat ErrUnavailable entsteht, wenn es keinen Empfänger-Dialog für das Ereignis gibt. Dies ist der Fall, wenn bei JobOpen() beziehungsweise bei JobControl() - mit der Option JobEventReceiver - kein Dialog angegeben wurde. Sofern bereits ein JobEvent an den Dialog gesendet und noch nicht verarbeitet wurde, gibt die Funktion das Ergebnis ErrInProgress zurück.

Beispiel:

```
// Start des Tasks im ClienttJobID # JobStart(_JobThread, 10, 'JobTest:Work');if (tJobID > 0){ t
```

Mögliche Laufzeitfehler:

ErrHdlInvalid In (obj) wurde kein gültiger Deskriptor eines Job- oder JobControl-Objektes übergeben.

JobOpen(int1[,
handle2]) : handle
Kontrollobjekt erzeugen
int1 Id eines Job-Objekts
handle2 Deskriptor eines Dialogs
Deskriptor des
Resultat handle JobControl-Objekts
oder Fehlerwert
Verwandte Befehle,
Siehe JobControl, JobClose(),
Inter-Thread-Kommunikation
(Blog)


Die Anweisung kann innerhalb einer Ereignisfunktion des SOA-Service, sowie im Standard- oder Advanced-Client ausgeführt werden.

Mit diesem Befehl wird für den Job mit der Id (int1) ein Kontrollobjekt angelegt. Pro Job kann nur ein Kontrollobjekt vorhanden sein. Das Resultat ist ErrInUse, wenn bereits ein Kontrollobjekt für diesen Job besteht. Existiert kein Job mit der angegebenen Id, wird ErrUnknown zurückgeliefert. Das Kontrollobjekt sollte vor Ende der Prozedur mit JobClose() wieder geschlossen werden.

Im Parameter (handle2) kann optional der Deskriptor eines Dialoges angegeben werden. Ereignisse, die mit JobEvent() von dem Job ausgeführt werden, werden an das Ereignis EvtJob von dem angegebenen Dialog weitergeleitet.



Der Parameter (handle2) kann nur beim Aufruf aus dem Standard- oder Advanced-Client angegeben werden.

JobSleep(int1) : 

logic

Job suspendieren

int1 Wartezeit in
 Millisekunden

Resultat logic Wartezeit
 abgebrochen

Siehe Verwandte Befehle,
 JobControl()

Die Anweisung kann innerhalb eines Jobs ausgeführt werden. Jobs können durch eine Ereignisfunktion des SOA-Service, einen Standard- oder Advanced-Client gestartet werden.

Diese Funktion hält die Verarbeitung im Job für (int1) Millisekunden an. Im Unterschied zu SysSleep() kann der Wartezeit abgebrochen werden. In diesem Fall wird als Resultat true zurückgegeben. Ist die Wartezeit ohne Unterbrechung verstrichen, so ist das Resultat false. Ein Abbruch findet statt, wenn der komplette Task gestoppt werden soll oder wenn der Job durch JobControl(..., _JobWakeup) aufgeweckt wird.

Mögliche Laufzeitfehler:

_ErrValueInvalid In (int1) wurde ein Wert < 0 angegeben.

JobStart(int1, int2,
alpha3[, alpha4[,
alpha5[, handle6]]]) :



int

Job starten

Ausführungstyp:

JobThread Thread innerhalb des
Prozesses

JobProcess Betriebssystemprozess

JobCopyBuffers Inhalt der globalen
Feldpuffer kopieren (nur
vom Client aus)

int1

JobWinPrt Verwendung von
Oberflächenobjektbefehlen
ermöglichen (nur vom
Client aus)

JobStartNoRTE Laufzeitfehler bei JobStart
unterbinden

int2 Zeitspanne, in der die
Job-Informationen zur Verfügung
stehen

alpha3 Name der Ereignisfunktion

alpha4 Argumente für den Job (optional)

alpha5 Beschreibung des Jobs (optional)

handle6 Socket-Deskriptor (optional)

Job-Id oder Fehlerwert

Wert > 0 Job-Id des
neuen Jobs

Resultat int ErrTerminated Job
erfolgreich
durchgeführt

anderer Wert < Fehlerwert

0

Siehe Verwandte Befehle, JobOpen()

Die Anweisung kann innerhalb einer Ereignisfunktion des SOA-Service, sowie im
Standard- oder Advanced-Client ausgeführt werden.


Mit diesem Befehl wird ein neuer Job gestartet. In (int1) wird der Ausführungstyp des
Jobs angegeben: JobThread startet einen neuen Job-Thread innerhalb des aktuellen
Betriebssystemprozesses, JobProcess einen neuen Betriebssystemprozess für diesen
Job (siehe Verarbeitungshinweise).





Der Ausführungstyp JobProcess steht nur im SOA-Service zur Verfügung. Wird
sie im Standard- oder Advanced-Client angegeben, wird der Laufzeitfehler
ErrValueInvalid ausgelöst.

Der Ausführungstyp JobThread kann im Standard- oder Advanced-Client mit der
Optionen JobCopyBuffers und JobWinPrt kombiniert werden.

Ist die Option _JobCopyBuffers angegeben, werden beim Start des Jobs die aktuellen Feldpuffer des Clients in die Feldpuffer des Jobs übertragen.

-  Wurde zwischen Anmelden des Benutzers an der Datenbank und dem `JobStart(_JobThread | _JobCopyBuffers, ...)` die Datenstruktur neu aufgebaut, wird der Job nicht gestartet und die Eigenschaft _JobErrorCode auf den Fehler _ErrIllegalOp gesetzt.

Die Option _JobWinPrt ermöglicht die Verwendung von Dialogen und Oberflächenobjektbefehlen im Job.


-  Ein mit `JobStart()` gestarteter Thread ist von anderen Threads isoliert. Somit kann nur auf Eigenschaften und Ereignisse von Oberflächenobjekten des eigenen Threads zugegriffen werden. Zur Thread-Kommunikation können die Message-Exchange-Befehle verwendet werden. Zum Signalisieren von Nachrichten kann der Befehl JobEvent() verwendet werden.
-  Die Option _JobWinPrt sollte nur verwendet werden, wenn auch auf Oberflächenobjektbefehlen zugegriffen wird.

In (int2) wird die Zeitspanne in Sekunden angegeben, in der der Job noch nach seiner Beendigung mit JobOpen() geöffnet werden kann. Bei einem Wert von 0 kann der Job geöffnet werden, solange er ausgeführt wird. Der Maximalwert für (int2) beträgt 86400 Sekunden (24 Stunden).

In (alpha3) befindet sich der Name der Ereignisfunktion, die der Job aufrufen soll. In (alpha4) kann optional ein Argumenttext übergeben werden, der in der Jobprozedur über die Eigenschaft JobData abgerufen werden kann. In (alpha5) kann optional ein Beschreibungstext für den Job angegeben werden, der in der Jobprozedur über die Eigenschaft SvcDescription abgerufen werden kann. Die Argumente und die Beschreibung können jeweils bis zu 8192 Zeichen lang sein.

Optional kann in (handle6) ein Socket-Deskriptor (siehe SckConnect()) angegeben werden. Dabei wird eine Kopie des Deskriptors erstellt, der auf die gleiche Socket-Verbindung verweist. Diese ist in der Job-Funktion über die Eigenschaft JobSckHandle zugreifbar. Die Socket-Verbindung kann dann sowohl in dem Job, als auch in der aktuellen Prozedur verwendet werden. Somit kann beispielsweise im Job auf neue Nachrichten gewartet und gleichzeitig in der aktuellen Prozedur Nachrichten versendet werden. Hierbei sollte jedoch eine Seite nur lesen und eine nur schreiben.

Die Socket-Verbindung muss nur auf der Seite geschlossen werden, die JobStart() aufgerufen hat. Wird der Socket auf der Seite des Jobs geschlossen, wird die Verbindung auf beiden Seiten getrennt. Jedoch bleibt der Socket-Deskriptor in der Prozedur, die den Job gestartet hat, erhalten. Dieser muss dann separat mit SckClose() geschlossen werden. Am Ende des Jobs wird die erzeugte Kopie des Socket-Deskriptors entfernt. Die Socket-Verbindung bleibt am Ende des Jobs erhalten und kann in der aufrufenden Prozedur bis zum SckClose() verwendet werden.

-  Es kann kein Socket-Deskriptor angegeben werden, der mit einer der _SckTls...-Optionen erzeugt wurde.

Der Rückgabewert ist die Job-Id des neuen Jobs (Wert > 0). Ist der Job bereits durchgeführt, bevor `JobStart()` zurück gekommen ist, oder soll der Client, der Soa-Service oder der aktuelle Job beendet werden, wird _ErrTerminated

Kontakt

zurückgegeben. Falls beim Start des Jobs ein Fehler aufgetreten ist, enthält der Rückgabewert den Fehlerwert (Wert < 0). Die Job-Id ist innerhalb des laufenden Tasks eindeutig. Sie wird zum Erzeugen eines Kontroll-Objekts mit JobOpen() benötigt.



Ist die Option (int1) JobStartNoRTE angegeben, werden keine Laufzeitfehler ausgelöst. Statt dessen wird der Wert des Laufzeitfehlers als Fehlerwert zurückgeliefert.

Mögliche Laufzeitfehler:

ErrValueInvalid Der in (int1) übergebene Wert ist nicht gültig.

ErrNoProcInfo Prozedur ist nicht vorhanden oder wurde nicht übersetzt

ErrNoSub Prozedurfunktion ist nicht vorhanden

Befehle für PDF-Verarbeitung

Liste der Befehle und Konstanten zur Verarbeitung von PDF-Dateien

Befehlsgruppen,

Siehe Befehlsliste,

PDF

Befehle

- PdfAttachExportFile
- PdfAttachExportMem
- PdfAttachFile
- PdfAttachInfoGet
- PdfAttachMem
- PdfClose
- PdfInsertImage
- PdfInsertMeta
- PdfNew
- PdfOpen
- PdfPageClose
- PdfPageOpen
- PdfSavePage
- PdfTextColor
- PdfTextExtractMem
- PdfTextFont
- PdfTextWrite



obj -> PdfAttachExportFile(int1, alpha2) : int

Anhang eines PDF-Dokumentes in eine Datei exportieren

obj Deskriptor des
 PDF-Objekts

int1 Nummer des
 Anhangs

alpha2 Name der zu
 Zieldatei

Resultat int Fehlerwert



Verwandte Befehle,

PdfAttachInfoGet(),

Siehe PdfAttachExportMem(),

PdfAttachCount,

PdfAttachFile()

Der Befehl exportiert eine Anhangdatei eines PDF-Dokumentes (obj). Das PDF muss zuvor mit der Anweisung PdfOpen() geöffnet worden sein.

Im Argument (int1) muss die Nummer der Anhangdatei angegeben werden. Diese Nummer muss zwischen 1 und Der Anzahl Anhänge (siehe PdfAttachCount) liegen.

Die Zieldatei wird in (alpha2) angegeben. Diese Datei wird erzeugt bzw. überschrieben.

Fehlerwerte

Zusätzlich zu den Fehlerwerten für externe Dateioperationen (_ErrFsi...) kann von der Funktion _ErrGeneric zurückgegeben werden, wenn es einen Fehler beim Auslesen der Dateiinformationen gab.

Beispiel:

```
// Anhänge zählt AttachCount # tHdlPdf->spPdfAttachCount;// Anhänge durchgehenfor    tAttachNo #
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.

_ErrValueInvalid Ungültige Anhangsnummer (int1) angegeben.




obj -> PdfAttachExportMem(int1, handle2) : int
Anhang eines PDF-Dokumentes in eine Datei exportieren

obj Deskriptor des
 PDF-Objekts

int1 Nummer des
 Anhangs

handle2 Deskriptor eines
 Memory-Objektes

Resultat int Fehlerwert 

Verwandte Befehle,
PdfAttachInfoGet(),

Siehe PdfAttachExportFile(),
 PdfAttachCount,
 PdfAttachMem()

Der Befehl liest eine Anhangdatei eines PDF-Dokumentes (obj) in ein Memory-Objekt ein. Das PDF muss zuvor mit der Anweisung PdfOpen() geöffnet worden sein.

Im Argument (int1) muss die Nummer der Anhangdatei angegeben werden. Diese Nummer muss zwischen 1 und Der Anzahl Anhänge (siehe PdfAttachCount) liegen.

Mit dem Argument (handle2) wird das Memory-Objekt angegeben, in welches die Anhangdatei geschrieben wird. Das Memory-Objekt muss groß genug sein oder mit MemAutoSize angelegt werden. Der Inhalt des Memory-Objektes wird überschrieben.

Fehlerwerte

Von der Funktion kann ErrGeneric zurückgegeben werden, wenn es einen Fehler beim Auslesen der Dateiinformationen gab.

Beispiel:

```
// Anhänge zählt AttachCount # tHdlPdf->spPdfAttachCount;// Anhänge durchgehenfor    tAttachNo #
```

Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) bzw. (handle2) angegebene Deskriptor ist ungültig.

ErrValueInvalid Ungültige Anhangsnummer (int1) angegeben.

ErrMemExhausted Das Memory-Objekt (handle2) konnte nicht vergrößert werden.

obj -> PdfAttachFile(alpha1[,
int2[, alpha3[, logic4]]) : int
Anhangdatei zu PDF hinzufügen



obj Deskriptor des
 PDF-Objekts

 Pfad- und
alpha1 Dateiname der
 Anhangdatei

int2 Verknüpfungsart
 der Anhangdatei

alpha3 Beschreibung der
 Anhangdatei

logic4 Anhang
 komprimieren

Resultat int Fehlerwert

Verwandte Befehle,
Siehe PdfAttachMem(),
 PdfAttachExportFile(),
 Blog

Der Befehl fügt einem PDF eine extern vorliegende Anhangdatei hinzu. Das PDF muss zuvor mit der Anweisung PdfOpen() geöffnet worden sein.

Im Argument (alpha1) wird die einzufügende Anhangdatei angegeben.

Mit dem Argument (int2) kann die Verknüpfungsart definiert werden. Folgende Arten können angegeben werden:

- **PdfAttachAssociateNone**

Nur einfügen der Anhangdatei, keine weitere Verknüpfung.

- **PdfAttachAssociateAlternative**

Verknüpfung des Anhangs mit Beziehung "Alternative". Die Konstante sollte ab sofort anstelle von PdfAttachAssociateZUGFeRD verwendet werden.

- **PdfAttachAssociateSource**

Verknüpfung des Anhangs mit Beziehung "Source".

Mit dem Argument (alpha3) kann eine Beschreibung für den Anhang angegeben werden.

Mit dem Argument (logic4) wird definiert, ob die Anhangdatei komprimiert werden soll.



Eine Komprimierung führt nicht immer zu einer Verkleinerung des PDF. Im Extremfall ist das resultierende PDF mit Komprimierung größer als ohne. Dies hängt von den im Anhang enthaltenen Daten ab. Textdateien lassen sich meist gut komprimieren, wohingegen bereits komprimierte Dateien (z. B. ZIP-Dateien) sich nicht weiter komprimieren lassen.

Der Rückgabewert kann mit folgenden Konstanten verglichen werden:

Kontakt

__ErrOk kein Fehler aufgetreten
__ErrFsiNoFile Pfad- oder Dateiname nicht gefunden
__ErrOutOfMemory Arbeitsspeicher nicht ausreichend

Mögliche Laufzeitfehler:

__ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.
__ErrValueInvalid Kein Dateiname (alpha1) angegeben oder kein zulässiger Wert für
die Verknüpfungsart (int2).



obj -> PdfAttachInfoGet(int1, int2) : alpha

Informationen über einen Anhang eines PDF-Dokumentes ermitteln

obj Deskriptor des PDF-Objekts

int1 Nummer des Anhangs

Informationstyp

PdfAttachInfoName Name des Anhangs

PdfAttachInfoDescription Beschreibung des Anhangs

PdfAttachInfoSize Größe des Anhangs in Byte

int2 PdfAttachInfoDateCreate Erstellungsdatum des Anhangs

PdfAttachInfoTimeCreate Erstellungszeit des Anhangs

PdfAttachInfoDateModify Änderungsdatum des Anhangs

PdfAttachInfoTimeModify Änderungszeit des Anhangs

Resultat alpha Information 

Siehe Verwandte Befehle, PdfAttachExportFile(),
PdfAttachExportMem(), PdfAttachCount

Der Befehl liest Informationen einer Anhangdatei eines PDF-Dokumentes (obj) aus. Das PDF muss zuvor mit der Anweisung PdfOpen() geöffnet worden sein.

Im Argument (int1) muss die Nummer der Anhangdatei angegeben werden. Diese Nummer muss zwischen 1 und Der Anzahl Anhänge (siehe PdfAttachCount) liegen.

Zusätzlich wird in (int2) der Informationstyp angegeben. Folgende Werte können angegeben werden:

PdfAttachInfoName Name des Anhangs

PdfAttachInfoDescription Beschreibung des Anhangs

PdfAttachInfoSize Größe des Anhangs in Byte

PdfAttachInfoDateCreate Erstellungsdatum des Anhangs im internen Format (siehe FmtInternal)

PdfAttachInfoTimeCreate Erstellungszeit des Anhangs im internen Format (siehe FmtInternal)

PdfAttachInfoDateModify Änderungsdatum des Anhangs im internen Format (siehe FmtInternal)

PdfAttachInfoTimeModify Änderungszeit des Anhangs im internen Format (siehe FmtInternal)

Der Rückgabewert enthält die ermittelten Informationen des Anhangs. Im Fehlerfall wird der globale Fehlerwert gesetzt. Gab es einen Fehler beim Auslesen der Dateiinformatoren, wird der globale Fehlerwert auf ErrGeneric gesetzt.


Beispiel:

Kontakt

```
// Anhänge zählt AttachCount # tHdlPdf->spPdfAttachCount; // Anhänge durchgehen for tAttachNo #
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.
_ErrValueInvalid Ungültige Anhangsnummer (int1) oder ungültiger Informationstyp (int2) angegeben.

obj -> PdfAttachMem(handle1,
alpha2[, int3[, alpha4[, logic5]]) 
: int

Anhangdatei zu PDF hinzufügen

obj Deskriptor des
 PDF-Objekts

handle1 Memory-Objekt der
 Anhangdatei

alpha2 Name der
 Anhangdatei (nur
 intern)

int3 Verknüpfungsart der
 Anhangdatei

alpha4 Beschreibung der
 Anhangdatei

logic5 Anhang
 komprimieren

Resultat int Fehlerwert 

Siehe Verwandte Befehle,
 PdfAttachFile(),
 PdfAttachExportMem(),
 Blog

Der Befehl fügt einem PDF eine extern vorliegende Anhangdatei hinzu. Das PDF muss zuvor mit der Anweisung PdfOpen() geöffnet worden sein.

handle1 ist der Deskriptor des Memory-Objektes mit den Anhangdaten.

Im Argument (alpha2) wird ein interner Name für die einzufügende Anhangdatei angegeben.

Mit dem Argument (int3) kann die Verknüpfungsart definiert werden. Folgende Arten können angegeben werden:

- **PdfAttachAssociateAlternative**

Verknüpfung des Anhangs mit Beziehung "Alternative". Die Konstante sollte ab sofort anstelle von PdfAttachAssociateZUGFeRD verwendet werden.

- **PdfAttachAssociateSource**

Verknüpfung des Anhangs mit Beziehung "Source".

Mit dem Argument (alpha4) kann eine Beschreibung für den Anhang angegeben werden.

Mit dem Argument (logic5) wird definiert, ob die Anhangdatei komprimiert werden soll.



Eine Komprimierung führt nicht immer zu einer Verkleinerung des PDF. Im Extremfall ist das resultierende PDF mit Komprimierung größer als ohne. Dies hängt von den im Anhang enthaltenen Daten ab. Textdateien lassen sich meist

Kontakt

gut komprimieren, wohingegen bereits komprimierte Dateien (z. B. ZIP-Dateien) sich nicht weiter komprimieren lassen.

Der Rückgabewert kann mit folgenden Konstanten verglichen werden:

__ErrOk kein Fehler aufgetreten

__ErrOutOfMemory Arbeitsspeicher nicht ausreichend

Mögliche Laufzeitfehler:

__ErrHdlInvalid Der in (obj) oder (handle1) angegebene Deskriptor ist ungültig.

__ErrValueInvalid Kein Dateiname (alpha2) angegeben oder kein zulässiger Wert für die Verknüpfungsart (int3).

obj -> PdfClose([alpha1[, int2]]) :
int



PDF-Objekt schließen

obj Deskriptor des PDF-Objekts

alpha1 Dateiname der zu schreibenden Datei (optional)

Schreibmodus (optional)

PdfModePdfNormal

normales PDF
erzeugen

PdfModePdfA

PDF/A-1b:2005
erzeugen

PdfModePdfANormal

Versuchen
PDF/A-1b:2005 zu
erzeugen

PdfModePdfA2b

PDF/A-2b:2011
erzeugen

PdfModePdfA2bNormal

Versuchen
PDF/A-2b:2011 zu
erzeugen

PdfModePdfA3b

PDF/A-3b:2012
erzeugen

PdfModePdfA3bNormal

Versuchen
PDF/A-3b:2012 zu
erzeugen

PdfModeCancel

keine Datei erzeugen

int2 PdfModePdfZUGFeRDBasic

ZUGFeRD-konformes
PDF (Basic)
erzeugen

PdfModePdfZUGFeRDComfort

ZUGFeRD-konformes
PDF (Comfort)
erzeugen

PdfModePdfZUGFeRDExtended

ZUGFeRD-konformes
PDF (Extended)
erzeugen

PdfModePdfZUGFeRD10

ZUGFeRD
1.0-konformes PDF
erzeugen

PdfModePdfZUGFeRD20

ZUGFeRD
2.0-konformes PDF
erzeugen

PdfModePdfZUGFeRD21

ZUGFeRD
2.1-konformes PDF
erzeugen

PdfModePdfZUGFeRDXRechnung

ZUGFeRD
2.1-konformes PDF
mit
XRechnung-Profil

Resultat int

Fehlerwert

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

Mit dieser Anweisung wird ein zuvor mit [PdfNew\(\)](#) erstelltes bzw. [PdfOpen\(\)](#) geöffnetes PDF-Dokument zurückgeschrieben und das Objekt entfernt. Alle an dem Dokument durchgeführten Änderungen werden erst zu diesem Zeitpunkt in die Datei geschrieben.

Der Deskriptor des PDF-Dokuments wird als (obj) angegeben. Die zu schreibende Datei kann in (alpha1) übergeben werden. Wird kein Dateiname angegeben, wird das Dokument nicht geschrieben. In (int2) kann eine der folgenden Konstanten angegeben werden:

- **PdfModePdfNormal**

Es wird ein normales PDF-Dokument erzeugt.

- **PdfModePdfA**

Es wird ein PDF/A-1b:2005 konformes Dokument erzeugt.

- **PdfModePdfANormal**

Es wird versucht ein PDF/A-1b:2005 konformes Dokument zu erzeugen. Gelingt dies nicht, wird ein normales PDF-Dokument erstellt und [_ErrPdfNotPdfA](#) zurückgegeben.

- **PdfModePdfA2b**

Es wird ein PDF/A-2b:2011 konformes Dokument erzeugt.

- **PdfModePdfA2bNormal**

Es wird versucht ein PDF/A-2b:2011 konformes Dokument zu erzeugen. Gelingt dies nicht, wird ein normales PDF-Dokument erstellt und [_ErrPdfNotPdfA](#) zurückgegeben.

- **PdfModePdfA3b**

Es wird ein PDF/A-3b:2012 konformes Dokument erzeugt.

- **PdfModePdfA3bNormal**

Es wird versucht ein PDF/A-3b:2012 konformes Dokument zu erzeugen. Gelingt dies nicht, wird ein normales PDF-Dokument erstellt und [_ErrPdfNotPdfA](#) zurückgegeben.

- **PdfModePdfZUGFeRDBasic**

Erstellung eines ZUGFeRD-konformen PDF mit Basic-Profil.

- **PdfModePdfZUGFeRDComfort**

Erstellung eines ZUGFeRD-konformen PDF mit Comfort-Profil.

- **PdfModePdfZUGFeRDExtended**

Erstellung eines ZUGFeRD-konformen PDF mit Extended-Profil.

- **PdfModePdfZUGFeRD10**

Erstellung eines ZUGFeRD Version 1.0-konformen PDF.

- **PdfModePdfZUGFeRD20**

Erstellung eines ZUGFeRD Version 2.0-konformen PDF.

- **PdfModePdfZUGFeRD21**

Erstellung eines ZUGFeRD Version 2.1-konformen PDF.

- **PdfModePdfZUGFeRDXRechnung**

Erstellung eines ZUGFeRD Version 2.1-konformen PDF mit XRechnung-Profil.

- **PdfModeCancel**

Die Bearbeitung des PDF-Dokuments wird beendet, ohne das Dokument zu speichern.

Für ZUGFeRD-konforme PDFs kann je eine der Profil-Konstanten (PdfModePdfZUGFeRDBasic, PdfModePdfZUGFeRDComfort, PdfModePdfZUGFeRDExtended) mit je einer der Versions-Konstanten (PdfModePdfZUGFeRD10, PdfModePdfZUGFeRD20, PdfModePdfZUGFeRD21) kombiniert werden. Ist eine ZUGFeRD Profil-Konstante, jedoch keine Versions-Konstante angegeben, wird die Version 1.0 verwendet.



Die Konformität des XML-Anhangs beim Generieren einer ZUGFeRD-PDF-Datei wird nicht gewährleistet. Die XML-Datei kann mit Schema-Dateien von FeRD überprüft werden. Fertige PDF-Dateien können [hier](#) validiert werden.

Folgende Fehlercodes können zurückgegeben werden:

<u>ErrOk</u>	Kein Fehler aufgetreten.
<u>ErrGeneric</u>	Es ist ein nicht spezifizierten Fehler aufgetreten.
<u>ErrPdfNotPdfA</u>	Es konnte kein PDF/A-konformes Dokument erstellt werden.
<u>ErrOutOfMemory</u>	Der zur Erstellung des PDF-Dokuments notwendige Hauptspeicher konnte nicht allokiert werden.
<u>ErrFsiOpenFailed</u>	Beim erstellen der externen Datei ist ein Fehler aufgetreten.

Mögliche Laufzeitfehler:

<u>ErrHdlInvalid</u>	Der in (obj) angegebene Deskriptor ist ungültig.
<u>ErrValueInvalid</u>	In (int2) wurde ein ungültiger Wert angegeben.

obj -> PdfInsertImage(alpha1, float2, float3,
float4, float5, int6[, int7]) : int
Bild einfügen (BMP, GIF, JPEG, PNG und TIFF)



obj Deskriptor des
 PDF-Objekts
 Pfad- und
alpha1 Dateiname des
 Bildes
float2 Abstand vom
 linken Seitenrand
float3 Abstand vom
 oberen Seitenrand
float4 Breite des Bildes
float5 Höhe des Bildes
int6 Seitennummer
 Größenanpassung
int7 des Bildes oder
 des PDFs
 (optional)

Resultat int Fehlerwert

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

Die Anweisung fügt ein Bild der Formate BMP, GIF, JPEG, PNG oder TIFF in die zum Bearbeiten geöffnete Seite ein. Die Seite muss zuvor mit der Anweisung [PdfPageOpen\(\)](#) geöffnet werden. Das Bild wird an der angegebenen Position (float2 und float3) in der angegebenen Größe (float4 und float5) eingefügt. Die Positions- und Größenangaben erfolgen in Millimeter.

Beinhaltet die Bild-Datei mehr als ein Bild (zum Beispiel bei einem Multipage-TIFF), wird der Index des Bildes in (int6) übergeben.

Soll eine Anpassung der Größe des Bildes oder der Seite des PDF-Dokuments vorgenommen werden, kann eine der folgenden Konstanten in (int7) angegeben werden:

- **PdfInsertNormal**

Das Bild wird ohne Anpassungen des Bildes oder des PDF-Dokuments eingefügt.

- **PdfInsertFitImage**

Das Bild wird an das Pdf angepasst. Es wird entweder an die Seitenhöhe oder die Seitenbreite angepasst, sodass das komplette Bild in der maximalen Größe dargestellt wird. Das Verhältnis der Höhe zur Breite bleibt bestehen.

- **PdfInsertFitPdf**

Das Pdf wird auf die Größe des Bildes angepasst. Dabei kann bei den Parametern (float4) und (float5) angegeben werden, ob die Originalgröße des Bildes beibehalten werden soll (-1.0), oder es können die gewünschten Werte in Millimeter übergeben werden.

- **PdfOptUseLogSize**

Kontakt

Wird diese Konstante mit der Konstanten `_PdfInsertFitImage` oder `_PdfInsertFitPdf` kombiniert, wird beim Skalieren die logische Größe des Bildes berücksichtigt. Bei Kombination mit `_PdfInsertFitPdf` muss die Breite (float4) und die Höhe (float5) jeweils -1.0 sein.



Die eingefügten Bilder werden anhand der übergebenen Argumente zwischengespeichert. Bei erneutem Aufruf mit den selben Argumenten, wird das bereits gespeicherte Bild eingefügt. Die Größe und das Änderungsdatum der Datei werden hierbei nicht berücksichtigt.

Der Rückgabewert kann mit folgenden Konstanten verglichen werden:

<u><code>_ErrOk</code></u>	kein Fehler aufgetreten
<u><code>_ErrFsiNoFile</code></u>	Pfad- oder Dateiname nicht gefunden
<u><code>_ErrOutOfMemory</code></u>	Arbeitsspeicher nicht ausreichend
<u><code>_ErrPdfPageClosed</code></u>	keine Seite geöffnet
<u><code>_ErrPdfImageFormat</code></u>	nicht unterstütztes Bildformat
<u><code>_ErrGeneric</code></u>	anderer Fehler aufgetreten

Mögliche Laufzeitfehler:

`_ErrHdlInvalid` Der in (obj) angegebene Deskriptor ist ungültig.


obj -> PdfInsertMeta(alpha1,
float2, float3, float4, float5) : 
int

Bild einfügen (WMF und EMF)

	Deskriptor
obj	des PDF-Objekts
	Pfad- und
alpha1	Dateiname des Bildes
	Abstand vom
float2	linken Seitenrand
	Abstand vom
float3	oberen Seitenrand
	Breite des
float4	Bildes
	Höhe des
float5	Bildes

Resultat int Fehlerwert

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

Die Anweisung fügt ein Windows Metafile (WMF) oder ein Enhanced-Metafile (EMF) in die zum Bearbeiten geöffnete Seite ein. Die Seite muss zuvor mit der Anweisung [PdfPageOpen\(\)](#) geöffnet werden. Das Bild wird an der angegebenen Position (float2 und float3) in der angegebenen Größe (float4 und float5) eingefügt. Die Positions- und Größenangaben erfolgen in Millimeter.

Konnte das Bild eingefügt werden, wird [_ErrOk](#), sonst [_ErrPdfInsertMetaFile](#) zurückgegeben.

Mögliche Laufzeitfehler:

[_ErrHdlInvalid](#) Der in (obj) angegebene Deskriptor ist ungültig.

PdfNew([alpha1[,
alpha2]]) : handle



PDF-Dokument erstellen

alpha1 Kennwort des Eigentümers

alpha2 Kennwort zum Öffnen


Resultat handle Deskriptor des
PDF-Objekts

Siehe Verwandte Befehle, PDF,
PdfClose(), PdfOpen(),
PDF-Verarbeitung (Blog)

Diese Anweisung öffnet ein neues PDF-Objekt. Der Deskriptor auf das Objekt wird von der Anweisung zurückgegeben. Das PDF-Dokument kann mit einem Eigentümer-Kennwort und/oder einem Öffnen-Kennwort versehen werden, diese können in (alpha1) und (alpha2) angegeben werden. Das Dokument wird erst mit der Anweisung PdfClose() extern gespeichert.

Wurde ein Eigentümer- oder Öffnen-Kennwort angegeben, können die Berechtigungen über die Eigenschaft PdfUserRights gesetzt werden. Ohne die Angabe eines Kennwortes wird diese Eigenschaft nicht ausgewertet.

Standardmäßig werden Seiten in dem Dokument in der Größe DinA4 angelegt. Die Seitengröße kann über die Eigenschaften PdfPageWidth und PdfPageHeight definiert werden.

PdfOpen(alpha1[,
int2, alpha3[, int4]]) : 
handle

PDF-Datei öffnen

alpha1 Pfad- und Dateiname

Art des Kennworts (optional)

_PdfOpenDefault Das

Open-Kennwort

int2 ist angegeben

_PdfOpenOwner Das

Owner-Kennwort

ist angegeben

alpha3 Kennwort (optional)

Inhalte des PDF-Dokuments

_PdfImportDefault Import ohne

dynamische

int4 Inhalte

_PdfImportAll Import aller

Inhalte

Resultat handle Deskriptor des
PDF-Objekts

Verwandte Befehle, PDF,

Siehe PdfClose(), PdfNew(),

PDF-Verarbeitung (Blog)

Diese Anweisung öffnet die in (alpha1) angegebene PDF-Datei für die Verarbeitung. Kann die Datei geöffnet werden, wird der Deskriptor auf das PDF-Objekt zurückgegeben.

Ist das PDF-Dokument durch ein Kennwort geschützt, kann das Kennwort in (alpha3) angegeben werden. (int2) bestimmt, ob das Kennwort zum Öffnen des Dokuments (_PdfOpenDefault) oder das Kennwort des Eigentümers (_PdfOpenOwner) übergeben wurde.

In (int4) kann der zu importierende Inhalt aus dem PDF-Dokument eingeschränkt werden:

- **_PdfImportDefault**

Es werden keine dynamischen Inhalte, wie zum Beispiel Formularfelder, importiert.


- **_PdfImportAll**

Es werden alle Inhalte des PDF-Dokuments importiert. Das kann dazu führen, dass JavaScript-Aktionen durchgeführt werden, wenn das Dokument JavaScript enthält.

Tritt beim Öffnen des Dokuments ein Fehler auf, wird einer der folgenden Fehlerwerte zurückgegeben:

Kontakt

<u>ErrFsiNoFile</u>	Der Pfad oder die Datei ist nicht vorhanden.
<u>ErrFsiInvalidFormat</u>	Die Datei ist nicht im PDF-Format oder ist defekt.
<u>ErrPdfPassword</u>	Es wurde das falsche Owner- bzw. Open-Kennwort bei einer verschlüsselten PDF-Datei angegeben.
<u>ErrOutOfMemory</u>	Es steht nicht genügend Hauptspeicher zum Öffnen der Datei zur Verfügung.
<u>ErrGeneric</u>	Es ist ein anderer Fehler aufgetreten.
Mögliche Laufzeitfehler:	
<u>ErrValueInvalid</u>	Der in (int2) angegebene Parameter ist ungültig.

obj -> PdfPageClose([int1]) 

Bearbeitete Seite schließen

Deskriptor

obj des
PDF-Objekts

Seite
speichern

int1 oder
verwerfen
(optional)


Siehe Verwandte
Befehle

Mit der Anweisung wird eine Seite, die zuvor mit PdfPageOpen() zur Bearbeitung geöffnet wurde, wieder geschlossen. Als (obj) wird der Deskriptor des entsprechenden PDF-Objekts übergeben.

In (int1) kann angegeben werden, ob die Seite gespeichert (_PdfPageCloseNormal) oder die Änderungen verworfen (_PdfPageCloseCancel) werden sollen. Wird kein Parameter angegeben, wird die Seite gespeichert. In jedem Fall wird die Seite aus dem Speicher entfernt und steht nicht mehr für die entsprechenden Anweisungen für die Bearbeitung von PDF-Seiten zur Verfügung.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

obj -> PdfPageOpen(int1) : int 

Seite anlegen oder bearbeiten

Deskriptor

obj eines
PDF-Objekts

int1 Nummer der
Seite

Resultat int Fehlerwert


Siehe Verwandte
Befehle

Mit dieser Anweisung wird eine Seite eines zuvor mit PdfOpen() geöffneten oder PdfNew() angelegten PDF-Dokuments zur Bearbeitung geöffnet. Die Seitennummer wird in (int1) angegeben. Ist die Seite noch nicht vorhanden, wird sie erzeugt.

Von einem Dokument kann immer nur eine Seite zur Bearbeitung geöffnet werden. Konnte die Seite geöffnet werden wird ErrOk, sonst ErrGeneric zurückgegeben.


Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.

PdfSavePage(alpha1, alpha2, int3[, point4[, point5[, int6[, alpha7]]]]) : int 

Grafikdatei aus einer Seite des PDF-Dokuments erzeugen

alpha1 Dateiname der
PDF-Datei
alpha2 Dateiname der
Grafik-Datei
int3 Seitennummer
point4 Größe des Bildes
(optional)
point5 Auflösung des
Bildes (optional)
int6 Qualität des
Bildes (nur
JPEG) (optional)
alpha7 Kennwort der
PDF-Datei
(optional)

Resultat int Fehlerwert 

Siehe Verwandte
Befehle

Der Befehl erzeugt eine Grafik-Datei aus einer Seite einer PDF-Datei. Der Pfad und der Name der PDF-Datei wird in (alpha1) angegeben. Die Angabe der zu erzeugenden Grafik-Datei erfolgt in (alpha2). Abhängig von der angegebenen Dateinamen-Erweiterung wird eine Datei des entsprechenden Formats erzeugt.

- .jpg, .jpeg - JPEG-Grafik
- .tif, .tiff - TIFF-Grafik
- .png - PNG-Grafik
- .bmp - Windows-Bitmap

Bei anderen Erweiterungen oder wenn keine Erweiterung vorhanden ist, wird als Resultat _ErrGeneric zurückgegeben.



Ist die in (alpha2) angegebene Datei bereits vorhanden, wird sie überschrieben.

Die Seite, die in ein Bild gewandelt werden soll, wird in (int3) angegeben. Die Nummer muss im Bereich 1 bis Anzahl der Seiten im PDF liegen. Bei einem Wert außerhalb liefert die Anweisung das Resultat _ErrPdfPageNotExisting.

Die Größe des Bildes wird in (point4) angegeben. Wird das Argument nicht angegeben oder (0, 0) übergeben, wird die Seitengröße der PDF-Seite verwendet.

In (point5) kann die Auflösung des resultierenden Bildes angegeben werden. Die Komponente :x enthält dabei die horizontale, die Komponente :y die vertikale Auflösung. Wird das Argument nicht angegeben oder (0, 0) übergeben, wird ein Bild mit 72x72 DPI generiert. Ist nur einer der beiden Werte 0, dann ist die Auflösung in beiden Dimensionen identisch. D. h. bei der Übergaben von (300, 0) wird ein Bild mit der Auflösung 300x300 DPI erzeugt.

Kontakt

Handelt es sich bei der erzeugten Grafik um eine JPEG-Datei, kann in (int6) die Qualität des Bildes angegeben werden. Der Wert muss im Bereich 1 (schlecht) bis 100 (gut) liegen. Liegt er außerhalb wird er entsprechend modifiziert. Die Angabe wird bei anderen Grafik-Formaten ignoriert.

Ist das in (alpha1) angegebene PDF-Dokument durch ein Kennwort geschützt, muss dieses Kennwort in (alpha7) angegeben werden.

Die Anweisung gibt einen Fehlerwert zurück. Der Wert kann mit folgenden Konstanten verglichen werden:

<u>_ErrOk</u>	Datei wurde erfolgreich erstellt.
<u>_ErrOutOfMemory</u>	Speicher nicht ausreichend.
<u>_ErrFsiInvalidFormat</u>	Keine PDF-Datei oder PDF-Datei korrupt.
<u>_ErrPdfPassword</u>	PDF-Datei ist kennwortgeschützt bzw. Kennwort falsch.
<u>_ErrGeneric</u>	Ein unbekannter Fehler ist aufgetreten. Dieser Fehler wird auch zurückgegeben, wenn die Dateierweiterung der Ausgabedatei nicht zulässig ist.
<u>_ErrPdfPageNotExisting</u>	Die angegebene Seite existiert nicht.
<u>_ErrFsiOpenFailed</u>	Die Ausgabedatei konnte nicht angelegt werden.
<u>_ErrFsiWriteFault</u>	Fehler beim Schreiben der Ausgabedatei.

obj ->

PdfTextColor(int1) : int

Farbe für Text festlegen

Deskriptor

obj des

PDF-Objekts

int1 Farbe des

Textes

Resultat int Fehlerwert

Siehe Verwandte

Befehle

Diese Anweisung setzt die Farbe des Fonts, der für die nachfolgenden PdfTextWrite()-Anweisungen verwendet wird. Als (obj) wird ein PDF-Objekt angegeben. Die Farbe wird als RGB-Wert oder mit einer der Farbkonstanten (_WinColBlack, _WinColBlue, _WinColGreen, _WinColRed, _WinColYellow, _WinColMagenta, _WinColCyan, _WinColWhite, _WinColDarkGray, _WinColLightBlue, _WinColLightGreen, _WinColLightRed, _WinColLightYellow, _WinColLightMagenta, _WinColLightCyan, _WinColLightGray) angegeben.



Die Farbe muss nach dem Öffnen einer Seite im PDF-Dokument angegeben werden und ist bis zum Schließen der Seite gültig. Bei der Bearbeitung mehrerer Seiten, muss nach jedem Öffnen einer Seite mit der Anweisung PdfPageOpen() die Farbe gesetzt werden.>

Konnte die Farbe gesetzt werden, gibt die Anweisung _ErrOk, sonst _ErrGeneric zurück.

Beispiel:

```
...tHdlPdf # PdfOpen(_Sys->spPathMyDocuments + '\Document.pdf');tHdlPdf->PdfPageOpen(1);tColor->
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

obj -> PdfTextExtractMem(handle1) : int
Textinformationen einer PDF-Seite ermitteln



obj Deskriptor eines
 PDF-Objekts

handle1 Deskriptor eines
 Memory-Objekts

Resultat int Fehlerwert

Verwandte


Siehe Befehle,
 PdfPageOpen()

Mit dieser Anweisung wird die Textinformation einer zuvor mit PdfPageOpen() geöffneten Seite eines PDF-Dokuments ermittelt und in das Memory-Objekt (handle1) übertragen.

Ist keine Seite geöffnet, gibt die Anweisung den Fehlerwert _ErrPdfPageClosed zurück. Der ermittelte Text wird an das Memory-Objekt angehängt (siehe Len). Der Zeichensatz des Memory-Objekts wird entsprechend der Eigenschaft Charset berücksichtigt.

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) angegebene Deskriptor ist ungültig.

obj -> PdfTextFont(alpha1, ) : int
float2[, int3[, logic4]]) : int

Font für Text festlegen

obj Deskriptor des PDF-Objekts

alpha1 Name des Fonts

float2 Größe des Font

Font-Attribute (optional)

_WinFontAttrNormal Normale
Darstellung

int3 _WinFontAttrItalic Kursiv

_WinFontAttrBold Fett

_WinFontAttrUnderline Unterstrichen

_WinFontAttrStrikeOut Durchgestrichen

logic4 Font einbetten (optional)

Resultat int Fehlerwert

Siehe Verwandte Befehle

Diese Anweisung lädt einen Font, der für die nachfolgenden

PdfTextWrite()-Anweisungen verwendet wird. Als (obj) wird ein PDF-Objekt

angegeben. Der Name des Fonts wird in (alpha1), seine Größe in (float2) angegeben.

Die Angabe der Größe erfolgt in Millimetern.



Der Font muss nach dem Öffnen einer Seite im PDF-Dokument angegeben werden und ist bis zum Schließen der Seite gültig. Bei der Bearbeitung mehrerer Seiten, muss nach jedem Öffnen einer Seite mit der Anweisung PdfPageOpen() der Font gesetzt werden.

In (int3) können die Font-Attribute angegeben werden. Sind keine Font-Attribute angegeben, erfolgt die normale Darstellung. Es kann eine Kombination aus folgenden Konstanten angegeben werden:

_WinFontAttrNormal Normale Darstellung

_WinFontAttrItalic Kursiv

_WinFontAttrBold Fett

_WinFontAttrUnderline Unterstrichen

_WinFontAttrStrikeOut Durchgestrichen

Über den Parameter (logic4) kann bestimmt werden, ob der Font in das Dokument eingebunden wird (true), oder nicht (false). Standardmäßig wird der Font nicht eingebunden. Eingebundene Fonts müssen auf dem System, auf dem das Dokument angezeigt werden soll, nicht vorhanden sein. Ist ein Font nicht eingebunden und nicht auf dem System installiert, wird ein ähnlicher Font zur Anzeige des Textes durch das System gewählt.

Konnte der Font gesetzt werden, gibt die Anweisung _ErrOk, sonst _ErrGeneric zurück.

Beispiel:

```
tHdLPdf # PdfOpen(_Sys->spPathMyDocuments + '\Document.pdf');tHdLPdf->PdfTextFont('Arial', 5.0, _
```


Mögliche Laufzeitfehler:

ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

obj ->
 PdfTextWrite(float1,
 float2, alpha3)



: int

Text schreiben

obj Deskriptor
 des
 PDF-Objekts
 Abstand vom
float1 linken
 Seitenrand
 Abstand vom
float2 oberen
 Seitenrand
 zu
alpha3 schreibender
 Text

Resultat int Fehlerwert

Siehe Verwandte
Befehle

Diese Anweisung schreibt einen Text in ein PDF-Dokument. In (obj) wird der Deskriptor des PDF-Objekts angegeben. Die zu bearbeitende Seite muss zuvor mit PdfPageOpen() geöffnet worden sein. Die Position des Textes wird in den Parametern (float1) und (float2) in Millimetern angegeben. Der zu schreibende Text wird in (alpha3) übergeben.

Der zu verwendende Font und die Textfarbe müssen zuvor mit den Anweisungen PdfTextFont() und PdfTextColor() definiert werden.

Konnte der Text geschrieben werden, gibt die Anweisung _ErrOk, sonst _ErrGeneric zurück.

Beispiel:

```
tHdlPdf # PdfOpen(_Sys->spPathMyDocuments + '\Document.pdf');if (tHdlPdf->PdfPageOpen(1) = _ErrOk
```

Mögliche Laufzeitfehler:

_ErrHdlInvalid Der in (obj) übergebene Deskriptor ist ungültig.

Befehle für Diagramme

Liste der Befehle zur Bearbeitung von Diagrammen

Siehe [Befehlsgruppen](#),
[Befehlsliste](#)

Befehle

- [ChartClose](#)
- [ChartDataAdd](#)
- [ChartDataClose](#)
- [ChartDataOpen](#)
- [ChartDataSort](#)
- [ChartOpen](#)
- [ChartSave](#)

Erstellen von Diagrammen

Ein Diagramm wird aus zwei verschiedenen Objekten erstellt. Den äußeren Rahmen eines Diagramms stellt das [Chart](#)-Objekt zur Verfügung. In den Eigenschaften dieses Objekts werden der Typ des Diagramms und alle Form gebenden Charakteristika angegeben. Die darzustellenden Daten werden in einem oder mehreren [ChartData](#)-Objekten angegeben. Die Darstellung einer Datenreihe erfolgt dabei auf Basis der Eigenschaften des [Chart](#)-Objekts. Durch Änderung der Eigenschaften können auch unterschiedliche Darstellungen der Datenreihen erfolgen.

Ein Diagramm wird mit der Anweisung [ChartOpen\(\)](#) erzeugt. Dabei wird der Diagramm-Typ, die Größe des Ausgabebereiches und der Titel des Diagramms angegeben.

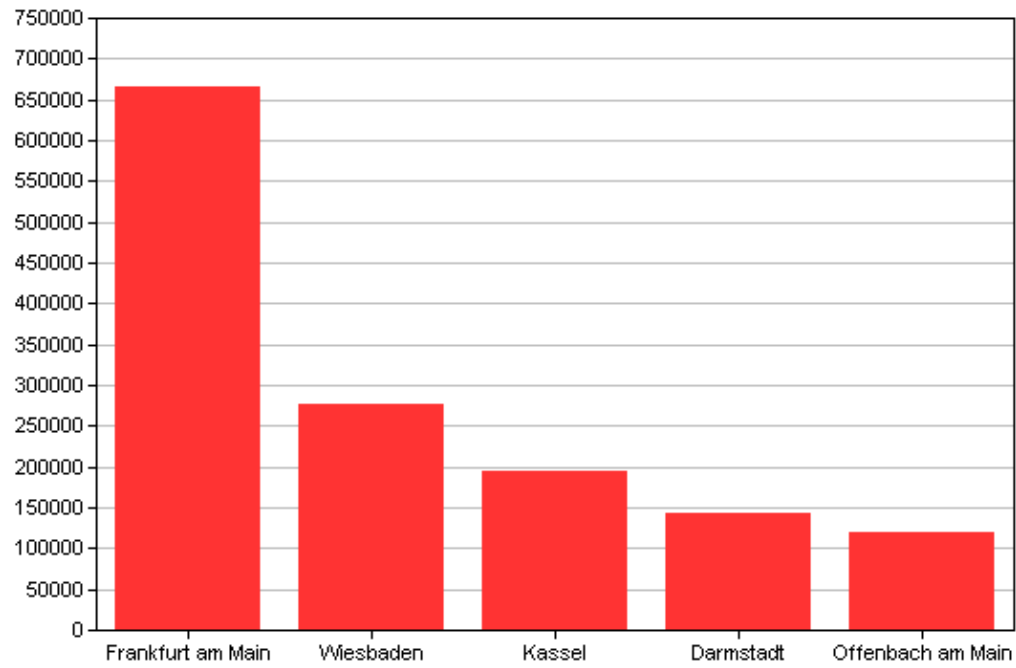
Anschließend wird ein [ChartData](#)-Objekt erzeugt. Diesem Objekt werden mit der Anweisung [ChartDataAdd\(\)](#) die darzustellenden Daten zugewiesen. In dem folgenden Beispiel sind das die Einwohnerzahlen der fünf größten Städte Hessens (Stand 2008) und die Bezeichner der einzelnen Daten (die Namen der Städte). Die Zuweisung kann ebenfalls über Arrays erfolgen, wird hier aber der Einfachheit wegen einzeln durchgeführt.

Die Ausgabe des Diagramms erfolgt in eine externe Datei ([ChartSave\(\)](#)).

Die Möglichkeiten bei der Erstellung von Diagrammen sind in einem Beispiel in der Datenbank "CodeLibrary" ausgeführt. Die Beschreibung der Eigenschaften befinden sich im Abschnitt [Eigenschaften eines Chart-Objekts](#).

```
@A+@C+mainlocal{ tChart      : handle; tChartData : handle;}{ tChart # ChartOpen(_ChartXY, 800,
```

Die fünf größten Städte Hessens




ChartOpen(int1, int2, int3[, 
alpha4[,int5]]) : handle

Chart-Grafik anlegen

	Typ der Chart-Grafik	
	_ChartPie	Torten-Diagramm
int1	_ChartXY	Koordinaten-Diagramm
	_ChartPyramid	Pyramiden-Diagramm
	_ChartSurface	Oberflächen-Diagramm
int2	Breite des <u>Chart</u> -Objekts	
int3	Höhe des <u>Chart</u> -Objekts	
alpha4	Titel der Grafik (optional)	
	Legende (optional)	
	_ChartOptDefault	keine Legende
int5	_ChartOptLegendVertical	vertikal angeordnete Legende
	_ChartOptLegendHorizontal	horizontal angeordnete Legende

Resultat handle Deskriptor des 
Chart-Objekts

Siehe Verwandte Befehle, ChartClose()

Mit dieser Anweisung wird ein Chart-Objekt angelegt. Der Typ der Grafik wird im Parameter (int1) angegeben. Folgende Konstanten stehen dafür zur Verfügung:

- **_ChartPie**

Es wird eine Tortengrafik angelegt. Es können Torten- oder Donut-Grafiken erzeugt werden. Donut-Grafiken werden dabei durch die zusätzliche Angabe eines Innenradius in der Eigenschaft ChartPieInnerRadius erstellt. Die einzelnen Daten werden als Teil der Gesamtmenge angezeigt.

- **_ChartXY**

Es wird ein Koordinaten-Diagramm angezeigt. Es können Balken-, Linien- oder Flächen-Grafiken erzeugt werden. Die Darstellung wird über die Eigenschaft ChartXYStyleData bestimmt.

- **_ChartPyramid**

Es wird ein Pyramiden-Diagramm angezeigt. Es können Pyramiden-, Kegel- oder Trichter-Grafiken erzeugt werden. Die Darstellung wird über die Eigenschaft ChartPyramidStyleData bestimmt.

- **_ChartSurface**

Es wird ein Oberflächen-Diagramm angezeigt.

Die Parameter (int2) und (int3) bestimmen die Breite und Höhe der Grafik. Die Angaben erfolgen in Pixel. In (alpha4) kann ein Titel für die Grafik angegeben werden. Der Titel wird oben innerhalb der Grafik angezeigt.