Turbo Pascal Hilfeindex

- Assembler, integriert
- Kommandozeilenoptionen
- Compiler-Befehle
  - Parameter-Direktiven
  - Schalter-Befehle
- Konstanten
  - Turbo Pascal
  - ObjectWindows
  - Windows API
- Dialogfenster
  - Turbo Pascal IDE
  - ObjectWindows
  - Windows API
- Direktiven
  - Compiler-Befehle
  - Standardanweisungen
- Dynamische Linkbibliotheken (DLLs)
- Editorbefehle
  - Blockbefehle (CUA und Alternativ)
  - Blockbefehle (Borlandstil)
  - Cursorbewegung
  - Einfügen und Löschen
  - Diverse Befehle
- Fehlermeldungen
  - Compilierzeit
  - Laufzeit
- Funktionen
  - Standard
  - ObjectWindows
  - Windows API
- Grafische Geräteschnittstelle (GDI)
- Menüs
  - Turbo Pascal IDE
- Methoden
  - Definiert
  - ObjectWindows
- ObjectWindows
  - Konstanten
  - Felder
  - Funktionen
  - Hierarchie
  - Methoden
  - Objekte
  - Prozeduren
  - Records
  - Typen
  - Variablen
  - WObjects.TPU
- Operatoren
- Optionen
  - Kommandozeile (TPCW.EXE)
  - Compiler
  - Linker

- Suchen & Ersetzen

Prozeduren
- Standard
- ObjectWindows
- Windows API

Reservierte Wörter

Ressourcen

Strings
- Null-terminiert
- Strings.TPU
- Turbo Pascal

Turbo Pascal Sprachspezifikation

Typen
- Standard Turbo Pascal
- ObjectWindows
- Windows API

Units
- Strings.TPU
- System.TPU
- WinCrt.TPU
- WinDos.TPU
- WinProcs.TPU
- WinTypes.TPU
- WObjects.TPU

Variablen

Fenster-Objekte

Windows API
- Konstanten
- Funktionen
- Meldungen
- Prozeduren
- Records
- Typen
- WinProcs.TPU
- WinTypes.TPU

Turbo Pascal IDE Dialogfenster

Die integrierte Entwicklungsumgebung (IDE) von Turbo Pascal für Windows enthält folgende Dialogfenster:

Compiler Options (Compiler-Optionen)
Configuration Save As (Konfiguration speichern unter)
Directories (Verzeichnisse)
File Open (Datei öffnen)
File Save As (Datei speichern unter)
Find Text (Suche nach Text)
Linker Options (Linker-Optionen)
Open Configuration File (Konfigurationsdatei laden)
Parameters (Parameter)
Preferences (Vorgaben)
Primary File (Hauptdatei)
Replace Text (Text ersetzen)
Select Printer (Drucker auswählen)
Set Up (Drucker)
Task List (Task-Liste)

Turbo Pascal IDE Menübefehle

Systemmenü (Turbo Pascal)

- **Restore (Wiederherstellen)**
- **Move (Verschieben)**
- **Size (Größe ändern)**
- **Minimize (Symbol)**
- **Maximize (Vollbild)**
- **Close (Schließen)**
- **Switch to (Wechseln zu)**

Systemmenü (Edit- bzw. Bearbeiten-Fenster)

- **Restore (Wiederherstellen)**
- **Move (Verschieben)**
- **Size(Größe ändern)**
- **Minimize (Symbol)**
- **Maximize (Vollbild)**
- **Close (Schließen)**
- **Next (Nächstes)**

File- bzw. Datei-Menü

- **New (Neu)**
- **Open (Öffnen)**
- **Save (Speichern)**
- **Save As (Speichern unter)**
- **Save All (Alles speichern)**
- **Print (Drucken)**
- **Printer Setup (Druckerinstallation)**
- **Exit (Beenden)**
- **Liste geschlossener Dateien**

Edit- bzw. Bearbeiten-Menü

- **Undo (Rückgängig)**
- **Redo (Widerrufen)**
- **Cut (Ausschneiden)**
- **Copy (Kopieren)**
- **Paste (Einfügen)**
- **Clear (Löschen)**

Search- bzw. Suchen-Menü

- **Find (Suchen nach)**
- **Replace (Ersetzen)**
- **Search Again (Weitersuchen)**
- **Go to Line Number (Gehe zu Zeile)**
- **Show Last Compile Error (Letzter Compiler-Fehler)**
- **Find Error (Fehler suchen)**

Run- bzw. Ausführen-Menü

- **Run (Ausführen)**
- **Debugger (Debugger)**
- **Parameters (Parameter)**

Compile- bzw. Compilieren-Menü

- **Compile (Compilieren)**
- **Make (Projekt aktualisieren)**
- **Build (Alle Projektdateien compilieren)**

Primary File (Hauptdatei)
Clear Primary File (Eintrag für Hauptdatei löschen)
Information (Information)

Options- bzw. Opotionen-Menü
Compiler (Compiler)
Linker (Linker)
Directories (Verzeichnisse)
Preferences (Vorgaben)
Open (Laden)
Save (Speichern)
Save As (Speichern unter)

Window- bzw. Fenster-Menü
Tile (Nebeneinander)
Cascade (Überlappend)
Arrange Icons (Symbole anordnen)
Close All (Alle schließen)

Help- bzw. Hilfe-Menü
Index (Index)
Topic Search (Suche über Schlüsselwort)
Using Help (Hilfe über Hilfe)
Compile Directives (Compiler-Befehle)
ObjectWindows (ObjectWindows)
Procedures and Functions (Prozeduren und Funktionen)
Reserved Words (Reservierte Wörter)
StandardUnits (Standard Units)
Turbo Pascal Language (Sprachspezifikation)
Windows API (Windows API)
About Turbo Pascal (Info über Turbo Pascal)

Index der Turbo Pascal Konstanten

Kategorien von Konstanten

- **Dateiattribut-Konstanten**
- **Dateimodus-Konstanten**
- **Dateiname-Länge**
- **Dateiaufteilungs-Flags**
- **Flag-Konstanten**

Alphabetische Liste der Konstanten

- **faAnyFile**
- **faArchive**
- **faDirectory**
- **faHidden**
- **faReadOnly**
- **faSysFile**
- **fAuxiliary**
- **faVolumeID**
- **fCarry**
- **fcDirectory**
- **fcExtension**
- **fcFileName**
- **fcWildcards**
- **fmClosed**
- **fmInOut**
- **fmInput**
- **fmOutput**
- **fOverflow**
- **fParity**
- **fsDirectory**
- **fsExtension**
- **fsFileName**
- **fSign**
- **fsPathName**
- **fZero**

Index der Turbo Pascal Variablen

AutoTracking	WinCrt.TPU
CheckBreak	WinCrt.TPU
CheckEOF	WinCrt.TPU
CmdLine	System.TPU
CmdShow	System.TPU
Cursor	WinCrt.TPU
DosError	WinDos.TPU
ErrorAddr	System.TPU
ExitCode	System.TPU
ExitProc	System.TPU
FileMode	System.TPU
HeapBlock	System.TPU
HeapError	System.TPU
HeapLimit	System.TPU
HeapList	System.TPU
HInstance	System.TPU
HPrevInst	System.TPU
InactiveTitle	WinCrt.TPU
InOutRes	System.TPU
Input	System.TPU
Origin	WinCrt.TPU
Output	System.TPU
PrefixSeg	System.TPU
RandSeed	System.TPU
ScreenSize	WinCrt.TPU
WindowOrg	WinCrt.TPU
WindowSize	WinCrt.TPU
WindowTitle	WinCrt.TPU

Index der reservierten Wörter von Turbo Pascal

and
asm
array
begin
case
const
constructor
destructor
div
do
downto
else
end
exports
file
for
function
goto
if
implementation
in
inline
interface
label
library
mod
nil
not
object
of
or
packed
pointer
procedure
program
record
repeat
set
shl
shr
string
then
to
type
unit
until
uses
var
while
with
xor

Siehe auch

Standard-Befehle

Standardanweisungen und Prozeduranweisungen

Wir verstehen darunter die Standardanweisungen von Turbo Pascal, von denen manche auch Prozeduranweisungen genannt werden.

Im Gegensatz zu **reservierten Wörtern** können diese Begriffe neu definiert werden, wozu aber nicht geraten wird.

absolute
assembler (Prozeduranweisung)
export (Prozeduranweisung)
external (Prozeduranweisung)
far (Prozeduranweisung)
forward (Prozeduranweisung)
index
interrupt (Prozeduranweisung)
name
near (Prozeduranweisung)
private
resident
virtual (Prozeduranweisung)

absolute

Deklariert eine **Variable** an einer absoluten Adresse.

Syntax

```
  var ident: type absolute seg:ofs;
```

oder

```
  var ident: type absolute variable;
```

Die erste Form gibt die Adresse der Variablen direkt an (segment und offset).

Die zweite Form deklariert eine neue Variable zu der bereits vorhandenen Variablen (an derselben Adresse).

Beispiel:

```
  type
    VectorTable = array[0..255] of pointer;
  var
    IntVectors: VectorTable absolute 0:0;
    CrtMode: Byte absolute $0040:$0049;
    Str: string;
    StrLen: Byte absolute Str;
```

array (reserviertes Wort)

Dient zur Definition eines Arraytyps.

Syntax

```
array [index-type] of element-type
```

Verschiedene Indextypen sind erlaubt, wenn sie durch Kommata voneinder getrennt werden.

Als Elementtyp ist jeder **Typ** zulässig, der Indextyp muß jedoch ein **ordinaler** Typ sein.

Beispiel:

```
type
  IntList  = array[1..100]     of Integer;
  CharData = array['A'..'Z']   of Byte;
  Matrix   = array[0..9, 0..9] of real;
```

Siehe auch

Arraytypen-Konstanten

asm (reserviertes Wort)

Greift auf den integrierten Assembler zu.

Syntax

```
asm
  AssemblerStmt <Separator AssemblerStmt>
end
```

- AssemblerStmt ist eine **Assembler-Anweisung**
- Separator ist ein Strichpunkt, Zeilenende oder ein Pascal-Kommentar

Mehrfache Assembleranweisungen in einer Zeile werden durch Strichpunkte getrennt.

In einer **asm-Anweisung** bedeutet ein Strichpunkt nicht, daß der Rest der Zeile Kommentar ist. Kommentare müssen im Pascal-Stil mit { und } oder (* und *) gekennzeichnet werden.

Die Verwendung von Registern

Für die Anwendung von Registern in einer **asm**-Anweisung gelten dieselben Regeln wie die einer **external** Prozedur oder Funktion.

Eine **asm-**Anweisung muß folgende Register speichern:

BP	SP
SS	DS

Eine **asm**-Anweisung kann folgende Register ändern:

AX	BX
CX	DX
SI	DI
ES	Flags

Beim Eintritt in die **asm**-Anweisung kennt der integrierte Assembler nur die Register BP, SP, SS und DS.

assembler

Mit der **assembler Anweisung** können Sie vollständige Prozeduren und Funktionen schreiben, ohne **begin...end** Anweisung.

Die **assembler**-Anweisung veranlaßt den Turbo Pascal Compiler, die Code-Generierung so zu optimieren:

- Wert-Parameter: Der Compiler generiert keinen Code, der Wert-Parameter in lokale Variablen kopiert.
- Funktionsergebnis-Variable: Der Compiler weist keine Funktionsergebnis-Variable zu; eine Referenz auf das Symbol **@Result** ist ein Fehler.
- Stack Frame: Der Compiler generiert keinen Stack Frame für Prozeduren und Funktionen, die keine Parameter oder lokale Variablen besitzen.

Die Wert-Parameter-Optimierung betrifft alle String-Parameter und andere Wert-Parameter, deren Größe nicht 1, 2, oder 4 Bytes ist.

Innerhalb der Prozedur oder Funktion müssen solche Parameter wie **var** Parameter behandelt werden.

String-Funktionen sind eine Ausnahme von Funktionsergebnis-Optimierungen. Sie haben immer einen @Result-Zeiger, der vom Aufrufer zugewiesen wird.

Siehe auch

Inline-Assembler Eintritts- und Austrittscode

assignment Operator (:=)

Weist den Wert eines Ausdrucks einer Variablen zu.

Syntax

```
variable := expression
```

Die Variable muß zuweisungskompatibel zum Ergebnistyp des Ausdrucks.

Beispiel:

```
X    := Y;
Done := (I > 0) and (I < 100)
A[I] := A[I] + 1;
```

Siehe auch

expression
statement
variable

begin...end Konstrukt (reservierte Wörter)

Dieses Konstrukt ist eine zusammengesetzte Anweisung. Die reservierten Wörter **begin** und **end** dienen als Anweisungsklammern.

Syntax

```
begin
  statement;
  statement;
  ...
  statement
end;
```

Wenn **Anweisungen** wie beschrieben geklammert werden, werden sie als eine einzelne Anweisung behandelt.

Beispiel:

```
(* Compound statement used within an "if" statement *)
if First < Last then
begin
  Temp := First;
  First := Last;
  Last := Temp;
end;
```

Boolesche Typen

Es gibt drei vordefinierte Boolesche Typen: Boolean, WordBoolean und LongBool.

Syntax

```
type
  Boolean = (False, True);
  WordBool = (False, True);
  LongBool = (False, True);
```

Diese Typen haben die folgenden Größen:
- Boolean hat Byte-Größe (8 Bits)
- tWordBool hat Word-Größe (16 Bits)
- LongBool hat Longint-Größe (32 Bits)

Da Boolesche Typen **ordinale** Aufzählungstypen sind, gelten die folgenden Relationen:

False < True
Ord(False) = 0
Ord(True) = 1
Succ(False) = True
Pred(True) = False

Der Typ Boolean wird am häufigsten verwendet, er benötigt weniger Speicher als die anderen Booleschen Typen.. WordBool und LongBool wurden hauptsächlich aufgenommen, um Kompatibilität zur Windows-Umgebung herzustellen.

In einem **Ausdruck** erzeugen diese **relationale Operatoren** Ergebnisse vom Typ Boolean:

```
=    <>    >    <    >=    <=    IN
```

case (reserviertes Wort)

Die Anweisung **case** besteht aus einem **Ausdruck** (dem Selektor) und einer Liste von **Anweisungen**, denen jeweils das reservierte Wort **case** voransteht.

Syntax

```
case expression of
  case: statement;
  ...
  case: statement;
end
```

oder

```
case expression of
    case: statement;
 ...
  case: statement;
else
  statement
end
```

Ein **case**-Ausdruck besteht aus einer oder mehreren Konstanten oder Bereichen, die durch Kommata voneinander getrennt sind.

else ist optional.

Beispiel:

```
case Ch of
  'A'..'Z', 'a'..'z':  WriteLn('Letter');
  '0'..'9':            WriteLn('Digit');
  '+', '-', '*', '/':  WriteLn('Operator');
else
  WriteLn('Special character');
end;
```

Char Typen

Variablen vom **ordinalen** Typ Char speichern ASCII Zeichen.

Zeichen-Konstanten werden in Apostrophe gesetzt:

```
'A', '3', oder '*'
```

Das Zeichen ' wird doppelt in einfachen Anführungszeichen geschrieben:

```
''''
```

Die Funktion **Chr** wandelt einen Integerwert in das entsprechende ASCII-Zeichen um.

Die Funktion **Ord** gibt den ASCII-Weret des Zeichens zurück.

Konstantendeklarationen (const: reserviertes Wort)

Eine Konstanten-Deklaration (**const**) definiert einen **Bezeichner**, der einen konstanten Wert hat.

Syntax

```
const
  identifier = expression;
  ...
  identifier = expression;
```

Ausdrücke in Konstanten-Deklarationen müssen so geformt sein, daß sie vom Compiler während der Compilierung ausgewertet werden können.

Im Unterschied zu Standard-Pascal sind in Turbo Pascal **konstante Ausdrücke** erlaubt.

Über die Deklaration von **typsisierten Konstanten** werden initialisierte Variablen deklariert.

Beispiel

```
(* Constant Declarations *)
const
  MaxData  = 1024 * 64 - 16;
  NumChars = Ord('Z') - Ord('A') + 1;
  Message  = 'Hello world...';
const
  identifier: type = value;
  ...
  identifier: type = value;
```

Konstante Ausdrücke

Ein konstanter Ausdruck ist ein Ausdruck, dessen Auswertung durch den Compiler nciht die Ausführung des Programms erfordert.

Da der Compiler in der Lage sein muß, den konstanten Ausdruck während der Compilierung vollständig auszuwerten, sind folgende Elemente in konstanten Ausdrücke sind nicht erlaubt:

- Referenzen auf Variablen und typisierte Konstanten, außer in **konstanten Adreßausdrücken**.
- Funktionsaufrufe
- der **Operator @** (außer in konstanten Adreßausdrücken)

Diese Standard-Funktionen können in konstanten Ausdrücken verwendet werden:

Abs
Chr
Hi
Length
Lo
Odd
Ord
Pred
Ptr
Round
SizeOf
Succ
Swap
Trunc

constructor (reserviertes Wort)

Ein **Konstruktor** ist eine besondere **Methode**, die ein Objekt mit virtuellen Methoden initialisiert.

Syntax

```
constructor Method (Param1, Param2 : Integer);
```

Ein Konstruktor muß mit dem reservierten Wort **constructor** deklariert werden**.**

Der Konstruktor initialisiert ein Objekt, indem er ein Link zwischen dem Objekt und der Tabelle virtueller Methoden einrichtet, die Adressen des Codes seiner virtuellen Methoden enthält.

Ein Konstruktor kann auch die Datenfelder des Objekts initialisieren.

Siehe auch

destructor
object
virtual

destructor (reserviertes Wort)

Ein **Destruktor** ist eine besondere **Methode**, die dynamische Objekte vom Heap entfernt.

Syntax

```
destructor method;
```

Der Destruktor muß mit dem reservierten Wort **destructor** deklariert werden.

Siehe auch

constructor
object
virtual

do (reserviertes Wort)

Das reservierte Wort **do** wird in den Anweisungen **while**, **for** und **with** verwendet.

Beispiel:

```
while Ch = ' ' do Ch := GetChar;
for Ch := 1 to 100 do Ch := GetChar;
with Date[I] do month := 1;
```

else (reserviertes Wort)

Das reservierte Wort **else** wird in der Anweisung **if** und **case** verwendet.

Beispiel:

```
(* using "if" statement *)
if (I < Min) oder (I > Max) then I := 0;

if ParamCount <> 2 then
begin
  WriteLn('Bad command line');
  Halt(1);
end
else
begin
  ReadFile(ParamStr(1));
  WriteFile(ParamStr(2));
end;


(* using "case" statement *)
case Ch of
  'A'..'Z', 'a'..'z':  WriteLn('Letter');
  '0'..'9':            WriteLn('Digit');
  '+', '-', '*', '/':  WriteLn('Operator');
else
  WriteLn('Special character');
end;
```

end (reserviertes Wort)

Das reservierte Wort **end** wird mit den folgenden Anweisungen verwendet:

- **begin**, zusammengesetzte Anweisungen
- **case**, case-Anweisungen
- **record**, Deklaration von Record-Typen
- **object**, Deklaration von Objekt-Typen
- **asm**, Aufruf des integrierten Assemblers

Beispiel:

```
(* with "begin" to form compound statement *)
if First < Last then
begin
  Temp := First;
  First := Last;
  Last := Temp;
end;


(* with "case" statement *)
case Ch of
  'A'..'Z', 'a'..'z':  WriteLn('Letter');
  '0'..'9':            WriteLn('Digit');
  '+', '-', '*', '/':  WriteLn('Operator');
else
  WriteLn('Special character');
end;


(* with record type definitions *)
type
  Class  = (Num, Dat, Str);
  Date   = record
             D, M, Y: Integer;
           end;
  Facts = record
            Name: string[10];
            case Kind: Class of
              Num: (N: real);
              Dat: (D: Date);
              Str: (S: string);
            end;


(* with object type definitions *)
type
  LocationPtr = ^Location;
  Location = object
    X, Y: Integer;
    procedure Init(PX, PY: Integer);
    function GetX: Integer;
    function GetY: Integer;
  end;


(* with asm *)
```

```
asm
  mov ax,1
  mov cx, 100
end;
```

export

Die Anweisung **export** macht eine Prozedur oder Funktion exportierbar, durch
- Erzwingen des Aufrufmodells **far**
- Generieren eines besonderen Prozedur-Eintritts- und Austritts-Codes.

Das wirkliche Exportieren einer Prozedur oder Funktion geschieht erst, wenn die Prozedur oder Funktion in einer **Exportklausel** einer Bibliothek aufgeführt ist.

Siehe auch

Dynamische Linkbibliotheken (DLLs)
DLLs verwenden
DLLs schneiden

exports (reserviertes Wort)

Die Klausel **exports** einer Bibliothek listet Prozeduren oder Funktionen auf, die von einer DLL exportiert werden.

Diese Klausel kann überall und beliebig oft im Deklarationsteil eines Programms oder einer Bibliothek stehen.

Jeder Eintrag der Klausel **exports** gibt den Bezeichner einer Prozedur oder Funktion an, die exportiert werden soll. Der **Bezeichner** muß eine Prozedur oder Funktion nennen, die mit dem **export**-Befehl compiliert wurde.

Eine **exports**-Klausel kann auch Folgendes enthalten:

- eine **index**-Klausel
- eine **name**-Klausel
- das reservierte Wort **resident**

Aufzählungstypen

Die Deklaration eines Aufzählungstypen definiert eine Menge von Konstanten, die mögliche Werte des Typs bilden.

Syntax

```
type
  name = (identifier,
    identifier, ...,
    identifier );
```

Die **Bezeichner** in der Typdefinition werden Konstanten des Aufzählungstyps.

Die erste Konstante hat den Ordinalwert 0, die zweite 1, und so weiter.

Aufzählungstypen sind eine Unterklasse der **ordinalen** Typen.

Beispiel:

```
type
  Suit = (Club, Diamond, Heart, Spade);
```

Mit dieser Deklaration ist Heart eine Konstante des Typs Suit.

Die Standard-Funktion **Ord** gibt den Ordinalwert einer Aufzählungstypen-Konstante zurück. In diesem Beispiel:

```
Ord(Club)     =  0
Ord(Diamond)  =  1
Ord(Heart)    =  2
```

und so weiter.

far (reserviertes Wort)

Die Anweisung **far** bewirkt die Vewendung von Far-Aufrufen, d.h. Aufrufe von Anweisungen aus anderen Codesegmenten.

Syntax

```
procedure Name; far;
```

Die im **interface**-Teil einer **Unit** deklarierten Prozeduren sind als **far** deklariert, damit sie von anderen Units aufgerufnen werden können.

Mit dem **Compilerbefehl $F** die Standardaufrufmethode des Compiler überschreiben werden. Alle Prozeduren und Funktionen, die unter Verwendung von in the **{$F+}** deklariert werden, sind ebenfalls far deklariert.

Siehe auch

near

Ausdrücke

Ausdrücke bestehen aus Operatoren und Operanden. Es folgen die Operanden:

- **Konstanten**
- **Variablen**
- **Funktionsaufrufe**
- **Mengen-Konstruktoren**

Teilausdrücke können geklammert werden, um die Bearbeitungsreihenfolge zu ändern.

Siehe auch

- **Rangfolge der Operatoren**

Objekttypen (object: reserviertes Wort)

Ein **Objekt** ist eine Datenstruktur, die eine bestimmte Anzahl von Komponenten enthält.

Syntax

```
object
  field;
  field;
  ...
  method;
  method;
end;
```

Die Komponenten von Objekten sind entweder ein Feld (das Daten eines bestimmten Datentyp enthält) oder eine Methode, die eine Operation auf dem Objekt ausführt.

Die Deklaration eines Feldes setzt sich zusammen aus einem Bezeichner (der das Feld benennt) und dem Datentyp.
Die Deklaration einer **Methode** enthält die Kopfzeile einer **Prozedur**, **Funktion**, eines **Konstruktors** oder **Destruktor**.

```
Feld      = FieldName(s): type;
Methode   = procedure MethodName(<parameter(s)>:type);
     oder = function MethodName(<parameter(s)>:type):type;
     ode= constructor MethodName(<parameter(s)>: type [;<parameter(s)>:
            type]); [virtual];
     oder   = destructor MethodName[(<parameters>: type)];[virtual];
```

Ein Objekttyp kann Komponenten von anderen Objekttypen erben.
The domain of an object type consists of itself and all its descendants.

Siehe auch

Objekttypenkonstanten

private

Private ist nur innerhalb von **Objekten** ein reserviertes Wort.

external

External-Deklarationen erlauben Zugriff auf getrennt compilierte in Assembler geschriebene **Prozeduren** und **Funktionen**.

Der external-Code wird mit Pascal-Unit oder -Programm mit der Compiler-Direktive **$L dateiname** verknüpft.

Beispiel:

```
function GetMode:       Word;  external;
procedure SetMode(Mode: Word); external; $L CURSOR.OBJ
```

file (reserviertes Wort)

Ein **file**-Typ besteht aus einer linearen Folge von Komponenten von **type**, die alle Typen außer dem **file**-Typ haben können.

Syntax

```
file of type
  oder
file
```

Werden das Wort **of** und der Komponententyp nicht angegeben, gibt der Typ eine untypisierte Datei an.

Der vordefinierte **file**-Typ Text steht für eine Datei, die Zeichen in Zeilen enthält.

Beispiel:

```
(* File type Deklarationen *)
type
  Person = record
     FirstName: string[15];
     LastName : string[25];
     Address  : string[35];
  end;
  PersonFile = file of Person;
  NumberFile = file of Integer;
  SwapFile = file;
```

for...to, for...downto (reserviertes Wörter)

Die Anweisung **for** bewirkt, daß die **Anweisungen** nach **do** für jeden Wert im Bereich first bis last einmal ausgeführt werden.

Syntax

```
for var := first to last do
  statement
```

oder

```
for var := first downto last do
  statement
```

Die Steuer-**Variable** und Anfangs- wie Endwerte müssen einen **ordinalen**-Typ haben.

TO

Mit **to** wird der Wert der Steuervariablen bei jedem Schleifendurchlauf um 1 erhöht.

DOWNTO

Mit **downto** wird der Wert der Steuervariablen um jeweils 1 dekrementiert.

Beispiel:

```
(* for ... to, for ... downto *)
for I := 1 to ParamCount do
  WriteLn(ParamStr(I);

for I := 1 to 10 do
  for J := 1 to 10 do
  begin
    X := 0;
    for K := 1 to 10 do
      X := X + Mat1[I, K] * Mat2[K, J];
    Mat[I, J] := X;
  end;
```

forward

Mit einer **forward**-Deklaration können Sie eine **Prozedur** oder **Funktion** deklarieren, wobei der Anweisungsblock erst zu einem späteren Zeitpunkt definiert werden muß.

Zwischen der **forward**-Deklaration und der Definition der Prozedur ist die Deklaration anderer Prozeduren und Funktionen möglich. Auf diese Weise ist der sich überkreuzende Bezug zwischen mehreren Prozeduren herstellbar.

Nach der **forward**-Deklaration muß die Prozedur oder Funktion durch eine Deklaration des Anweisungsblock definiert werden.

Diese Deklaration kann die Parameterliste des Prozedur- oder Funktionkopfs entbehren.

Beispiel:

```
(* Forward-Prozedur *)
procedure Flip(N: Integer); forward;

procedure Flop(N: Integer);
begin
  WriteLn('Flop');
  if N > 0 then Flip(N - 1);
end;

procedure Flip;
begin
  WriteLn('Flip');
  if N > 0 then Flop(N - 1);
end;
```

function (reserviertes Wort)

Eine **Funktion** definiert einen Programmteil, der einen Wert berechnet und als Funktionsergebnis zurückgibt.

Syntax

```
function ident : type;
  oder
function ident (parameters) : type;
```

Der **Funktionskopf** legt den **Bezeichner**, die formalen **Parameter** (soweit vorhanden) und den **Ergebnistyp** der Funktion fest.

Gültige Ergebnistypen sind **ordinal**, **real**, **string** und **Zeiger-** -Typen.

Eine Funktion wird durch einen Funktionsaufruf in einem **Ausdruck** aktiviert.

Dem **Funktionskopf** folgt
- der Deklarationsteil, der lokale Objekte deklariert
- der Anweisungsteil, der die beim Aufruf auszuführenden Anweisungen spezifiziert.

Der Anweisungsteil sollte zumindest eine Zuweisung enthalten, die dem Funktionsbezeichner einen Wert gibt; das Funktionsergebnis ist der letzte zugewiesene Wert.

Anstatt Deklaration und Anweisungsteil kann eine **Funktion** **forward**, **external**, **far** oder **inline** deklariert werden.

Beispiel:

```
(* Funktions-Deklaration *)
function UpCaseStr(S: string): string;
var
  I: Integer;
begin
  for I := 1 to Length(S) do
  if (S[I] >= 'a') and (S[I] <= 'z') then
  Dec(S[I], 32);
  UpCaseStr := S;
end;
```

goto (reserviertes Wort)

goto übergibt die Programmausführung den **Anweisungen** mit dem angesprungenen **Label**.

Syntax

```
goto label
```

Das Label muß im selben Block wie die **goto**-Anweisung sein. Sie kann nicht aus einer Prozedur oder Funktion hinausspringen.

Beispiel:

```
label 1, 2;
goto 1
    .
    .
    .
1: WriteLn ('Abnormal program termination');
2: WriteLn ('Normal program termination');
```

Bezeichner

Bezeichner können für folgende Elemente stehen:

Funktionen
Labels
Konstanten
Prozeduren
Programme
Recordfelder
Typen
Units
Variablen

Bezeichner können beliebig lang sein, es sind aber nur die ersten 63 Zeichen signifikant.

- Das erste Zeichen eines Bezeichners muß ein Buchstabe sein
- die folgenden Zeichen müssen Ziffern, Buchstaben oder Unterstriche sein.

Wie **reservierte Wörter** können Bezeichner groß oder klein geschrieben werden.

Qualifizierte Bezeichner

Wenn verschiedene Instanzen desselben Bezeichners existieren, können Sie den Bezeichner durch einen **Unit**-Bezeichner unterscheiden, um eine bestimmte Instanz anzusprechen.

Ein so kombinierter Bezeichner wird qualifizierter Bezeichner genannt.

Beispiel:

```
WriteLn
Exit
Real2String

(* Qualifizierter Bezeichner *)
System.MemAvail  (* unit = System, identifier = MemAvail *)
Dos.Exec         (* unit = Dos, identifier = Exec *)
Crt.Window       (* unit = Crt, identifier = Window *)
```

if...then...else (reserviertes Wörter)

Geben die Bedingungen an, unter denen eine Anweisung ausgeführt wird.

Syntax

```
if expr then statement
```
oder
```
if expr then statement else statement
```

Ist der Boolesche **Ausdruck** nach **if** True, werden **Anweisungen** nach **then** ausgeführt.

Gibt es einen **else**-Teil, werden ansonsten die Anweisungen nach **else** ausgeführt.

Beispiel:

```
(* "if" Anweisungen *)
if (I < Min) oder (I > Max) then I := 0;

if ParamCount <> 2 then
begin
  WriteLn('Bad command line');
  Halt(1);
end
else
begin
  ReadFile(ParamStr(1));
  WriteFile(ParamStr(2));
end;
```

implementation (reserviertes Wort)

Der **implementation**-Teil einer Unit ist der **private** Teil. Deklarationen in diesem Teil können nur in diesem Abschnitt der Unit verwendet werden.

Alle Deklarationen im **Interface**-Teil sind in der Implementation sichtbar.

Die Implementation kann zusätzliche eigene Deklarationen enthalten, die für Programme, welche die Unit verwenden, unsichtbar sind.

Eine **uses**-Klausel kann in der Implementation unmittelbar nach dem reservierten Wort **implementation** folgen.

Sind Prozeduren **external** deklariert, sollten ein oder mehrere **$L dateiname**-Befehle im Programmm vor dem letzten **end** der Unit stehen.

Die normalen (nicht inline) Prozeduren und Funktionen, die im Interface deklariert sind, müssen in der Implementation wieder erscheinen.

Der **Prozedur**/**Funktion**-Kopf in der Implementation sollte entweder
- identisch mit der Deklaration im Interface oder
- aus einer **Kopfzeile** bestehen.

Kopfzeilen (Prozeduren und Funktionen)

Als Kopfzeile einer Prozedur oder Funktion wird hier eine einzeilige Deklaration bezeichnet, die das reservierte Wort (**procedure** oder f**unction**) und den Namen (Bezeichner) der Routine enthält

Die bezeichnete Routine enthält die lokalen Deklarationen (Labels, Konstanten, Typen, Variablen und verschachtelten Prozeduren und Funktionen) und den Rumpf der Prozedur oder Funktion.

Lokale Routinen der **Implementation** (die nicht im Interface-Abschnitt deklariert sind) müssen mit einem vollständigen Prozedur/Funktionskopf angegeben werden.

index

Eine **Index-Klausel** besteht aus dem Wort Index gefolgt von einer Integer-Konstante zwischen 1 und 32767.
Eine **Index-Klausel** ist in einer **exports**-Klausel enthalten.

Wenn eine **Index-Klausel** verwendet wird, benützt die zu exportierende Prozedur oder Funktion diese Zahl. Ohne **Index-Klausel** wird eine Ordinalzahl zugewiesen.

Beispiel

```
procedure ImportByOrdinal; external 'TESTLIB' index 5;
```

Siehe auch

DLLs verwenden

inline (reserviertes Wort)

Inline-Anweisungen fügen kurze Assembler-Programmteile direkt in ein Programm ein.

Syntax

```
inline ( data / data / ... data )
```

Als **Anweisungen** werden die **inline**-Daten direkt in das Programm eingefügt.

Als Anweisung in einer **Prozedur** oder **Funktion**-Deklaration werden die **inline**-Datenelemente bei jedem Aufruf der Prozedur oder Funktion eingefügt.

Ein **inline**-Datenelement besteht aus einer oder mehreren Konstanten und/oder den Adressen der angegeben Bezeichner, denen optional eine Größenangabe < oder > voransteht.

Ein Variablenbezeichner kann von + (plus) oder - (minus) und einer Konstanten gefolgt werden, die einen Offset von der Adresse der Variablen angibt.

Ein **inline**-Datenelement erzeugt 1 Byte Code, wenn es eine Konstante im Bereich von 0..255 ist; sonst ein Word.

Mit den Operatoren < und > überschreiben Sie die automatische Größenwahl:

- < Bedeutet immer Generierung eines Byte
- > Bedeutet immer Generierung eines Word

Beispiel:

```
(* "inline" statement *)
procedure FillWord(var Dest; Count: Word;
  Data: Word);
begin
  inline(
    $C4/$7E/<Dest/     (* LES   DI,Dest[BP] *)
    $8B/$4E/<Count/    (* MOV   CX,Count[BP]*)
    $8B/$46/<Data/     (* MOV   AX,Data[BP] *)
    $FC/               (* CLD               *)
    $F3/$AB);          (* REP   STOSW       *)
end;
```

Siehe auch

Der integerierte Assembler

Integer-Typen

Turbo Pascal enthält fünf vordefinierte Integer-Typen. Jeder Typ steht für eine Untermenge ganzer Zahlen:

Typ	Bereich	Größe
Shortint	-128..127	8 Bit
Integer	-32768..32767	16 Bit
Longint	-2147483648..2147483647	32 Bit
Byte	0..255	8 Bit
Word	0..65535	16 Bit

Alle Integertypen sind **ordinale** Typen.

interface (reserviertes Wort)

Der **Interface-Teil** einer Unit ist ihr öffentlicher (**public)** Teil.

Er bestimmt, was für Programme oder andere Units sichtbar und zugänglich ist.

Der Interface-Teil beginnt mit dem reservierten Wort **interface**, der nach dem Unit-Vorspann steht, und endet mit dem reservierten Wort **implementation**.

Im Interface-Teil werden Konstanten, Datentypen, Variablen, Prozeduren und Funktionen deklariert.

Die "Anweisungsrümpfe" der public-Prozeduren und Funktionen stehen im Abschnitt implementation.

Forward-Deklarationen sind hier nicht erlaubt.

Eine **uses**-Klausel ist im Interface-Teil zuilässig.
Falls vorhanden, muß **uses** unmittelbar nach dem reservierten Wort **implementation** folgen.

interrupt

Die Anweisung **interrupt** dient zur Deklaration von Interrupt**-Prozeduren**.

Syntax:

```
procedure IntProc(Flags, CS, IP, AX, BX,
  CX, DX, SI, DI, DS, ES, BP: Word);
interrupt;
```

Die Register werden als Pseudo-Parameter übergeben, damit sie im Code der Interrupt-Prozedur änderbar sind.

label (reserviertes Wort)

Eine **label**-Deklaration deklariert Labels, die **Anweisungen** im Anweisungsteil markieren.

Syntax

```
label identifier, ... identifier;
```

Jedes Label darf nur eine Anweisung markieren.

Außer einem **Bezeichner** kann eine Zahl zwischen 0 und 9999 als Label verwendet werden.

Ein Label wird mit einer **goto**-Anweisung angesprungen.

Beispiel:

```
label 1, 2;
goto 1
   .
   .
   .
1:  WriteLn ('Abnormal program termination');
2:  WriteLn ('Normal program termination');
```

library (reserviertes Wort)

Eine dynamische **Linkbibliothek (DLL)** beginnt mit einer **library**-Kopfzeile. Diese Kopfzeile weist Turbo Pascal an, eine ausführbare Datei mit der Kennzeichnung .DLL anstatt .EXE zu erzeugen.

Siehe auch

export
exports
name
Importunits
index

MaxInt und MaxLongInt (Konstanten)

MaxInt und MaxLongInt sind vordefinierte Konstanten.

- MaxInt enthält den größten **Integer** (32,767)
- MaxLongint enthält den größten **Longint** (2,147,483,647)

Methoden

Eine Methode ist eine Prozedur oder Funktion, die innerhalb einer **object**-Typ-Deklaration deklariert wurde.

Syntax

```
procedure Method(Param1, Param2 : Integer);
```

Methoden können auf die Datenfelder des Objekts zugreifen, ohne sie als Parameter übergeben zu erhalten.

Die Deklaration innerhalb der Objekttyp-Deklaration besteht nur aus der Kopfzeile.

Der Körper der Methode wird außerhalb der Objekttyp-Deklaration definiert.

Die Kopfzeile muß den Namen des Objekttyps enthalten:

```
procedure ObjectType.Method(Param1, Param2: Integer);
begin
...
end;   (* Method *)
```

Methoden können statisch, **virtuell** oder **dynamisch** sein. Zusätzlich zu regulären Prozeduren und Funktionen enthält das objektorientierte Pascal zwei besondere Methodentypen: **constructor** und **destructor.**

Siehe auch

Self

Dynamische Methoden

Eine Methode, deren Aufruf zur Laufzeit in den Code eingebunden wird, wird dynamische Methode genannt.

Im Unterschied zu einer **virtuellen** Methode enthält die Deklaration einer dynamischen Methode einen dynamischen Methodenindex.

Syntax

```
procedure MethodName (<parameter list>); virtual DynamicMethodIndex
```

DynamicMethodIndex ist ein Integerwert, eine Integer-Konstante oder ein Ausdruck, der zu einem Integerwert ausgewertet wird.

Wie virtuelle Methoden erlauben dynamische Methoden, daß derselbe Name in verschiedenen Ebenen der Objekthierarchie unterschiedlich implementiert wird. Wenn Sie mit Hilfe von vielen virtuellen Methoden Typen von anderen Typen ableiten, kann durch die Verwendung dynamischer Methoden unter Umständen Speicher sparen.

name

Eine Namensklausel ist in der Klausel **exports** eingeschlossen. Sie besteht aus dem Wort **name**, gefolgt von einer Stringkonstante.

Bei Vorhandensein einer Namensklausel verwendet die Prozedur oder Funktion, die exportiert werden soll, den Namen in der Stringkonstante.

Wird keine Namensklausel verwendet, wird die Prozedur oder Funktion über ihren Bezeichner exportiert und in Großbuchstaben verwandelt.

near (reserviertes Wort)

Die Anweisung **near** bewirkt die Verwendung der Aufrufmethode near. In einem Near-Aufruf wird die Steuerung an eine andere Position innerhalb desselben Codesegments übergeben.

Syntax

```
procedure Name; near;
```

Prozedureen, die in einem Programm oder im **Implementationsteil** einer Unit deklariert werden, sind als near deklariert

Siehe auch

far

nil (reserviertes Wort)

Der reservierte Begriff **nil** benennt eine **Zeiger**-Typkonstante, die auf nichts zeigt.

nil ist zu allen Zeigertypen kompatibel.

of (reserviertes Wort)

Der reservierte Begriff **of** wird in **array**-, **set**- und **file**-Deklarationen und in **case** Anweisungen verwendet.

Syntax

```
identifier = array[X..Y] of type
identifier = set of type
identifier = file of type
case expression of <case statement list>
```

Beispiel:

```
(* array Deklaration *)
type
  IntList  = array[1..100]     of Integer;
  CharData = array['A'..'Z']   of Byte;
  Matrix   = array[0..9, 0..9] of real;


(* Set-Typen *)
type
  Day = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
  CharSet = set of Char;
  Digits = set of 0..9;
  Days = set of Day;


(* File type Deklarationen *)
type
  Person = record
     FirstName: string[15];
     LastName : string[25];
     Address  : string[35];
  end;
  PersonFile = file of Person;
  NumberFile = file of Integer;
  SwapFile = file;


(* case statement *)
case Ch of
  'A'..'Z', 'a'..'z':  WriteLn('Letter');
  '0'..'9':            WriteLn('Digit');
  '+', '-', '*', '/':  WriteLn('Operator');
else
  WriteLn('Special character');
end;
```

Ordinal-Typen

Turbo Pascal enthält neun vordefinierte ordinale **Typen**:

Fünf dieser ordinalen **Integer**-Typen sind eine bestimmte Untermenge ganzer Zahlen:

Typ	Bereich	Größe
Shortint	128..127	8 Bit
Integer	-32768..32767	16 Bit
Longint	-2147483648..2147483647	32 Bit
Byte	0.255	8 Bit
Word	0.65535	16 Bit

Die anderen vier vordefinierten Ordinaltypen sind **Boolean**, **WordBool**, **LongBool)**und **Char**.

Zwei weitere Klassen von benutzerdefinierten Ordinaltypen sind **Aufzählungstypen** und **Teilbereichstypen**.

Folgende Standard-Funktionen können mit allen Ordinaltypen verwendet werden:

Ord
Pred
Succ

packed (reserviertes Wort)

Der reservierte Begriff **packed** kann einer **array** -Typdeklaration voranstehen.

Er hat jedoch in Turbo Pascal keine Auswirkung, weil Datenkompression automatisch erfolgt.

Parameterliste

Der Kopf einer **Prozedur** oder **Funktion** kann eine formale Parameterliste enthalten, wie beispielswiese: `( params; params; ... params )`

Parametergruppen können wie folgt geformt sein:

```
identifier,     identifier: type (* list of value parameters    *)
var identifier, identifier: type (* list of variable parameters *)
var identifier, identifier       (* list of untyped variable parameters
*)
```

Zeiger-Variable (pointer: reserviertes Wort)

Eine Zeiger-Variable enthält die Adresse einer dynamischen Variablen eines bestimmten **Basistyps**.

Sie können einer Zeiger-Variablen wie folgt einen Wert zuweisen:
- mit dernProzeduren **New** oder **GetMem**
- mit dem **@-Operator**
- mit der **Ptr**-Funktion

Der reservierte Begriff **nil** benennt eine Zeigerkonstante, die auf nichts zeigt.

Pointer

Der vordefinierte Zeigertyp **pointer** steht für einen untypisierten Zeiger (einen Zeiger, der auf keinen bestimmten Typ zeigt).

PChar

Der vordefiniete Typ PChar steht für einen Zeiger auf einen **null-termininierten String**.

PChar ist deklariert asl

```
type PChar = ^Char;
```

Turbo Pascal für Windows unterstützt erweiterte Syntaxregeln (ansprechbar über den **$X Compilerbefehl**), die durch Einsatz des Typs PChar die Stringverwaltung erleichtern.

Beispiel:

```
{ Zeiger-Typdeklaration }
  type
    BytePtr  = ^Byte;
    WordPtr  = ^Word;
    IdentPtr = ^IdentRec;
    IdentRec = record
                 Ident: string[15];
                 RefCount: Word;
                 Next: IdentPtr;
               end;
```

Siehe auch

Zeigertypen-Konstanten

resident

Der Begriff **resident** wird in der Klausel **exports** eingeschlossen. Wenn **resident** verwendet wird, bleibt die Exportinformation im Speicher, wenn die DLL geladen wird.

Die Option resident verkürzt die Ziet, die Windows benötigt, einen DLL-Eintrag nach Namen zu suchen.

Wenn Client-Programme, die die DLL benutzen, Einträge mit Namen importieren, sollten sie mit dem Begriff **resident** exportiert werden.

procedure (reserviertes Wort)

Eine **Prozedur** ist ein Programmteil, der eine bestimmte Aufgabe ausführt, oft auf der Grundlage einer Gruppe von Parametern.

Syntax

```
  procedure identifier;
```

oder

```
  procedure identifier ( parameters );
```

Im Prozedurkopf werden der **Bezeichner** der Prozedur und die formalen **Parameter** (falls vorhanden) angegeben.

Eine **Prozedur** wird von einer Prozedur-**Anweisung** aktiviert.

Dem Prozedurkopf folgt:

- eine Deklaration, die lokale Objekte deklariert
- Anweisungen zwischen **begin** und **end**, die beim Aufruf der Prozedur auszuführende Aktionen enthalten.

Mit der Anweisung **interrupt** können Sie Interrupt-Prozeduren deklarieren.

Statt Deklarations- und Anweisungsteile kann eine Prozedur-Deklaration eine **forward**, **external** oder **inline** Anweisung enthalten.

Beispiel:

```
  { Procedure Deklaration }
 procedure WrStr(X, Y: integer; S: string);
  var
    SaveX, SaveY: Integer;
  begin
    SaveX := WhereX;
    SaveY := WhereY;
    GotoXY(X, Y);
    Write(S);
    GotoXY(SaveX, SaveY);
  end;
```

Siehe auch

prozedurale Typenkonstanten

program (reserviertes Wort)

Ein Turbo-Pascal-Programm hat folgende allgemeine Struktur:

```
program       ... ;        { Programmkopf }
uses          ... ;        { Uses-Klausel }
label         ... ;        { Labels }
const         ... ;        { Konstanten }
type          ... ;        { Typen }
var           ... ;        { Variablen}
procedure     ... ;        { Prozeduren }
function      ... ;        { Funktionen }
begin
   statement;              { Anweisungen }
    ...
end.
```

- Im Programmkopf stehen Name und Parameter des Programms; der Programmkopf hat im Programm selbstwirkt sich auf das Programm nicht aus.

- Die Klausel **uses** enthält die Units, die das Programm verwendet.

- **Label**, **Konstanten**, **Variablen**, **Prozedur**- und **Funktion**-Deklaration können in beliebiger Reihenfolge aufgeführt und wiederholt werden.

- Der Anweisungteil gibt die beim Ablauf des Programms auszuführenden Anweisungen an.

Realtypen

Jedem Realtyp ist eine Wertemenge zugeordnet, die eine Teilmenge der Realzahlen darstellt und in Gleitkommanotation mit einer festen Anzahl von Dezimalstellen dargestellt werden kann.

Turbo Pascal enthält fünf vordefinierte **Realtypen**. Jeder Typ hat einen bestimmten Bereich und Genauigkeit:

Typ	Bereich	Stellen	Bytes	
real	2.9e-39..1.7e38	11-12	6	
single	1.5e-45..3.4e38	7-8	4	
double	5.0e-324..1.7e308	15-16	8	
extended	3.4e-4932..1.1e4932	19-20	10	
comp	-9.2e18..9.2e18	19-20	8	

Anmerkung: comp ist ein 64 Bit großen Integer.

Turbo Pascal unterstützt zwei Arten der Gleitkommagenerierung:

- **Software-Gleitkomma** `{$N-}`
- **8087 Gleitkomma** `{$N+}`

Mit dem Compilerbefehl **$N** schalten sie zwischen diesen Modellen um.

record (reserviertes Wort)

Ein **Record** enthält verschiedene Komponenten oder Felder, die aus verschiedenen Typen bestehen können.

Syntax

```
record
  fields;
  fields;
  ...
  fields
end;
```

oder

```
record
  fields;
  ...
  case tag: type of
  case: ( fields );
  ...
  case: ( fields )
end;
```

Jede Feldliste ist eine durch Kommata unterteilte Liste von **Bezeichnern**, denen ein Strichpunkt und die Angabe eines **Typs** folgt.

Beispiel:

```
{ Record Type Definitions }
type
  Class  = (Num, Dat, Str);
  Date   = record
     D, M, Y: Integer;
  end;
  Facts = record
    Name: string[10];
    case Kind: Class of
      Num: (N: real);
      Dat: (D: Date);
      Str: (S: string);
  end;
```

Siehe auch

Recordtyp-Konstanten

repeat..until (reservierte Wörter)

Anweisungen zwischen **repeat** und **until** werden wiederholt, bis der **Boolesche Ausdruck** True ist.

Syntax

```
repeat
  statement;
  statement;
  ...
  statement
until expression
```

Die Sequenz wird mindestens einmal ausgeführt.

Beispiel:

```
{ Repeat Anweisungen }
  repeat Ch := GetChar until Ch <> ' ';

  repeat
    Write('Enter value: ');
    ReadLn(I);
  until (I >= 0) and (I <= '9');
```

Self

Self ist ein impliziter Parameter, der immer weitergegeben wird, wenn eine **Methode** aufgerufen wird.

Self ist ein 32 Bit-Zeiger auf die Instanz, die die Methode aufrief.

Er garantiert unter anderem, daß die korrekten **virtuellen** Methoden für diese bestimmte Objektinstanz aufgerufen werden.

set (reserviertes Wort)

Syntax

```
set of type
```

Der Basistyp eines Mengentypen muß ein **ordinaler** Typ mit nicht mehr als 256 möglichen Werten sein.

Die Ordinalwerte der oberen und unteren Grenzen des Basistyps müssen im Bereich 0..255 liegen.

Ein Mengen-Konstruktor wird durch **Ausdrücke** in Klammern hergestellt. Jeder Ausdruck benennt einen Wert der Menge.

Die Notation `[ ]` steht für eine leere Menge, die zu allen Mengen-Typen kompatibel ist.

Beispiel:

```
{ Mengen-Typen }
  type
    Day = (Sun, Mon, Tue, Wed, Thu, Fri, Sat);
    CharSet = set of Char;
    Digits = set of 0..9;
    Days = set of Day;

  { Set constructors }

  ['0'..'9', 'A'..'Z', 'a'..'z', '_']
  [1, 5, I + 1 .. J - 1]
  [Mon..Fri]
```

Siehe auch

Mengentypen-Konstanten

Anweisungen

Unter einer Anweisung wird eines der folgenden Konstrukte verstanden:

Zuweisung (:=)
begin..end
case..of..else..end
for..to/downto..do
goto
if..then..else
inline(...)
Prozeduraufruf
repeat..until
while..do
with..do

String-Typen (string: reserviertes Wort)

Eine Variable vom Typ **string** ist eine Folge von **Zeichen** mit dynamischer Länge und einer konstanten Maximalgröße zwischen 1 und 255.

Syntax

```
string [ constant ]
  oder
string
```

Ein **string** ohne Maximal-Deklaration erhält die Größe 255.

Stringkonstanten werden in Anführungszeichen geschrieben:

```
- 'Turbo'
- 'That''s all'
```

Zwei einfache Anführungszeichen zeigen eines im String an.

Die folgenden **Operatoren** können mit Werten vom Typ String verwendet werden:

+ = <> < > <= >=

Die Standard-Funktion **Length** gibt die dynamische Länge eines String zurück.

Beispiel:

```
{ String Type Definitions }
  const
    LineLen = 79;
  type
    Name = string[25];
    Line = string[LineLen];
```

Siehe auch

String-Typenkonstanten

Teilbereichstypen

Ein Teilbereichstyp umfaßt einen **ordinalen** Wertebereich, der in diesem Zusammenhang als Grundmenge bezeichnet wird.

Syntax

```
constant .. constant
```

Die Definition eines Teilbereichstypen enthält den niedrigsten und höchsten Wert.

Beide Konstanten müssen von selben Ordinaltyp sein, und die erste muß kleiner oder gleich der zweiten sein.

Die Compileranweisung **$R** steuert die Bereichsüberprüfung von Teilbereichstypen.

Beispiel:

```
{ Subranges }
 0..99
 -128..127
```

type (reserviertes Wort)

Eine **type**-Deklaration spezifiziert einen **Bezeichner**, der seinerseits für einen bestimmten Typ steht.

Syntax

```
type
  identifier = AnyType;
  ...
  identifier = AnyType;
```

Typen lassen sich in neun Klassen unterteilen:

- **array**
- **file**
- **object**
- **ordinal**
- **pointer**
- **real**
- **record**
- **set**
- **string**

unit (reserviertes Wort)

Units bilden die Grundlage der modularen Programmierung mit Turbo Pascal.

Mit Units bilden Sie Bibliotheken und unterteilen große Programme in logisch bezogene Module.
Eine Unit setzt sich aus folgenden Komponenten zusammen:
- **unit**-Kopf
- **interface**-Teil
- **implementation**-Teil
- Initialisierungsteil

Syntax

```
unit identifier     ...;   { Name }
interface           ...;   { Programmkopf }
uses                ...;   { Uses-Klausel }
label               ...;   { Labels }
const               ...;   { Konstanten }
type                ...;   { Typen }
var                 ...;   { Variablen }
procedure           ...;   { Prozeduren }
function            ...;   { Funktionen }

implementation             { Private Symbole }
interface           ...;   { Programmkopf }
uses                ...;   { Uses-Klausel }
label               ...;   { Labels }
const               ...;   { Konstanten }
type                ...;   { Typen }
var                 ...;   { Variablen }
procedure           ...;   { Prozeduren }
function            ...;   { Funktionen }

    begin                  { Initialisierung }
      statement;           { Anweisungen }
      ...
      statement
    end.
```

Unit-Kopf

Der **unit**-Kopf benennt die Unit. Dieser Name wird bei Bezug auf die Unit in einer uses-Klausel benützt.

Interface

Der **interface-Teil** deklariert Bezeichner public (für die Verwender der Unit zugänglich). Prozeduren und Funktionen werden nur als Kopfzeile im Interface-Teil aufgeführt.

Implementation

Der **implementation-Teil** definiert den Rumpf aller public Prozeduren und Funktionen. In ihm werden auch private Bezeichner deklariert, die für Verwender der Unit nicht zugänglich sind.

Der Initialisierungs-Teil ist der Letzte Teil einer Unit. In ihm steht
- der reservierte Begriff **end** (keine Initialisierung) oder
- ein Anweisungsteil, der zur Initialisierung der Unit ausgeführt wird.

uses

Jeder **Bezeichner** einer **uses**-Klausel benennt eine **Unit**, die vom Programm oder einer anderen Unit verwendet wird.

Syntax

```
uses identifier, ... identifier;
```

Der Compiler sucht eine Unit, die in einer **uses**-Klausel genannt wird, zuerst unter den residenten Units, die von TURBO.TPL geladen wurden.

Wenn der Compiler dort die Unit (UnitName) nicht findet, sucht er auf der Festplatte unter UNITNAME.TPU.

Der Compiler sucht im aktuellen Verzeichnis, danach in denen in Options|Directories| **Unit Directories (Verzeichnisse)** genannten.

Beispiel:

```
program MyProgram;
uses WinDos, WinCrt;    { program can now use }
                        { routines in WinDos und WinCrt }
```

var

Eine Variablen-Deklaration (**var**) assoziiert einen **Bezeichner** und einen **Typ** mit einer Speicheradresse, an der Werte dieses Typs gespeichert werden können.

Syntax

```
var
  identifier, ... identifier: type;
  ...
  identifier, ... identifier: type;
```

Mit der **absolute**-Klausel wird eine absolute Speicheradresse benannt.

Das reservierte Wort **var** wird auch zur Deklaration von Variablen-**Parametern** verwendet.

Beispiel:

```
{ Variablen-Deklarationen }
  var
    X, Y, Z: real;
    I, J, K: Integer;
    Done, Error: Boolean;
    Vector: array[1..10] of real;
    Name: string[15];
    InFile, OutFile: Text;
    Letters: set of 'A'..'Z';
```

virtual

Eine **virtuelle Methode** wird zur Laufzeit in den zugehörigen Code, diesen Prozeß nennt man dynamische Bindung.

Syntax

```
procedure Method(<parameter list>); virtual;
```

Siehe auch

Dynamische Methoden
Self

while (reserviertes Wort)

Eine **while**-Anweisung enthält einen Ausdruck, der die wiederholte Ausführung einer Anweisung steuert (die eine zusammengesetzte Anweisung sein darf).

Syntax

```
while expression do statement
```

Anweisungen nach **do** werden solange wiederholt ausgeführt, wie der **Boolesche Ausdruck** True ist.

Der Ausdruck wird ausgewertet, bevor die Anweisung ausgeführt wird. Wenn der Ausdruck False ist, wird die Anweisung nicht ausgeführt.

Beispiel:

```
{ "while"-Anweisungen }
  while Ch = ' ' do Ch := GetChar;

while not Eof(InFile) do
begin
  ReadLn(InFile, Line);
  WriteLn(OutFile, Line);
  Inc(LineCount);
end;
```

with (reserviertes Wort)

Die **with**-Anweisung ist ein Kürzel für den Bezug auf Felder eines **Records**.

Syntax:

```
with var, var, ... var do statement
```

Innerhalb der **Anweisungen** nach **do** kann auf die Felder einer oder mehrerer Record-**Variablen** Bezug genommen werden, wobei nur die Angabe der Feldbezeichner nötig ist.

Beispiel:

```
{ "with" statement }
  with Date[I] do
  begin
    month := 1;
    year  := year + 1;
  end;
```

Dies entspricht

```
Date[I].month := 1;
Date[I].year  := Date[I].year + 1;
```

Rangfolge der Operatoren

Operator	Priorität	Kategorie
@ not	**1** (höchste)	Unäre Operatoren
*** / div mod and shl shr**	**2**	Multipliplizierende Operatoren
+ - or xor	**3**	Addierende Operatoren
= <> < > <= >= in	**4** (niedrigste)	Relationale Operatoren

Auswertungsregeln

1) Ein Operand zwischen zwei Operatoren unterschiedlicher Priorität wird an den Operator höherer Priorität gebunden.
2) Ein Operand zwischen Operatoren gleicher Priorität wird an den Operator gebunden, der links von ihm steht.
3) Ausdrücke in Klammern werden zuerst ausgewertet und der gesamte Ausdruck wird als ein einzelner Operand behandelt.

Operationen gleicher Priorität werden normalerweise von links nach rechts ausgeführt, allerdings ordnet der Compiler unter Umständen Operatoren neu an, um die Codegenerierung zu optimieren.

Siehe auch

Binäre arithmetische Operatoren
Unäre arithmetische Operatoren
Boolesche Operatoren
Logische Operatoren
PChar Operatoren
Relationale Operatoren
Mengen-Operatoren
String-Opeatoren
@-Operator

Binäre arithmetische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
+	Addition	Integer	Integer
		Real	Real
-	Subtraktion	Integer	Integer
		Real	Real
*	Multiplikation	Integer	Integer
		Real	Real
/	Division	Integer	Real
		Real	Real
div	Integer-Division	Integer	Integer
mod	Divisionsrest	Integer	Integer

Hinweis

Der Operator **+** wird auch als **String-Operator** oder **Mengen-Operator,**

Die Operatoren **+**, **-** und * werden auch als **Mngen-Operatoren,**

Unäre arithmetische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
+	Identität	Integer	Integer
		Real	Real
-	Negation	Integer	Integer
		Real	Real

Hinweis

Operanden, die **Teilbereiche** **ordinaler** Typen darstellen, werden auf dieselbe Weise wie der ursprüngliche Ordinal yp behandelt.

Wenn beide Operanden eines Operators **+**, **-**, *, **div** oder **mod** einen **Integer**-Typ haben, hat das Ergebnis den Typ des gemeinsamen Formats.

Ist einer der beiden Operanden eines Operators **+**, **-** oder * vom Typ **Real**, dann hat das Ergebnis im Modus **{$N-}** den Typ Real, im Modus **{$N+}** den Typ Extended.

Ist der Operand eines Identitäts- oder Negations-Operators ein Integer-Typ, dann ist auch das Ergebnis ein selben Format. Ist der Operand dagegen ein Real-Typ, dann ergibt sich ein Wert vom Typ Real oder Extended.

Der Wert von `X/Y` ist immer vom Typ Real bzw. Extended, wobei der Typ der Operanden keine Rolle spielt. Wenn der Wert `Y` den Wert 0 hat, ergibt sich ein Laufzeitfehler.

Der Wert von `I div J` ist der mathematische Quotient von I / J, auf einen Integerwet abgerundet. Wenn `J` den Wert 0 hat, ergibt sich ein Laufzeitfehler.

Der Operator **mod** liefert den Rest der Division seiner beiden Operanden zurück, also:

```
I mod J = I - (I div J) * J
```

Das Ergebnis einer Operation mit **mod** übernimmt das Vorzeichen von `I`. Wenn J den Wert 0 hat, ergibt sich ein Laufzeitfehler.

Logische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
not	Bitweise Negation	Integer	Integer
and	Bitweises and	Integer	Integer
or	Bitweises or	Integer	Integer
xor	Bitweises xor	Integer	Integer
shl	Linksschieben	Integer	Integer
shr	Rechtsschieben	Integer	Integer

Hinweis

Der Operator **not** ist ein unärer Operator.

Wenn der Operand von **not** einen **Integer**-Typ hat, dann ist auch das Ergebnis im selben Format.

Wenn beide Operanden von **and**, **or** oder **xor** einen Integer-Typ haben, dann hat das Ergebnis das gemeinsame Format der beiden Operanden.

Die Operationen `I shl J` und `I shr J` verschieben den Wert von `I` um J Bitpositionen nach links bzw. nach rechts. Der Typ des Ergebnisses ist der Typ von `I`.

Boolesche Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
not	Negation	Boolean	Boolean
and	logisches and	Boolean	Boolean
or	logisches or	Boolean	Boolean
xor	logisches xor	Boolean	Boolean

Der Operator **not** ist ein unärer Operator.

String-Operator

Operator	Operation	Operandentyp	Ergebnistyp
+	Verbindung	String, Char oder packed String	String

In Turbo Pascal kann der Operator **+** verwendet werden, um zwei String-Operanden miteinander zu verbinden.

Das Ergebnis ist mit jedem **String**-Typ kompatibel (allerdings nicht mit den Typen Char und packed String).

Wenn das Ergebnis die Länge von 255 Zeichen überschreitet, wird es nach dem 255. Zeichen abgeschnitten.

Siehe auch

Relationale Operatoren (=, <>, <, >, <=, >=)

PChar-Operatoren

Die erweiterte Syntax (aktiviert mit **{$X+} Compiler-Direktive**) erlaubt eine Reihe neuer Operationen mit Char-Zeigern.

Die Operatoren plus (**+**) und minus (**-**) können verwendet werden, um den Offset-Teil eines Zeigerwertes zu inkrementieren bzw. dekrementieren.

Der Operator minus kann zur Berechnung des Abstands (Differenz) zwischen den Offset-Teilen zweier Char-Zeiger verwendet werden.

Folgende Konstrukte sind erlaubt, vorausgesetzt, P und Q sind Werte vom Typ PChar und I ist ein Wert vom Typ Word:

Konstrukt	Ergebnis
`P + I`	Addiert `I` zum Offset-Teil von `P`
`I + P`	Addiert `I` zum Offset-Teil von `P`
`P - I`	Subtrahiert`I` vom Offset-Teil von `P`
`P - Q`	Subtrahiert den Offset-Teil von `Q` vom Offset-Teil von `P`

P + I, I + P

Die Operationen **`P + I`** und **`I + P`** addieren I zur durch P bezeichneten Adresse und liefern einen Zeiger, der auf die Position I Zeichen hinter `P` zeigt.

P - I

Die Operation `P - I` subtrahiert `I` von der durch `P` bezeichneten Adresse und resultiert in einem Zeiger, der auf die Position `I` Zeichen vor `P` zeigt.

P - Q

Die Operation `P - Q` berechnet den Abstand zwischen `Q` (der niedrigeren Adresse) und `P` (der höheren Adresse), sie resultiert in einem Wert vom Typ Word, der die Anzahl von Zeichen zwischen `Q` und `P` angibt.

Diese Operation setzt voraus, daß `P` und `Q` auf dasselbe Char-Array zeigen. Wenn die beiden Char-Zeiger auf verschiedene Char-Arrays zeigen, ist das Ergebnis undefiniert.

Mengen-Operatoren

Operator	Operation	Operandentyp
+	Vereinigung	kompatible Mengen
-	Differenz	kompatible Mengen
*****	Durchschnitt	kompatible Mengen

DieResultate der Operationen ergeben sich aus den Gesetzen der Mengenlehre:

- Ein Ordinalwert C ist nur dann in A + B enthalten, wenn er in A oder B enthalten ist.
- Ein Ordinalwert C ist nur dann in A - B enthalten, wenn er in A, aber nicht in B enthalten ist.
- Ein Ordinalwert C ist nur dann in A * B enthalten, wenn er sowohl in A als auch ind B enthalten ist.

Wenn A als Element des Ergebnisses einer Mengen-Operation der kleinste Ordinalwert innerhalb dieser Menge ist und B der größte, dann ist `set of A..B` der Typ des Ergebnisses.

Relationale Operatoren

Operator	Operation	Ergebnistyp	Operandentyp
=	gleich	Boolean	kompatible einfache, Zeiger-, Mengen-, String- oder gepackte String-Typen
<>	ungleich	Boolean	kompatible einfache, Zeiger-, Mengen-, String- oder gepackte String-Typen
<	kleiner als	Boolean	kompatible einfache, String-, gepackte String-Typen oder PChar
>	größer als	Boolean	kompatible einfache, String-, gepackte String-Typen oder PChar
<=	kleiner oder gleich	Boolean	kompatible einfache, String-, gepackte String-Typen oder PChar
>=	größer oder gleich	Boolean	kompatible einfache, String- oder gepackte String-Typen oder PChar
<=	Teilmenge von	Boolean	kompatible Mengen-Typen
>=	Obermenge von	Boolean	kompatible Mengen-Typen
in	Element von	Boolean	linker Operand: beliebiger Ordinaltyp T; rechter Operand: set of Typ T.

Der Adreß-Operator @: Zeiger-Operationen

Über den Adreß-Operator @ kann die Adresse einer Variablen bestimmt und einem Zeiger zugeordnet werden.

Operator	Operation	Operand types	Ergebnistyp
@	Adreß-Ermittlung	Variableneferenz, Prozedure- oder Funktionsbezeichner	Zeiger (derselbe Typ wie **nil**)

@ ist ein unärer Operator. Für die Verwendung des Operators **@** mit einer **prozeduralen Variablen.** sind besondere Regeln zu beachten.

Der Typ dieses Wertes entspricht dem des Typs von **nil** und kann deshalb jeder beliebigen Zeiger-Variablen zugewiesen werden.

@ mit einer Variablen

Gegeben seien folgende Deklarationen:

```
type
  TwoChar = array[0..1] of Char;
var
  Int: Integer;
  TwoCharPtr: ^TwoChar;
```

Die folgende Anweisung weist nun dem Zeiger TwoCharPtr die Adresse der Variablen Int zu:

```
TwoCharPtr := @Int;
```

Da TwoCharPtr^ als Zeiger auf ein Char-Array definiert wurde, läßt sich der Wert von Int, ebenfalls als **array**[0..1] of Char interpretieren.

@ mit einem Wert-Parameter

Wenn **@** auf einen formalen Wert-Parameter angewendet wird, resultiert ein Zeiger auf die Adresse innerhalb des Stacks, die den aktuellen Wert enthält.

Angenommen, Fred sei ein formaler Wert-Parameter in einer Prozedur und FredPtr sei eine Zeiger-Variable.

Wenn die Prozedur die folgende Anweisung ausführt

```
FredPtr := @Fred;
```

bezieht sich FredPtr^ auf den Wert von Fred.

Allerdings bezieht sich FredPtr^ nicht den Parameter Fred selbst, sondern auf den durch Fred übergebenen Wert, der sich auf dem Stack befindet, also auf eine Kopie des Wertes.

@ mit einem Variablen-Parameter

Wird **@** auf einen formalen Variablen-Parameter angewendet, ergibt sich ein Zeiger auf den aktuellen Parameter, dessen Adresse vom Stack gelesen wird.

Nehmen wir beispielsweise folgendes an:

- One ist ein formaler Variablen-Parameter einer Prozedur,
- Two ist eine Variable, die der Prozedur als aktueller Parameter von One übergeben wird,
- OnePtr ist eine Zeiger-Variable.

Wenn die Prozedur folgende Anweisung ausführt:

```
OnePtr := @One;
```

ist OnePtr ein Zeiger auf Two, und OnePtr^ bezieht sich direkt auf Two.

@ mit Prozeduren und Funktionen

Die Anwendung des Operators **@** auf den Bezeichner einer Prozedur oder Funktion liefert die Startadresse der entsprechenden Routine zurück. Turbo Pascal enthält keine weiteren Mechanismen zur Verwendung dieses Zeigerwertes.

Der einzige Zweck der Anwendung von **@** auf Routinen besteht in der Übergabe von Adressen an Maschinensprache-Unterprogramme, d.h. **inline**-Anweisungen.

@ mit Methoden

Der Operator **@** kann auf qualifizierte Methoden-Bezeichner angewandt werden, wenn der Einsprungpunkt der Methode ermittelt werden soll.

Index der ObjectWindows-Objekttypen (alphabetisch)

Dieser Index ist eine alphabetische Liste der ObjectWindows-Objekte. Eine alphabetische Liste der Objekt-Felder finden Sie im Hilfebildschirm **Index der ObjectWindows-Felder**, eine alphabetische Liste der Objekt-Methoden im Hilfebildschirm **Index der ObjectWindows-Methoden.**

Eine Hierarchieliste der Objekttypen finden Sie im Hilfebildschirm **ObjectWindows-Objekthierarchie**.

TApplication
TBufStream
TButton
TCheckBox
TCollection
TComboBox
TControl
TDialog
TDlgWindow
TDosStream
TEdit
TEmsStream
TGroupBox
TListBox
TMDIClient
TMDIWindow
TObject
TRadioButton
TScrollBar
TScroller
TSortedCollection
TStatic
TStrCollection
TStream
TWindow
TWindowsObject

ObjectWindows-Objekthierarchie

In diesem Objekthierarchiediagramm wird die Relation zwischen Vorfahren und abgeleiteten Objekten von links nach rechts dargestellt:

TObject ist Vorfahr von TApplication und TWindowsObject
TWindowsObject ist Vorfahr von TDialog und TWindow
TWindow ist Vorfahr von TEditWindow, TMDIWindow, und TControl
TControl ist Vorfahr von TMDIClient,.etc.

Eine alphabetische Liste der ObjectWindows-Objekttypen enthält der Hilfebildschirm **Index der ObjectWindows-Objekttypen**.

- **TObject**
  - **TApplication**
  - **TWindowsObject**
    - **TDialog**
      - **TDlgWindow**
    - **TWindow**
      - **TMDIWindow**
      - **TControl**
        - **TMDIClient**
        - **TButton**
        - **TScrollBar**
        - **TStatic**
          - **TEdit**
        - **TListBox**
        - **TComboBox**
        - **TGroupBox**
        - **TCheckBox**
          - **TRadioButton**
    - **TScroller**
  - **TStream**
    - **TEMSStream**
    - **TDosStream**
      - **TBufStream**
  - **TCollection**
    - **TSortedCollection**
      - **TStrCollection**

Index der ObjectWindows-Felder

Es folgt eine alphabetische Liste der ObjectWindows-Felder.

Eine alphabetische Liste der Objekt-Methoden enthält der Hilfebildschirm **Index der ObjectWindows-Methoden**. Eine alphabetische Liste der Objekttypen finden Sie im Hilfebildschirm **Index der ObjectWindows-Objekttypen.**

Attr
AutoMode
BufEnd
Buffer
BufPtr
BufSize
ChildList
ChildMenuPos
ClientAttr
ClientWnd
Count
DefaultProc
Delta
DialogProc
Duplicates
ErrorInfo
Flags
FocusChildHandle
Group
HAccTable
Handle
HasHScrollBar
HasVScrollBar
HWindow
Instance
IsModal
Items
KBHandlerWnd
Limit
LineMagnitude
MainWindow
Name
NotifyParent
PageCount
PageMagnitude
Parent
Position
Scroller
Size
Status
TextLen
TrackMode
TransferBuffer
Window
XLine
XPage
XPos

XRange
XUnit
YLine
YPage
YPos
YRange
YUnit

Index der ObjectWindows-Methoden

Eine alphabetische Liste der ObjectWindow-Felder finden Sie im Hilfebildschirm **Index der ObjectWindows-Felder.** Eine alphabetische Liste der Objekte enthält der Hilfebildschirm **Index der ObjectWindows-Objects.**

AddString
ArrangeIcons
At
AtDelete
AtFree
AtInsert
AtPut
AutoScroll
BeginView
BNClicked
Cancel
CanClose
CanUndo
CascadeChildren
Check
ChildWithID
Clear
ClearList
ClearModify
CloseChildren
CMArrangeIcons
CMCascadeChildren
CMCloseChildren
CMCreateChild
CMEditClear
CMEditCopy
CMEditCut
CMEditDelete
CMEditPaste
CMEditUndo
CMTileChildren
Compare
Copy
CopyFrom
Create
CreateChild
CreateChildren
Cut
DefChildProc
DefCommandProc
DefNotificationProc
DefWndProc
Delete
DeleteAll
DeleteLine
DeleteSelection
DeleteString
DeleteSubText
DeltaPos

DisableAutoCreate
DisableTransfer
DispatchScroll
Done
EnableAutoCreate
EnableKBHandler
EnableTransfer
EndDlg
EndView
Error
ExecDialog
Execute
FirstThat
Flush
ForEach
Free
FreeAll
FreeItem
Get
GetCheck
GetChildPtr
GetClassName
GetClient
GetCount
GetID
GetItem
GetItemHandle
GetLine
GetLineFromPos
GetLineIndex
GetLineLength
GetMsgID
GetNumLines
GetPos
GetPosition
GetRange
GetSelection
GetSelIndex
GetSelString
GetSiblingPtr
GetSize
GetString
GetStringLen
GetSubText
GetText
GetWindowClass
HideList
HScroll
IndexOf
Init
InitApplication
InitChild
InitClientWindow
InitInstance
InitMainWindow

InitResource
Insert
InsertString
IsFlagSet
IsModified
IsVisibleRect
KeyOf
LastThat
Load
MakeWindow
MessageLoop
Next
Ok
Pack
Paint
Paste
Previous
ProcessAccels
ProcessAppMsg
ProcessDlgMsg
ProcessMDIAccels
Put
PutChildPtr
PutItem
PutSiblingPtr
Read
ReadStr
Register
Reset
Run
SBBottom
SBLineDown
SBLineUp
SBPageDown
SBPageUp
SBThumbPosition
SBThumbTrack
SBTop
Scroll
ScrollBy
ScrollTo
Search
Seek
SelectionChanged
SetSelection
SendDlgItemMsg
SetCheck
SetFlags
SetKBHandler
SetLimit
SetPageSize
SetPosition
SetRange
SetSBarRange
SetSelIndex

SetSelString
SetText
SetUnits
SetupWindow
Show
ShowList
Store
StrRead
StrWrite
TileChildren
Toggle
Transfer
TransferData
Truncate
Uncheck
Undo
ValidWindow
VScroll
WMActivate
WMClose
WMCommand
WMDestroy
WMHScroll
WMInitDialog
WMLButtonDown
WMNCDestroy
WMPaint
WMVScroll
WMSize
Write
WriteStr

Attr Felder

TDialog.Attr
TWindow.Attr

Handle Felder

TDosStream.Handle
TEmsStream.Handle

Status Felder

TApplication.Status
TStream.Status
TWindowsObject.Status

TextLen Felder

TStatic.TextLen
TComboBox.TextLen

ArrangeIcons Methoden

TMDIClient.ArrangeIcons
TMDIWindow.ArrangeIcons

CanClose Methoden

TApplication.CanClose
TWindowsObject.CanClose

CascadeChildren Methoden

TMDIClient.CascadeChildren
TMDIWindow.CascadeChildren

CloseWindow

CloseWindow ist sowohl eine ObjectWindows-Methode als auch eine Windows API-Funktion.

Wählen Sie den gewünschten Begriff:
CloseWindow (API-Function)
TWindowsObject.CloseWindow (Methode)

Compare Methoden

TSortedCollection.Compare
TStrCollection.Compare

Copy

Copy ist sowohl eine Turbo-Pascal-Function als auch eine ObjectWindows-Methode.

Wählen Sie den gewünschten Begriff:
Copy (Funktion)
TEdit.Copy (Methode)

Create Methoden

TDialog.Create
TDlgWindow.Create
TWindow.Create
TWindowsObject.Create

DefWndProc Methoden

TDialog.DefWndProc
TMDIWindow.DefWndProc
TWindow.DefWndProc
TWindowsObject.DefWndProc

Delete

Delete ist sowohl eine Turbo-Pascal-Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den gewünschten Begriff:
Delete (Prozedur)
TCollection.Delete

Done Methoden

TApplication.Done
TBufStream.Done
TCollection.Done
TDialog.Done
TDosStream.Done
TEmsStream.Done
TMDIWindow.Done
TObject.Done
TWindow.Done
TWindowsObject.Done

Error Methoden

TApplication.Error
TCollection.Error
TStr.Error

FirstThat Methoden

TCollection.FirstThat
TWindowsObject.FirstThat

Flush

Flush ist sowohl eine Turbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den entsprechenden Begriff:
Flush (Prozedur)
TBufStream.Flushm Methode
TStream.Flush Methode

Flush Methoden

TBufStream.Flush
TStream.Flush

ForEach Methoden

TCollection.ForEach
TWindowsObject.ForEach

Free Methoden

TCollection.Free
TObject.Free

FreeItem Methoden

TCollection.FreeItem
TStrCollection.FreeItem

GetClassName

GetClassName ist sowohl eine ObjectWindows-Methode (für verschiedene Objekte) und eine Windows API-Funktion.

Wählen Sie den gewünschten Begriff:

GetClassName (Windows API-Function)
TButton.GetClassName (Methode)
TComboBox.GetClassName (Methode)
TControl.GetClassName (Methode)
TEdit.GetClassName (Methode)
TGroupBox.GetClassName (Methode)
TListBox.GetClassName (Methode)
TMDIClient.GetClassName (Methode)
TMDIWindow.GetClassName (Methode)
TScrollBar.GetClassName (Methode)
TStatic.GetClassName (Methode)
TWindowsObject.GetClassName (Methode)

GetClient Methoden

TMDIWindow.GetClient
TWindowsObject.GetClient

GetID Methoden

TWIndows.GetID
TWIndowsObject.GetID

GetItem Methoden

TCollection.GetItem
TStrCollection.GetItem

GetPos Methoden

TBufStream.GetPos
TDosStream.GetPos
TEmsStream.GetPos
TStream.GetPos

GetSize Methoden

TBufStream.GetSize
TDosStream.GetSize
TEmsStream.GetSize
TStream.GetSize

GetWindowClass Methoden

TDlgWindow.GetWindowClass
TMDIDlgWindow.GetWindowClass
TWindow.GetWindowClass
TWindowsObject.GetWindowClass

IndexOf Methoden

TCollection.IndexOf
TSortedCollection.IndexOf

Init Methoden

TApplication.Init
TBufStream.Init
TButton.Init
TCheckBox.Init
TCollection.Init
TComboBox.Init
TControl.Init
TDialog.Init
TDlgWindow.Init
TDosStream.Init
TEdit.Init
TEmsStream.Init
TGroupBox.Init
TListBox.Init
TMDIClient.Init
TMDIWindow.Init
TOBject.Init
TRadioButton.Init
TScrollBar.Init
TScroller.Init
TStatic.Init
TWindow.Init
TWindowsObjectInit

InitResource Methoden

TButton.InitResource
TCheckBox.InitResource
TControlInit.InitResource
TComboBox.InitResource
TGroupBox.InitResource
TStatic.InitResource
TWindow.InitResource

Insert

Insert ist sowohl eine Turbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den entsprechenden Begriff:
Insert (Prozedur)
TCollection.Insert (Methode)
TEdit.Insert (Methode)
TSortedCollection.Insert (Methode)

Insert-Methoden

TCollection.Insert
TEdit.Insert
TSortedCollection.Insert

Load-Methoden

TCheckBox.Load
TCollection.Load
TComboBox.Load
TDialog.Load
TGroupBox.Load
TMDIClient.Load
TMDIWindow.Load
TScrollBar.Load
TScroller.Load
TSortedCollection.Load
TStatic.Load
TWindow.Load
TWindowsObject.Load

Name

Name ist sowohl eine Turbo Pascal Standardanweisung und eine ObjectWindows-Methode.

Wählen Sie den entsprechenden Begriff:
Name (Standardanweisung)
TApplication.Name

PutItem Methoden

TCollection.PutItem
TStrCollection.PutItem

Read

Read ist sowohl eine Turbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den entsprechenden Begriff:
Read (Prozedur)
TBufStream.Read (Methode)
TDosStream.Read (Methode)
TEmsStream.Read (Methode)
TStream.Read (Methode)

Read Methoden

- **TBufStream.Read**
- **TDosStream.Read**
- **TEmsStream.Read**
- **TStream.Read**

Register Methoden

TControl.Register
TWindowsObject.Register

Reset

Reset ist sowohl eineTurbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den gewünschten Begriff:
Reset (Prozedur)
TStream.Reset (Methode)

Search methods

TEdit.Search
TSortedCollection.Search

Seek

Seek ist sowohl eine Turbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den gewünschten Begriff:

Seek (Prozedur)
TBufStream.Seek (Methode)
TDosStream.Seek (Methode)
TEmsStream.Seek (Methode)
TStream.Seek (Methode)

Seek-Methoden

TBufStream.Seek
TDosStream.Seek
TEmsStream.Seek
TStream.Seek

SetRange Methoden

TScrollBar.SetRange
TScroller.SetRange

SetupWindow Methoden

TMDIWindow.SetupWindow
TScrollBar.SetupWindow
TWindow.SetupWindow
TWindowsObject.SetupWindow

ScrollTo

ScrollTo ist sowohl eine Turbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den entsprechenden Begriff:
ScrollTo (Prozedur)
TScroller.ScrollTo (Methode)

Store Methoden

TCheckBox.Store
TCollection.Store
TComboBox.Store
TDialog.Store
TGroupBox.Store
TMDIClient.Store
TMDIWindow.Store
TScrollBar.Store
TScroller.Store
TSortedCollection.Store
TWindowsObject.Store
TStatic.Store
TWindow.Store

TileChildren Methoden

TMDIClient.TileChildren
TMDIWindow.TileChildren

Transfer Methoden

TCheckBox.Transfer
TComboBox.Transfer
TEdit.Transfer
TListBox.Transfer
TScrollBar.Transfer
TStatic.Transfer
TWindowsObject.Transfer

Truncate

Truncate ist sowohl eine Turbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den gewünschten Begriff:

Truncate (Prozedur)
TBufStream.Truncate (Methode)
TDosStream.Truncate (Methode)
TEmsStream.Truncate (Method)
TStream.Truncate (Methode)

Truncate Methoden

TBufStream.Truncate
TDosStream.Truncate
TEmsStream.Truncate
TStream.Truncate

WMActivate Methoden

TWindow.WMActivate
TWindowsObject.WMActivate

WMClose methods

TDialog.WMClose
TWindowsObject.WMClose

WMHScroll Methoden

TWindow.WMHScroll
TWindowsObject.WMHScroll

WMPaint Methoden

TControl.WMPaint
TWindow.WMPaint

WMVScroll Methoden

TWindow.WMVScroll
TWindowsObject.WMVScroll

Write

Write ist sowohl eine Turbo Pascal Prozedur als auch eine ObjectWindows-Methode.

Wählen Sie den gewünschten Begriff:

Write (Prozedur)
TBufStream.Write (Methode)
TDosStream.Write (Methode)
TEmsStream.Write (Methode)
TStream.Write (Methode)

Write Methoden

TBufStream.Write
TDosStream.Write
TEmsStream.Write
TStream.Write

Index der Funktionen von Turbo Pascal

Abs	Gibt den absoluten Wert des Arguments zurück
Addr	Liefert die Adresse des angegebenen Objekts
ArcTan	Gibt den Arcustangens des Arguments zurück
Chr	Liefert das Zeichen, dessen ASCII-Code dem Wert von x entspricht.
Concat	Verbindet zwei oder mehrere Stringteile
Copy	Gibt einen Teil eines Strings zurück.
Cos	Gibt den Cosinus des Arguments zurück.
CSeg	Gibt die Adresse des aktuellen Code-Segments (CS-Register) zurück.
DosVersion	**Gibt die DOS-Versionsnummer zurück**
DSeg	Gibt die Adresse des Daten-Segments (d.h. den Inhalt des DS-Registers) zurück.
Eof	Prüft, ob das Ende der durch f angegebenen Datei erreicht ist.
Eoln	Prüft, ob das Zeilenende in einer Textdatei erreicht ist.
Exp	Gibt "e hoch" dem Argument zurück.
FilePos	Liefert die aktuelle Position innerhalb einer Datei.
FileSize	Gibt die Größe einer Datei zurück.
Frac	Gibt den nicht-ganzzahligen Teil des Arguments zurück.
Hi	Gibt das höherwertige Byte des Arguments zurück.
Int	Gibt den ganzzahligen Anteil des Arguments zurück.
IOResult	Gibt den Status der letzten Ein-/Ausgabe-Operation als Integer zurück
Length	Gibt die aktuelle Länge des als "s" übergebenen Stringausdrucks zurück.
Ln	Liefert den natürlichen Logarithmus des Arguments.
Lo	Gibt das niederwertige Byte des Arguments zurück.
MaxAvail	Gibt die Umfang des größten freien Blocks auf dem Heap in Bytes zurück.
MemAvail	Liefert den Gesamtumfang des freien Platzes auf dem Heap in Bytes.
Odd	Prüft, ob das Argument eine ungerade Zahl darstellt.
Ofs	Gibt den Offset-Anteil der Adresse des angegebenen Objekts zurück.
Ord	Gibt die Ordinalzahl des Arguments zurück.
ParamCount	Gibt die Anzahl der Kommandozeilen-Parameter zurück, die dem Programm übergeben wurden.
ParamStr	Gibt den Kommandozeilen-Parameter [Index] zurück.
Pi	Gibt den Wert der mathematischen Konstanten Pi zurück.
Pos	Sucht den als s übergebenen String-Ausdruck nach dem ersten Vorkommen des String-Ausdrucks substr ab.
Pred	Gibt den Vorgänger des Arguments zurück.
Ptr	Konvertiert zwei Angaben für Segment und Offset in einen Wert des Typs Pointer.
Random	Liefert eine Zufallszahl.
Round	Rundet das (Real-)Argument auf einen ganzzahligen Wert.
SeekEof	Prüft, ob sich zwischen der momentanen Position und dem Dateiende noch "lesbare" Daten befinden
SeekEoln	Prüft, ob sich zwischen der momentanen Position und dem nächsten Zeilenende "lesbare" Daten befinden.
Seg	Gibt die Segmentadresse des angegebenen Objekts zurück.
Sin	Liefert den Sinus des Arguments.
SizeOf	Gibt die Anzahl von Bytes zurück, die das Argument an Speicherplatz belegt.
SPtr	Gibt den momentanen Wert des Stackzeigers (SP-Register) zurück.
Sqr	Gibt das Quadrat des Arguments zurück.
Sqrt	Gibt die Quadratwurzel des Arguments zurück.
SSeg	Gibt die Adresse des Stack-Segments zurück, d.h. den Wert des SS-Registers.

Succ	Gibt den Nachfolger des Arguments zurück.
Swap	Vertauscht das niederwertige und das höherwertige Byte des Arguments.
Trunc	Wandelt einen Realwert durch Abschneiden der Nachkommastellen in einen Integerwert um.
UpCase	Konvertiert Klein- in Großbuchstaben. Die deutschen Umlaute werden nicht berücksichtigt.

Index der Prozeduren von Turbo Pascal

Append	Öffnet eine existierende Datei für das Anhängen weiterer Daten
Assign	Ordnet einer Datei-Variablen eine externe Datei zu.
BlockRead	Liest einen oder mehrere Records in eine Puffervariable.
BlockWrite	Schreibt einen oder mehrere Records aus einer Puffervariablen.
ChDir	Wechselt das aktuelle Verzeichnis
Close	Schließt die durch f angegebene Datei.
Dec	Zählt eine Variable um einen bestimmten Wert herunter.
Delete	Löscht count Zeichen ab der Position index im String s.
Dispose	Gibt den einer Zeigervariablen zugeordneten Speicherplatz auf dem Heap wieder frei.
DoneWinCrt	Löscht ein CRT-Fenster.
Erase	Löscht eine externe Datei.
Exit	Verläßt den momentanen Block mit sofortiger Wirkung.
FillChar	Füllt angegebene Anzahl (count) zusammenhängender Bytes mit angegebenem Wert vom Typ Byte oder Char.
Flush	Erzwingt das physikalische Schreiben des Puffers einer Textdatei, die für Ausgaben geöffnet wurde.
FreeMem	Gibt einen Speicherbereich bestimmter Größe auf dem Heap wieder frei.
GetDir	Ermittelt das momentan gesetzte Verzeichnis eines Laufwerks.
GetMem	Belegt einen Bereich von size Bytes auf dem Heap und weist die Startadresse dieses Bereichs der als p übergebenen Variablen zu, erzeugt also eine dynamische Variable.
Halt	Bricht die Ausführung des Programms ab und führt einen Rücksprung zur DOS-Kommandoebene bzw. dem Programm durch, von dem aus dieses Programm gestartet wurde.
Inc	Erhöht eine Variable um 1 bzw. den durch n angegebenen Wert.
Insert	Fügt Substring in String ein.
MkDir	Erzeugt ein Unterverzeichnis.
Move	Kopiert "count" Bytes von dem durch "source" angegebenen Speicherbereich in den durch "dest" bezeichneten Bereich.
New	Belegt einen Bereich auf dem Heap und setzt die als p übergebene Zeigervariable auf die Startadresse dieses Bereichs, erzeugt also eine dynamische Variable.
Randomize	Initialisiert den "Zufallsgenerator" mit einem Wert, der sich aus Datum und Uhrzeit des Systems ergibt.
Read	Bei typisierten Dateien: Liest eine oder mehrere Komponenten in eine Variable. Bei Textdateien: Liest einen oder mehrere Werte in eine oder mehrere Variablen.
Readln	Führt einen Aufruf von Read aus und springt dann zum Anfang der nächsten Zeile innerhalb der über f angegebenen Datei.
Rename	Gibt einer externen Datei einen neuen Namen.
Reset	Öffnet eine existierende Datei.
Rewrite	Erzeugt eine neue Datei und öffnet sie.
RmDir	Löscht das durch den Stringausdruck s bezeichnete leere Unterverzeichnis.
RunError	Erzeugt einen Laufzeitfehler, der das Programm definiert abbricht.
Seek	Setzt den Positionszeiger in der durch f bezeichneten Datei auf die Komponente mit der Nummer n, wobei die Zählung mit 0 beginnt.
SetTextBuf	Weist einer Textdatei-Variablen einen Puffer zu.
Str	Konvertiert den als x übergebenen Wert in eine Zeichenfolge.
Truncate	Schneidet eine Datei an der aktuellen Position ab.
Val	Interpretiert die Zeichenfolge des Strings s als numerischen Wert.

Write Bei typisierten Dateien: Schreibt Variable in eine Dateikomponente. Bei Textdateien: Schreibt einen oder mehr Werte in die Datei.

Writeln Ruft Write (für Textdateien) auf und schreibt dann einen Zeilenvorschub in die durch f angegebene Datei.

Abs (Funktion)

Gibt den Absolutwert des Arguments zurück.

Deklaration

```
function Abs(X): <Ergebnis hat denselben Typ wie das Argument>;
```

Beispielcode

Abs.PAS

Abstract (Prozedur) (Unit WObjects)

Deklaration

```
procedure Abstract;
```

Anmerkungen

Ein Aufruf dieser Prozedur beendet das Programm mit dem Laufzeitfehler 211.

Bei der Implementation eines abstrakten Objekttyps sollten Sie Aufrufe von Abstract in denjenigen virtuellen Methoden verwenden, die von abgeleiteten Typen überschrieben werden müssen.

Damit wird sichergestellt, daß jeder Versuch, Instanzen des abstrakten Objekttyps zu verwenden, scheitert.

Addr (Funktion)

Liefert die Adresse des angegebenen Objekts.

Deklaration

```
function Addr(X): pointer;
```

Siehe auch:

Ofs
Ptr
Seg

Beispielcode

Addr.PAS

AllocMultiSel (Funktion) (Unit WObjects)

Deklaration

```
function AllocMultiSel(Count: Integer): PMultiSelRec;
```

Anmerkungen

Legt eine **TMultiSelRec** Datenstruktur an, wobei der Referenzzähler gleich Count, gesetzt und im Feld Selections ausreichend Platz reserviert wird, um Count Auswahlen (0..Count-1) aufnehmen zu können.

Gibt **nil** zurück, falls nicht genügend Speicher für den gesamten Record zur Verfügung steht.

Siehe auch

FreeMultiSel (Prozedur)

Append (Prozedur)

Öffnet eine existierende Datei für das Anfügen weiterer Daten.

Deklaration

```
procedure Append(var F: Text);
```

F ist eine Textdatei-Variable.

Siehe auch:

Assign
Close
Reset
Rewrite

Beispielcode

Append.PAS

ArcTan (Funktion)

Gibt den Arcustangens des Arguments zurück.

Deklaration

```
function ArcTan(X: Real): Real;
```

Anmerkungen

Turbo Pascal verfügt über keine Funktion, um den Tangens Tan zu berechnen. Der Tangens eines Winkel x läßt sich aber mit Hilfe des folgenden Ausdrucks bestimmen:

Sin(x) / **Cos**(x)

Beispielcode

Arctan.PAS

Assign (Prozedur)

Ordnet einer Datei-Variablen eine externe Datei zu.

Deklaration

procedure Assign(**var** F; Name);

Einschränkungen

Kann bei offener Datei nicht verwendet werden.

Siehe auch:

Append
Close
Reset
Rewrite

Beispielcode

Assign.PAS

AssignCrt (Prozedur) (Unit WinCrt)

Ordnet dem Bildschirm ("CRT") eine Textdatei zu.

Deklaration

```
procedure procedure AssignCrt(var F: Text);
```

Beispielcode

AssignCr.PAS

BlockRead (Prozedur)

Liest einen oder mehrere Records in eine Puffervariable.

Deklaration

```
procedure BlockRead(var F: File; var Buf; Count: Word [; var Result:
Word]);
```

F	untypisierte Datei
Buf	beliebige Variable
Count	Ausdruck vom Typ Word
Result	Variable vom Typ Word

Einschränkungen

Die Datei muß offen sein.

Siehe auch:

BlockWrite

Beispielcode

Blockrd.PAS

BlockWrite (Prozedur)

Schreibt einen oder mehrere Records aus einer Puffervariablen in eine Datei.

Deklaration

```
procedure BlockWrite(var f: File; var Buf; Count: Word [; var Result: Word]);
```

F	untypisierte Datei
Buf	beliebige Variable
Count	Ausdruck vom Typ Word
Result	Variable vom Typ Word

Einschränkungen

Die Datei muß offen sein.

Siehe auch:

BlockRead

Beispielcode

Blockrd.PAS

ChDir (Prozedur)

Wechselt das aktuelle Verzeichnis.

Deklaration

```
procedure ChDir(S: String);
```

Anmerkungen

Mit **$I-** gibt IOResult 0 zurück, wenn die Operation erfolgreich war, sonst einen Fehlercode.

Siehe auch:

GetDir
MkDir
RmDir

Beispielcode

Chdir.PAS

Chr (Funktion)

Liefert das Zeichen mit dem angegebenen Ordinalwert.

Deklaration

```
function Chr(X: Byte): Char;
```

Siehe auch:

Ord

Beispielcode

Chr.Pas

Close (Prozedur)

Schließt eine offene Datei.

Deklaration

```
procedure Close(var F);
```

Siehe auch:

Append
Assign
Reset
Rewrite

Beispielcode

Close.PAS

ClrEol (Prozedur) (Unit WinCrt)

Löscht alle Zeichen ab der aktuellen Cursorposition bis zum Zeilenende, ohne dabei den Cursor zu bewegen.

Deklaration

```
procedure ClrEol;
```

Beispielcode

ClrEol.PAS

ClrScr (Prozedur) (Unit WinCrt)

Löscht den Bildschirminhalt und setzt den Cursor in die obere linke Ecke.

Deklaration

```
procedure ClrScr;
```

Beispielcode

ClrScr.PAS

Concat (Funktion)

Verbindet zwei oder mehrere Stringteile miteinander.

Deklaration

```
function Concat(s1 [, s2,..., sn]: String): String;
```

Siehe auch:

Copy
Delete
Insert
Length
Pos

Beispielcode

Concat.PAS

Copy (Funktion)

Gibt einen Teil eines Strings zurück.

Deklaration

```
function Copy(S: String; Index: Integer; Count: Integer): String;
```

Siehe auch:

Concat
Delete
Insert
Length
Pos

Beispielcode

Copy.PAS

Cos (Funktion)

Gibt den Cosinus des Arguments zurück. X ist ein Winkel in Rad (Bogenmaß).

Deklaration

```
function Cos(X: Real): Real;
```

Siehe auch:

ArcTan
Sin

Beispielcode

Cos.PAS

CreateDir (Prozedur) (Unit WinDos)

Legt ein neues Unterverzeichnis an.

Deklaration

```
procedure CreateDir(Dir: PChar)
```

Anmerkungen

Führt dieselben Funktionen wie **MkDir** aus, verwendet jedoch statt eines null-terminierten Strings einen Pascal-String.

Siehe auch

GetCurDir
SetCurDir
RemoveDir

Beispielcode

CreateDr.PAS

CSeg (Funktion)

Gibt die aktuelle Codesegment- Adresse (CS-Register) zurück.

Deklaration

```
function CSeg: Word;
```

Siehe auch:

DSeg
Sseg

Beispielcode

CSeg.Pas

CursorTo (Prozedur) (Unit WinCrt)

Bewegt den Cursor zu den gegebenen Koordinaten innerhalb des virtuellen Bildschirmfensters.

Deklaration

```
procedure CursorTo(X, Y: Integer);
```

Anmerkunen

Die Koordinaten der oberen linken Ecke des CRT-Fensters sind (0, 0). CursorTo setzt die Variable Cursor auf (X, Y).

Beispielcode

CursorTo.PAS

Dec (Prozedur)

Zählt eine Variable um einen bestimmten Wert herunter.

Deklaration

```
procedure Dec(var X[ ; N: Longint]);
```

Siehe auch:

Inc
Pred
Succ

Beispielcode

Dec.PAS

Delete (Prozedur)

Löscht einen Teilstring aus einem String.

Deklaration

```
procedure Delete(var S: String; Index: Integer; Count:Integer);
```

Siehe auch:

Concat
Copy
Insert
Length
Pos

Beispielcode

Delete.PAS

DiskFree (Funktion) (Unit WinDos)

Liefert die Größe des freien Speicherplatzes auf einem Laufwerk zurück.

Deklaration

```
function DiskFree(Drive: Byte): Longint;
```

wobei

0	Standardlaufwerk
1	Laufwerk A
2	Laufwerk B
3	Laufwerk C
etc.	

Rückgabewert

-1, falls die Laufwerksnummer ungültig ist.

Siehe auch

DiskSize
GetDir

Beispielcode

Diskfree.PAS

DiskSize (Funktion) (Unit WinDos)

Gibt die Gesamtkapazität (in Bytes) des angegebenen Laufwerks zurück.

Deklaration

```
function DiskSize(Drive: Byte): Longint;
```

wobei Laufwerk:

0	Standardlaufwerk
1	Laufwerk A
2	Laufwerk B
3	Laufwerk C
etc.	

Rückgabewert

-1, falls die Laufwerksnummer ungültig ist.

Siehe auch

DiskFree
GetDir

Beispielcode

Disksize.PAS

Dispose (Prozedur)

Gibt den durch eine dynamische Variable belegten Speicherplatz auf dem Heap wieder frei.

Deklaration

```
procedure Dispose(var P: Pointer [ , Destructor ]);
```

Anmerkungen

Die Syntax von Dispose wurde dahingehend erweitert, daß man damit auch ein **Objekt** auf dem Heap wieder freigegeben kann, indem man den entsprechenden **Destruktor** als zweiten Parameter angibt:

```
Dispose(P, Done);
```

Siehe auch:

FreeMem
GetMem
New

Beispielcode

Dispose.PAS

DoneWinCrt (Prozedur) (Unit WinCrt)

Löscht ein CRT-Fenster, falls dies nicht schon geschehen ist.

Deklaration

```
procedure DoneWinCrt;
```

Anmerkungen

Rufen Sie DoneWinCrt unmittelbar vor Programmende auf, damit ein CRT-Fenster nicht in den inaktiven Status gerät.

Beispielcode

DoneWCrt.PAS

DosVersion (Funktion) (Unit WinDos)

Gibt die DOS-Versionsnummer zurück.

Deklaration

```
function DosVersion: Word;
```

Anmerkungen

Das niederwertige Byte des Ergebnisses steht für die Hauptversion, das höherwertige für die Unterversion.

Beispielcode

DOSVrsn.PAS

DSeg (Funktion)

Gibt die Adresse des Datensegments (d.h. den Inhalt des DS-Registers) zurück.

Deklaration

```
function DSeg: Word;
```

Siehe auch:

CSeg
Sseg

Beispielcode

CSeg.PAS

Eof (Funktion)

Prüft, ob das Ende der durch F angegebenen Datei erreicht ist.

Deklaration

typisierte oder untypisierte Datei: `function Eof(var F): Boolean;`
Textdatei: `function Eof [ (var F: Text) ]: Boolean;`

Siehe auch:

Eoln
SeekEof

Beispielcode

Eof.PAS

Eoln (Funktion)

Prüft, ob ein Zeilenende in einer Textdatei erreicht ist.

Deklaration

```
function Eoln [(var F: Text) ]: Boolean;
```

Siehe auch:

Eof
SeekEoln

Beispielcode

Eoln.PAS

Erase (Prozedur)

Löscht eine externe Datei

Deklaration

```
procedure Erase(var F);
```

Siehe auch:

Rename

Beispielcode

Erase.PAS

Exit (Prozedur)

Verläßt den momentanen Block mit sofortiger Wirkung.

Deklaration

```
procedure Exit;
```

Anmerkungen

Exit innerhalb des Hauptprogramms beendet das gesamte Programm.

Siehe auch:

Halt

Beispielcode

Exit.PAS

Exp (Funktion)

Gibt e hoch dem Argument zurück.

Deklaration

```
function Exp(X: Real): Real;
```

Rückgabewert

Der Wert e hoch X, wobei e die Eulersche Zahl, die Basis des natürlichen Logarithmus ist.

Siehe auch:

Ln

Beispielcode

Exp.PAS

FileExpand (Funktion) (Unit WinDos)

Erweitert einen Dateinamen zu einem vollständigen DOS-Dateinamen.

Deklaration

```
function FileExpand(Dest, Name: PChar): PChar;
```

Siehe auch

Dateiattribut-Konstanten
FindFirst
FindNext
FileSplit
TSearchRec

Beispielcode

FileExp.PAS

FilePos (Funktion)

Liefert die aktuelle Position innerhalb einer Datei.

Deklaration

```
function FilePos(var F): Longint;
```

Siehe auch:

FileSize
Seek

Beispielcode

FilePos.PAS

FileSearch (Funktion) (Unit WinDos)

Sucht nach einer Datei.

Deklaration

```
function FileSearch(Dest, Name, List: PChar): PChar;
```

Anmerkungen

Sucht in einer Verzeichnisliste (DirList). Die Verzeichnisnamen in DirList müssen jeweils durch ein Semikolon voneinander getrennt sein.

Siehe auch

FileExpand
FileSplit

Beispielcode

FileSch.PAS

FileSize (Funktion)

Gibt die Größe einer Datei zurück.

Deklaration

```
function FileSize(var F): Longint;
```

Siehe auch:

FilePos

Beispielcode

FilePos.PAS

FileSplit (Funktion) (Unit WinDos)

Teilt einen Dateinamen in seine drei Komponenten.

Deklaration

```
function FileSplit(Path, Dir, Name, Ext: PChar): Word
```

Siehe auch

FileExpand
FindFirst
FindNext

Beispielcode

FileSplt.PAS

FillChar (Prozedur)

Füllt Count aufeinanderfolgende Speicherzellen mit dem übergebenen Wert vom Typ Byte oder Char.

Deklaration

```
procedure FillChar(var X; Count: Word; value);
```

Anmerkungen

Keine Bereichsüberprüfung!

Siehe auch:

Move

Beispielcode

Fillchar.PAS

FindFirst (Prozedur) (Unit WinDos)

Sucht im genannten Verzeichnis nach der entsprechenden Datei.

Deklaration

```
procedure FindFirst(Path: PChar; Attr: Word; var F: TSearchRec);
```

Anmerkungen

Sucht im angegebenen (oder aktuellen) Verzeichnis nach dem ersten Eintrag, der dem angegebenen Dateinamen und den genannten Attributen entspricht.

Siehe auch

Dateiattribut-Konstanten
FileExpand
FindNext
TSearchRec

Beispielcode

Findfrst.PAS

FindNext (Prozedur) (Unit WinDos)

Sucht den nächsten Eintrag, der dem Namen und den Attributen entspricht, die in einem vorherigen Aufruf von FindFirst. genannt wurden.

Deklaration

```
procedure FindNext(var F: TSearchRec);
```

Anmerkungen

Fehler werden in **DosError** ausgegeben. Es kann nur der Fehlercode 18 (keine weiteren Dateien) auftreten.

Siehe auch

Dateiattribut-Konstanten
FileExpand
FindFirst
TSearchRec

Beispielcode

Findfrst.PAS

Flush (Prozedur)

Erzwingt das physikalische Schreiben des Puffers einer Textdatei, die für Ausgaben geöffnet wurde.

Deklaration

```
procedure Flush(var F: Text);
```

Beispielcode

Flush.PAS

Frac (Funktion)

Gibt den nicht-ganzzahligen Teil des Arguments zurück.

Deklaration

```
function Frac(X: Real): Real;
```

Siehe auch:

Int

Beispielcode

Frac.PAS

FreeMem (Prozedur)

Gibt einen Speicherbereich bestimmter Größe auf dem Heap wieder frei.

Deklaration

```
procedure FreeMem(var P: Pointer; Size: Word);
```

Siehe auch:

Dispose
GetMem
New

Beispielcode

FreeMem.PAS

FreeMultiSel (Prozedur) (Unit WObjects)

Deklaration

```
procedure FreeMultiSel(P: PMultiSelRec);
```

Anmerkungen

Gibt einen **TMultiSelRec**-Record frei, der zuvor von **AllocMultiSel** angelegt wurde.

GetArgCount (Funktion) (Unit WinDos)

Liefert die Anzahl der Parameter, die dem Programm in der Kommandozeile übergeben wurden.

Deklaration

```
function GetArgCount: Integer
```

Beispielcode

GetArgCo.PAS

GetArgStr (Funktion) (Unit WinDos)

Liefert den durch Index bezeichneten Kommandozeilenparameter.

Deklaration

```
function GetArgStr(Dest: PChar; Index: Integer; MaxLen: Word): PChar;
```

Rückgabewert

Parameter	Rückgabewert
Index < 0 oder > GetArGCount	leerer String
Index = 0	Dateiname des aktuellen Moduls; Dest = Rückgabewert

Siehe auch

GetArgCount

Beispielcode

GetArgSt.PAS

GetCBreak (Prozedur) (Unit WinDos)

Ermittelt, bei welchen Operationen DOS auf Ctrl-Break prüft.

Deklaration

```
procedure GetCBreak(var Break: Boolean);
```

Siehe auch

SetCBreak

Beispielcode

GetCBrk.PAS

GetCurDir (Funktion) (Unit WinDos)

Liefert das aktuelle Verzeichnis des genannten Laufwerkes.

Deklaration

```
function GetCurDir(Dir: PChar; Drive: Byte): PChar
```

wobei Laufwerk=

0	Standardlaufwerk
1	Laufwerk A
2	Laufwerk B
3	Laufwerk C
etc.	

Siehe auch

SetCurDir
CreateDir
RemoveDir

Beispielcode

GetCurDr.PAS

GetDate (Prozedur) (Unit WinDos)

Liefert das aktuelle Betriebssystem-Datum.

Deklaration

```
procedure GetDate(var Year, Month, Day, DayOfWeek: Word);
```

Siehe auch

GetTime
SetDate
SetTime

Beispielcode

GetDate.PAS

GetDir (Prozedur)

Ermittelt das aktuelle Verzeichnis des angegebenen Laufwerks.

Deklaration

```
procedure GetDir(D: Byte; var S: String);
```

wobei D wie folgt gesetzt wird:

0	Standardlaufwerk
1	Laufwerk A
2	Laufwerk B
3	Laufwerk C
etc.	

Siehe auch:

ChDir
DiskFree
DiskSize
MkDir
RmDir

Beispielcode

GetDir.PAS

GetEnvVar (Funktion) (Unit WinDos)

Liefert einen Zeiger auf den Wert einer gegebenen Umgebungsvariable.

Deklaration

```
function GetEnvVar(Var Name: PChar): PChar;
```

Rückgabewert

0 Falls die genannte Umgebungsvariable nicht existiert.

Beispielcode

GetEnvVa.PAS

GetFAttr (Prozedur) (Unit WinDos)

Liefert die Attribute einer Datei.

Deklaration

```
procedure GetFAttr(var F; var Attr: Word);
```

Anmerkungen

F ist eine Dateivariable (typisiert, untypisiert oder eine Texdatei). Die zugeordnete Datei darf nicht offen sein.

Siehe auch

GetFTime
SetFAttr
SetFTime

Beispielcode

Getfattr.PAS

GetFTime (Prozedur) (Unit WinDos)

Liefert Datum und Uhrzeit der letzten Veränderung einer Datei.

Deklaration

```
procedure GetFTime(var F; var Time: Longint);
```

Anmerkungen

F muß eine Dateivariable sein (typisiert, untypisiert oder eine Texdatei), die zwar zugewiesen, aber nicht geöffnet wurde.

Der in Time zurückgelieferte Wert ist "gepackt" und kann mit Hilfe von UnpackTime interpretiert werden.

Siehe auch

PackTime
SetFAttr
SetFTime
UnpackTime

Beispielcode

GetFTime.PAS

GetIntVec (Prozedur) (Unit WinDos)

Liefert den Inhalt eines Interrupt-Vektors.

Deklaration

```
procedure GetIntVec(IntNo: Byte; var Vector: Pointer);
```

Siehe auch

SetIntVec

Beispielcode

GetIntVc.PAS

GetMem (Prozedur)

Belegt einen Bereich von Size Bytes auf dem Heap und weist die Startadresse dieses Bereichs der Variablen P zu, erzeugt also eine dynamische Variable.

Deklaration

```
procedure GetMem(var P: Pointer; Size: Word);
```

Siehe auch:

Dispose
FreeMem
New

Beispielcode

FreeMem.PAS

GetTime (Prozedur) (Unit WinDos)

Liefert die Uhrzeit des Betriebssystems.

Deklaration

```
procedure GetTime(var Hour, Minute, Second, Sec100: Word);
```

Siehe auch

GetDate
SetDate
SetTime
UnPackTime

Beispielcode

GetTime.PAS

GetVerify (Prozedur) (Unit WinDos)

Liefert den Status des Verify--Flags von DOS.

Deklaration

```
procedure GetVerify(var Verify: Boolean);
```

Siehe auch

SetVerify

Beispielode

GetVerify.PAS

GotoXY (Prozedur) (Unit WinCrt)

Bewegt den Cursor zu den genannten Koordinaten innerhalb des virtuellen Bildschirmfensters.

Deklaration

```
procedure GotoXY(X, Y: Integer);
```

Anmerkungen

Die linke obere Ecke des virtuellen Bildschirmfensters hat die Koordinaten (1,1).

Verwenden Sie CursorTo anstelle von GotoXY , wenn Sie neue Anwendungen entwickeln. GotoXY wurde aufgenommen, um die Kompatibilität zur Unit Crt von Turbo Pascal für DOS zu wahren.

Beispielcode

GotoXY.PAS

Halt (Prozedur)

Bricht die Ausführung des Programms ab und führt einen Rücksprung zur DOS-Kommandoebene bzw. zu dem Programm durch, von dem aus es gestartet wurde.

Deklaration

```
procedure Halt [ ( Exitcode: Word ) ];
```

Exitcode ist ein optionaler Ausdruck, der den Code beim Verlassen des Programms angibt.

Siehe auch:

Exit
RunError

Beispielcode

Halt.PAS

Hi (Funktion)

Gibt das höherwertige Byte des Arguments zurück.

Deklaration

```
function Hi(X): Byte;
```

Siehe auch:

Lo
Swap

Beispielcode

Hi.PAS

Inc (Prozedur)

Erhöht eine Variable um 1 bzw. den durch N angegebenen Wert.

Deklaration

```
procedure Inc(var X [ ; N: Longint ] );
```

Siehe auch:

Dec
Pred
Succ

Beispielcode

Inc.PAS

Insert (Prozedur)

Fügt einen Substring in einen String ein.

Deklaration

```
procedure Insert(Source: String; var S: String; Index: Integer);
```

Siehe auch:

Concat
Copy
Delete
Length
Pos

Beispielcode

Insert.PAS

Int (Funktion)

Gibt den ganzzahligen Anteil des Arguments zurück.

Deklaration

```
function Int(X: Real): Real;
```

Siehe auch:

Frac
Round
Trunc

Beispielcode

Int.PAS

InitWinCrt (Prozedur) (Unit WinCrt)

Erstellt ein CRT-Fenster, sofern dieses nicht bereits erzeugt wurde.

Deklaration

```
procedure InitWinCrt;
```

Anmerkungen

InitWinCrt wird automatisch aufgerufen, wenn Sie Read, Readln, Write oder Writeln in einer Datei verwendet haben, die einem CRT-Fenster zugeordnet ist.

InitWinCrt verwendet die Konstanten **WindowOrg**, **WindowSize** und **ScreenSize** und die Variable **WindowTitle** zur Festlegung der Eigenschaften des CRT-Fensters.

Beispielcode

InitWCrt.PAS

Intr (Prozedur) (Unit WinDos)

Führt einen angegebenen Software-Interrupt aus.

Deklaration

```
procedure Intr(IntNo: Byte; var Regs: TRegisters);
```

wobei
- IntNo die Software-Interrupt-Nummer (0...255) angibt und
- TRegisters ein in DOS definierter Record ist.

Beschränkungen

Verwenden Sie keine Software-Interrupts, die
- beim Einsprung bestimmte Werte in SP oder SS voraussetzen oder
- beim Verlassen SP oder SS verändern.

Siehe auch

MsDos

Beispielcode

Intr.PAS

IOResult (Funktion)

Gibt den Fehlerstatus der letzten Ein-/Ausgabe-Operation zurück.

Deklaration

```
function IOResult: Word;
```

Rückgabewert

0, falls erfolgreich.

Anmerkungen

Ein Aufruf von IOResult setzt das interne Fehler-Flag zurück.

Beispielcode

IOResult.PAS

KeyPressed (Funktion) (Unit WinCrt)

Ermittelt, ob eine Taste auf der Tastatur gedrückt wurde.

Deklaration

```
function KeyPressed: Boolean;
```

Anmerkungen

Die Taste kann mit Hilfe der Funktion ReadKey gelesen werden.

Rückgabewert

True	Taste wurde gedrückt
False	Taste wurde nicht gedrückt

Beispielcode

KeyPress.PAS

Length (Funktion)

Gibt die aktuelle Länge des übergebenen Stringausdrucks zurück.

Deklaration

```
function Length(S: String): Integer;
```

Siehe auch:

Copy
Concat
Delete
Insert
Pos

Beispielcode

Length.PAS

Ln (Funktion)

Liefert den natürlichen Logarithmus des Arguments.

Deklaration

```
function Ln(X: Real): Real;
```

Siehe auch:

Exp

Beispielcode

Ln.PAS

Lo (Funktion)

Gibt das niederwertige Byte des Arguments zurück.

Deklaration

```
function Lo(X): Byte;
```

Siehe auch:

Hi
Swap

Beispielcode

Lo.PAS

LongDiv (Funktion) (Unit WObjects)

Deklaration

```
function LongDiv(X: Longint; Y: Integer): Integer;
inline($59/$58/$5A/$F7/$F9);
```

Anmerkungen

Diese schnelle Inline-Assembler-Routine führt eine Division aus und gibt den Integer-Wert `X/Y` zurück.

LongMul (Funktion) (Unit WObjects)

Deklaration

```
function longmul(X, Y: Integer): Longint;
inline($5A/$58/$f7/$EA);
```

Anmerkungen

Diese schnelle Inline-Assembler-Routine führt eine Multiplikation aus und gibt den LongInt-Wert `X*Y` zurück.

LowMemory (Funktion) (Unit WObjects)

Deklaration

```
function LowMemory: Boolean;
```

Anmerkungen

Die Funktion LowMemory gibt True zurück, wenn eine Speicherzuweisung den Sicherheitsbereich am Ende des Heap in Anspruch genommen hat.

Die Größe dieses Sicherheitsbereichs wird durch die Variable **SafetyPoolSize** festgelegt.

LowMemory wird automatisch von **TApplication.MakeWindow** und **TApplication.ExecDialog** aufgerufen, also den Funktionen, die zum Anlegen von Fensterelementen verwendet werden sollten.

Speicherintensive Elemente (wie beispielsweise große, komplexe Dialogfenster) sollten LowMemory selbst periodisch aufrufen, um sicherzustellen, daß sie den verfügbaren Speicherplatz nicht überschreiten.

Siehe auch

MemAlloc
TApplication.ValidWindow

MaxAvail (Funktion)

Gibt die Umfang des größten freien Blocks auf dem Heap in Bytes zurück.

Deklaration

```
function MaxAvail: Longint;
```

Anmerkungen

Gibt den größeren Wert von den folgenden zurück:
- dem größten freien Block des Unterbereichs, den der Heapmanager verwaltet
- dem globalen Heap von Windows

Dieser Wert entspricht der größten dynamischen Variablen, die zu diesem Zeitpunkt erzeugt werden kann.

Siehe auch:

MemAvail

Beispielcode

FreeMem.PAS

MemAlloc (Funktion) (Unit WObjects)

Deklaration

```
function MemAlloc(Size: Word): Pointer;
```

Anmerkungen

Reserviert Size Bytes Heapspeicher und gibt einen Zeiger auf den Speicherblock zurück. Falls kein Speicherblock der gewünschten Größe reserviert werden kann, wird **nil** zurückgegeben.

MemAlloc läßt keine Reservierung zu, die den Sicherheitsbereich des Heap angreift.

Ein durch MemAlloc reservierter Block kann mit Hilfe der Standardprozedur **FreeMem** freigegeben werden.

MemAvail (Funktion)

Liefert den Gesamtumfang des freien Platzes auf dem Heap in Bytes

Deklaration

```
function MemAvail: Longint;
```

Siehe auch:

MaxAvail

Beispielcode

Memavail.PAS

MkDir (Prozedur)

Erzeugt ein Unterverzeichnis.

Deklaration

```
procedure MkDir(S: String);
```

Siehe auch:

ChDir
GetDir
RmDir

Beispielcode

Mkdir.PAS

Move (Prozedur)

Kopiert Count Bytes aus dem durch Source angegebenen Speicherbereich in den durch Dest bezeichneten Bereich.

Deklaration

```
procedure Move(var Source, Dest; Count: Word);
```

Anmerkungen

Move führt keinerlei Bereichsprüfung durch.

Verwenden Sie möglichst **SizeOf**, um Count zu bestimmen.

Siehe auch:

FillChar

Beispielcode

Move.PAS

MsDos (Prozedur) (Unit WinDos)

Führt einen DOS-Funktionsaufruf aus.

Deklaration

```
procedure MsDos(var Regs: TRegisters);
```

Beschränkungen

Verwenden Sie keine Software-Interrupts, die
- beim Einsprung bestimmte Werte in SP oder SS voraussetzen oder
- beim Verlassen SP oder SS verändern.

Siehe auch

Intr

Beispielcode

MsDos.PAS

New (Prozedur)

Erzeugt eine neue dynamische Variable und setzt einen Zeiger darauf.

Deklaration

```
procedure New(var P: Pointer [ , Init: Constructor ]);
```

Anmerkungen

Die Syntax von New wurde so erweitert, daß man optional als zweiten Parameter einen **Konstruktor** angeben kann, um ein **Objekt** auf dem Heap zu initialisieren:

```
New(P, Init(360, 174));
```

New wurde noch in anderer Hinsicht erweitert. Man kann New jetzt auch als Funktion verwenden, die einen Zeiger zurückgibt. Der übergebene Parameter ist dann der Typ des Zeigers auf das Objekt und nicht die Zeigervariable selbst.

Diese Funktionsform von New kann auf alle Datentypen und nicht nur auf Objekttypen angewandt werden. Wie bei der Prozedur New kann als zweiter Parameter der Konstruktor eines Objekttyps übergeben werden.

Siehe auch:

Dispose
FreeMem
GetMem

Beispielcode

Dispose.PAS

Odd (Funktion)

Prüft, ob das Argument eine ungerade Zahl ist

Deklaration

```
function Odd(X: Longint): Boolean;
```

Beispielcode

Odd.PAS

Ofs (Funktion)

Gibt den Offset-Anteil der Adresse des angegebenen Objekts zurück.

Deklaration

function Ofs(X): Word;

Siehe auch:

Addr
Seg

Beispielcode

CSeg.PAS

Ord (Funktion)

Gibt den Ordinalwert des Arguments (das ein Ordinaltyp sein muß) zurück.

Deklaration

```
function Ord(X): Longint;
```

Siehe auch:

Chr

Beispielcode

Ord.PAS

PackTime (Prozedur) (Unit WinDos)

Konvertiert einen TDateTime-Record.

Deklaration

```
procedure PackTime(var T: DateTime; var Time: Longint);
```

Anmerkungen

Wandelt den Record in ein gepacktes Format um, einen 4 Bytes langen Long-Integer-Wert, der von SetFTime verwendet wird.

Siehe auch

GetFTime
GetTime
SetFTime
SetTime
UnpackTime

Beispielcode

GetFTime.PAS

ParamCount (Funktion)

Gibt die Anzahl der Kommandozeilen-Parameter zurück, die dem Programm übergeben wurden.

Deklaration

```
function ParamCount: Word;
```

Siehe auch:

ParamStr

Beispielcode

Parcount.PAS

ParamStr (Funktion)

Gibt den Kommandozeilen-Parameter [Index] zurück.

Deklaration

```
function ParamStr(Index): String;
```

Siehe auch:

ParamCount

Beispielcode

Parstrng.PAS

Pi (Funktion)

Gibt den Wert der mathematischen Konstanten Pi zurück, definiert als 3.1415926535897932385.

Deklaration

```
function Pi: Real;
```

Beispielcode

Pi.PAS

Pos (Funktion)

Sucht den als S übergebenen String-Ausdruck nach dem ersten Vorkommen des String-Ausdrucks Substr ab.

Deklaration

```
function Pos(Substr: String; S: String): Byte;
```

Siehe auch:

Delete
Concat
Copy
Insert
Length

Beispielcode

Pos.PAS

Pred (Funktion)

Gibt den Vorgänger des Arguments zurück.

Deklaration

```
function Pred(X): < Derselbe Typ wie das Argument >;
```

Siehe auch:

Dec
Inc
Succ

Beispielcode

Pred.PAS

Ptr (Funktion)

Konvertiert zwei Angaben für Segment und Offset in einen Wert des Typs Pointer.

Deklaration

```
function Ptr(Seg, Ofs: Word): Pointer;
```

Rückgabewert

Ein Zeiger auf die mit Seg und Ofs gegebene Adresse.

Siehe auch:

Addr

Beispielcode

Ptr.PAS

Random (Funktion)

Liefert eine Zufallszahl.

Deklaration

```
function Random [ ( Range: Word) ]: < Real oder Word >;
```

Anmerkungen

Um den Zahlengenerator Random zu initialisieren, rufen Sie Randomize auf oder weisen einen Wert an RandSeed zu.

Rückgabewert

Wenn Range angegeben wird, eine Word-Zufallszahl im Bereich 0 <= X < Range.
Sonst ein Real-Wert im Bereich 0 <= X < 1.

Siehe auch:

Randomize

Beispielcode

Random.PAS

Randomize (Prozedur)

Initialisiert den "Zufallsgenerator" mit einem Wert, der sich aus Datum und Uhrzeit des Systems ergibt.

Deklaration

```
procedure Randomize;
```

Siehe auch:

Random

Beispielcode

Random.PAS

Read (Prozedur)

- Bei typisierten Dateien: Liest eine Dateikomponente in eine Variable
- Bei Textdateien: Liest einen oder mehrere Werte in eine oder mehrere Variablen

Deklaration

Bei typisierten Dateien: `procedure Read(F , V1 [, V2,...,Vn ] );`
Bei Textdateien: `procedure Read( [ var F: Text; ] V1 [, V2,...,Vn ] );`

Anmerkungen

Bei einer Stringvariablen liest Read alle Zeichen bis ausschließlich der nächsten Zeilenendemarkierung oder bis Eof(f) True ist.

Read überspringt das Zeilenende nicht. Nach dem ersten Read stößt also jedes folgende Read auf das Zeilenende und gibt einen String der Länge 0 zurück. Aufeinanderfolgende Stringwerte können Sie mit mehreren Aufrufen von Readln lesen.

Wenn die erweiterte Syntax gilt, kann Read nullterminierte Strings auf Zeichen-Arrays, deren Index bei Null beginnt, lesen.

Siehe auch:

Readln
Write
Writeln

Beispielcode

Eof.PAS

ReadBuf (Funktion) (Unit WinCrt)

Liest eine Eingabezeile aus dem CRT-Fenster.

Deklaration

```
function ReadBuf(Buffer: PChar; Count: Word): Word;
```

Anmerkungen

Buffer zeigt auf einen Zeilenpuffer, der Platz für maximal Count Zeichen bietet. Count nennt die Anzahl der zu lesenden Zeichen.

Bis zu Count - 2 Zeichen können eingegeben werden. Eine Zeilenendemarkierung (die Sequenz #13#10) wird automatisch an die Zeile angefügt, sobald der Benutzer Enter drückt.

Rückgabewert

Anzahl der gelesenen Zeichen, einschließlich der Zeilenende- oder der Dateiendemarkierung.

Beispielcode

ReadBuf.PAS

Readln (Prozedur)

Führt einen Aufruf von Read aus und springt dann zur nächsten Zeile der Datei.

Deklaration

```
procedure Readln([ var F: Text; ] V1 [, V2, ...,Vn ]);
```

Siehe auch:

Read

Beispielcode

Readln.PAS

ReadKey (Funktion) (Unit WinCrt)

Liest ein Zeichen von der Tastatur.

Deklaration

```
function ReadKey: Char;
```

Anmerkungen

Das Zeichen wird nicht auf den Bildschirm ausgegeben.

Beispielcode

ReadKey.PAS

RegisterType (Prozedur) (Unit WObjects)

Deklaration

```
procedure RegisterType(var S: TStreamRec);
```

Anmerkungen

Jeder ObjectWindows-Objekttyp muß mit Hilfe dieser Methode registriert werden, bevor er in einer Stream-Ein-/Ausgabeoperation verwendet werden kann.

Die Standard-Objekttypen werden mit ObjType-Werten aus dem reservierten Bereich 0..99 vorregistriert.

RegisterType erzeugt einen Eintrag in einer verketteten List e von **TStreamRec**-Records.

Siehe auch

TStream.Get
TStream.Put

RemoveDir (Prozedur) (Unit WinDos)

Löscht ein leeres Unterverzeichnis.

Deklaration

```
procedure RemoveDir(Dir: PChar);
```

Siehe auch

GetCurDir
CreateDir
RmDir
SetCurDir

Beispielcode

RemoveDr.PAS

Rename (Prozedur)

Gibt einer externen Datei einen neuen Namen.

Deklaration

```
procedure Rename(var F; Newname);
```

Siehe auch:

Erase

Beispielcode

Rename.PAS

Reset (Prozedur)

Öffnet eine existierende Datei.

Deklaration

```
procedure Reset(var F [: File; Recsize: Word ] );
```

Siehe auch:

Append
Assign
Close
Rewrite
Truncate

Beispielcode

Reset.PAS

Rewrite (Prozedur)

Erzeugt eine neue Datei und öffnet sie.

Deklaration

```
procedure Rewrite(var F: File [; Recsize: Word ] );
```

Siehe auch:

Append
Assign
Reset
Truncate

Beispielcode

Rewrite.PAS

RmDir (Prozedur)

Löscht ein leeres Unterverzeichnis.

Deklaration

```
procedure RmDir(S: String);
```

Siehe auch:

ChDir
GetDir
MkDir
RemoveDir

Beispielcode

Rmdir.PAS

Round (Funktion)

Rundet das (Real-)Argument auf einen ganzzahligen Wert.

Deklaration

```
function Round(X: Real): Longint;
```

Siehe auch:

Int
Trunc

Beispielcode

Round.PAS

RunError (Prozedur)

Erzeugt einen Laufzeitfehler, der das Programm definiert abbricht.

Deklaration

```
procedure RunError [ ( Errorcode: Byte ) ];
```

Siehe auch:

Exit
Halt
Laufzeitfehlermeldungen

Beispielcode

Runerror.PAS

ScrollTo (Prozedur) (Unit WinCrt)

Rollt den Inhalt des CRT-Fensters so, daß sich der Punkt mit den virtuellen Koordinaten (X,Y) in der oberen linken Ecke befindet.

Deklaration

```
procedure ScrollTo(X, Y: Integer);
```

Anmerkungen

Die obere linke Ecke des virtuellen Bildschirms hat die Koordinaten (0,0).

Beispielcode

ScrollTo.PAS

Seek (Prozedur)

Setzt den Positionszeiger einer Datei auf die gegebene Komponente.

Deklaration

```
procedure Seek(var F; N: Longint);
```

Siehe auch:

FilePos

Beispielcode

FilePos.PAS

SeekEof (Funktion)

Überspringt Leerzeichen, Leerzeilen und Tabulatoren und prüft dann, ob das Dateiende erreicht ist.

Deklaration

```
function SeekEof [ (var F: Text) ]: Boolean;
```

Anmerkungen

Nur für offene Textdateien.

Siehe auch:

Eof
SeekEoln

Beispielcode

SeekEof.PAS

SeekEoln (Funktion)

Überspringt Leerzeichen und Tabulatoren und prüft dann, ob das Zeilenende erreicht ist.

Deklaration

```
function SeekEoln [ (var F: Text) ]: Boolean;
```

Anmerkungen

Nur für offene Textdateien.

Siehe auch:

Eoln
SeekEof

Beispielcode

SeekEof.PAS

Seg (Funktion)

Gibt die Segment-Anteil der Adresse des angegebenen Objekts zurück.

Deklaration

```
function Seg(X): Word;
```

Siehe auch:

Addr
Ofs

Beispielcode

CSeg.PAS

SetCBreak (Prozedur) (Unit WinDos)

Legt fest, bei welchen Operationen DOS prüft, ob Ctrl-Break gedrückt wurde.

Deklaration

```
procedure SetCBreak(Break: Boolean);
```

Siehe auch

GetCBreak

Beispielcode

GetCBrk.PAS

SetCurDir (Prozedur) (Unit WinDos)

Macht den angegebenen Pfad zum aktuellen Verzeichnis.

Deklaration

```
procedure SetCurDir(Dir: PChar)
```

Anmerkungen

Wenn Dir eine Laufwerksbezeichnung enthält, wird das Laufwerk ebenfalls gewechselt.

Siehe auch

GetCurDir
ChDir
CreateDir
RemoveDir

Beispielcode

SetCurDr.PAS

SetDate (Prozedur) (Unit WinDos)

Setzt das Systemdatum.

Deklaration

```
procedure SetDate(Year, Month, Day: Word);
```

Siehe auch

GetDate
GetTime
SetTime

Beispielcode

SetDate.PAS

SetFAttr (Prozedur) (Unit WinDos)

Setzt die Attribute einer Datei.

Deklaration

```
procedure SetFAttr(var F; Attr: Word);
```

Anmerkungen

Auftretende Fehler werden in **DosError** dokumentiert. Mögliche Fehlercodes sind
- 3 (Ungültiger Pfad)
- 5 (Dateizugriff verweigert)

Beschränkungen

Die Datei dafr nicht offen sein..

Siehe auch

GetFAttr
GetFTime
SetFTime

Beispielcode

Setfattr.PAS

SetFTime (Prozedur) (Unit WinDos)

Setzt Datum und Uhrzeit der letzten Veränderung einer Datei.

Deklaration

```
procedure SetFTime(var F; Time: Longint);
```

Anmerkungen

Auftretende Fehler werden in **DosError** dokumentiert. Dabei kann nur der Fehlercode 6 (ungültiges Datei-Handle) vorkommen.

Beschränkungen

Die Datei muß offen sein.

Siehe auch

GetFTime
PackTime
SetFAttr
UnpackTime

Beispielcode

GetFTime.PAS

SetIntVec (Prozedur) (Unit WinDos)

Setzt den angegebenen Interrupt-Vektor auf die angebene Adresse.

Deklaration

```
procedure SetIntVec(IntNo: Byte; Vector: Pointer);
```

Siehe auch

GetIntVec

Beispielcode

GetIntVc.PAS

SetTime (Prozedur) (Unit WinDos)

Setzt die Uhrzeit des Systems.

Deklaration

```
procedure SetTime(Hour, Minute, Second, Sec100: Word);
```

Siehe auch

GetDate
GetTime
PackTime
SetDate
UnPackTime

Beispielcode

SetTime.PAS

SetTextBuf (Prozedur)

Weist einer Textdatei-Variablen einen Puffer zu.

Deklaration

```
procedure SetTextBuf(var F: Text; var Buf [ ; Size:   Word ] );
```

Anmerkungen

SetTextBuf sollte nicht auf eine offene Datei angewendet werden, kann aber sofort nach **Reset**, **Rewrite** und **Append** aufgerufen werden.

Wenn Sie **SetTextBuf** bei einer offenen Datei nach vorangegangenen E/A-Operationen aufrufen, können wegen des Wechsels des Puffers Daten verloren gehen.

Beispielcode

Settxtbf.PAS

SetVerify (Prozedur) (Unit WinDos)

Setzt das Verify-Flag von DOS.

Deklaration

```
procedure SetVerify(Verify: Boolean);
```

Siehe auch

GetVerify

Beispielcode

GetVerify.PAS

Sin (Funktion)

Liefert den Sinus des Arguments

Deklaration

```
function Sin(X: Real): Real;
```

Siehe auch:

ArcTan
Cos

Beispielcode

Sin.PAS

SizeOf (Funktion)

Gibt die Anzahl von Bytes zurück, die das Argument an Speicherplatz belegt.

Deklaration

```
function SizeOf: Integer;
```

Anmerkungen

Wenn SizeOf auf eine Instanz eines **Objekt**typs mit einer VMT angewendet wird, dann liefert es die Größe, die in der VMT gespeichert ist.

Siehe auch:

constructor
virtual

Beispielcode

Sizeof.PAS

SPtr (Funktion)

Gibt den momentanen Wert des Stackzeigers (SP-Register) zurück.

Deklaration

```
function SPtr: Word;
```

Siehe auch:

Sseg

Beispielcode

CSeg.PAS

Sqr (Funktion)

Gibt das Quadrat des Arguments zurück.

Deklaration

```
function Sqr(X): < derselbe Typ wie das Argument >;
```

Siehe auch:

Sqrt

Beispielcode

Sqr.PAS

Sqrt (Funktion)

Gibt die Quadratwurzel des Arguments zurück.

Deklaration

```
function Sqrt(X: Real): Real;
```

Siehe auch:

Sqr

Beispielcode

Sqr.PAS

SSeg (Funktion)

Gibt die Adresse des Stack-Segments zurück, d.h. den Wert des SS-Registers.

Deklaration

```
function SSeg: Word;
```

Siehe auch:

CSeg
DSeg
SPtr

Beispielcode

CSeg.PAS

Str (Prozedur)

Konvertiert den übergebenen numerischen Wert in eine Zeichenfolge

Deklaration

```
procedure Str(X [: Width [: Decimals ]]; var S);
```

Anmerkungen

Erzeugt dieselbe Stringdarstellung, die Write ausgeben würde.

Siehe auch:

Val
Write (Text)
Write (typisiert)

Beispielcode

Str.PAS

StrCat (Funktion) (Unit Strings)

Kopiert einen String an das Ende eines anderen und gibt den verketteten String zurück.

Deklaration

```
function StrCat(Dest, Source: PChar): PChar;
```

Anmerkungen

StrCat führt keinerlei Bereichsprüfungen durch. Der in Dest angegebene Zielpuffer muß Platz für wenigstens StrLen(Dest)+StrLen(Source)+1 Zeichen bieten.

Siehe auch

StrLCat

Beispielcode

StrCat.PAS

StrComp (Funktion) (Unit Strings)

Vergleicht zwei Strings.

Deklaration

```
function StrComp(Str1, Str2 : PChar): Integer;
```

Rückgabewert

<0 wenn Str1 < Str2
=0 wenn Str1 = Str2
>0 wenn Str1 > Str2

Siehe auch

StrIComp
StrLComp
StrLIComp

Beispielcode

StrComp.PAS

StrCopy (Funktion) (Unit Strings)

Kopiert einen String auf einen anderen.

Deklaration

```
function StrCopy(Dest, Source: PChar): PChar;
```

Anmerkungen

StrCopy führt keinerlei Längenprüfung durch. Der in Dest angegebene Zielpuffer muß StrLen(Source)+1 Zeichen aufnehmen können.

Siehe auch

StrECopy
StrLCopy

Beispielcode

StrCopy.PAS

StrDispose (Prozedur) (Unit Strings)

Entfernt einen String aus dem Heap.

Deklaration

```
procedure StrDispose(Str: PChar);
```

Anmerkungen

StrDispose entfernt einen String, der zuvor mit **StrNew**. angelegt worden ist.

Wenn Str **nil** ist, hat StrDispose keine Auswirkungen.

Beispielcode

StrNew.PAS

StrECopy (Funktion) (Unit Strings)

Kopiert einen String auf einen anderen und gibt einen Zeiger auf das Ende des resultierenden Strings zurück.

Deklaration

```
function StrECopy(Dest, Source: Pchar): PChar;
```

Anmerkungen

StrECopy führt keinerlei Längenprüfung durch. Der mit Dest angegebene Zielpuffer muß StrLen(Source)+1 Zeichen aufnehmen können.

Siehe auch

StrCopy
StrEnd

Beispielcode

StrECopy.PAS

StrEnd (Funktion) (Unit Strings)

Liefert einen Zeiger auf das Ende des übergebenen Strings.

Deklaration

```
function StrEnd(Str: PChar): Pchar;
```

Siehe auch

StrLen

Beispielcode

StrEnd.PAS

StrIComp (Funktion) (Unit Strings)

Vergleicht zwei Strings ohne Beachtung von Groß- und Kleinschreibung.

Deklaration

```
function StrIComp(Str1, Str2:PChar): Integer;
```

Siehe auch

StrComp
StrLComp
StrLIComp

Beispielcode

StrIComp.PAS

StrLCat (Funktion) (Unit Strings)

Hängt Zeichen aus einem String an das Ende eines andern an und liefert den verketteten String.

Deklaration

```
function StrLCat(Dest, Source: PChar; MaxLen: Word): PChar;
```

Siehe auch

StrCat

Beispielcode

StrLCat.PAS

StrLComp (Funktion) (Unit Strings)

Vergleicht zwei Strings bis zu einer maximalen Anzahl von Zeichen.

Deklaration

```
function StrLComp(Str1, Str2: PChar; MaxLen: Word): Integer;
```

Siehe auch

StrComp
StrIComp
StrLIComp

Beispielcode

StrLComp.PAS

StrLCopy (Funktion) (Unit Strings)

Kopiert Zeichen von einem String in einen anderen.

Deklaration

```
function StrLCopy(Dest, Source: PChar; MaxLen: Word): PChar;
```

Siehe auch

StrCopy

Beispielcode

StrLCopy.PAS

StrLen (Funktion) (Unit Strings)

Liefert die Anzahl der Zeichen von Str.

Deklaration

```
function StrLen(Str: PChar): Word;
```

Siehe auch

StrEnd

Beispielcode

StrLen.PAS

StrLIComp (Funktion) (Unit Strings)

Vergleicht zwei Strings bis zu einer maximalen Anzahl Zeichen ohne Beachtung von Groß- und Kleinschreibung.

Deklaration

```
function StrLIComp(Str1, Str2: PChar; MaxLen: Word): Integer;
```

Anmerkungen

Der Rückgabewert ist derselbe wie bei **StrComp**

Siehe auch

StrComp
StrIComp
StrLComp

Beispielcode

StrComp.PAS

StrLower (Funktion) (Unit Strings)

Wandelt einen String in Kleinbuchstaben um.

Deklaration

```
function StrLower(Str: PChar): PChar;
```

Siehe auch

StrUpper

Beispielcode

StrUpper.PAS

StrMove (Funktion) (Unit Strings)

Kopiert Zeichen aus einem String in einen anderen.

Deklaration

```
function StrMove(Dest, Source: PChar; Count: Word): PChar;
```

Beispielcode

StrMove.PAS

StrNew (Funktion) (Unit Strings)

Legt einen String auf dem Heap an.

Deklaration

```
function StrNew(Str: PChar): PChar;
```

Anmerkungen

StrNew legt eine Kopie von Str auf dem Heap an. Dazu werden StrLen(Str) + 1 Bytes reserviert.

Wenn Str **nil** ist oder auf einen leeren String zeigt, dann gibt StrNew **nil** zurück und reserviert keinerlei Platz auf dem Heap.

Siehe auch

StrDispose

StrPas (Funktion) (Unit Strings)

Konvertiert einen null-terminierten String in einen Pascal-String.

Deklaration

```
function StrPas(Str: PChar): PChar;
```

Siehe auch

StrPCopy

Beispielcode

StrPas.PAS

StrPCopy (Funktion) (Unit Strings)

Kopiert einen Pascal-String auf einen null-terminierten String.

Deklaration

```
function StrPCopy(Dest: PChar; Source: String);
```

Anmerkungen

StrPCopy führt keinerlei Längenprüfung durch. Der in Dest angegebene Zielpuffer muß Length(Source)+1 Zeichen aufnehmen können.

Siehe auch

StrCopy

Beispielcode

StrPCopy.PAS

StrPos (Funktion) (Unit Strings)

Liefert einen Zeiger auf das erste Vorkommen eines Strings in einem anderen String.

Deklaration

```
function StrPos(Str1, Str2: PChar): PChar;
```

Rückgabewert

0 wenn Str2 nicht in Str1 vorkommt.

Beispielcode

StrPos.PAS

StrScan (Funktion) (Unit Strings)

Liefert einen Zeiger auf das erste Vorkommen eines Zeichens in einem String.

Deklaration

```
function StrScan(Str: PChar; Chr: Char): PChar;
```

Rückgabewert

0 wenn Chr nicht in Str vorkommt.

Siehe auch

StrRScan

Beispielcode

StrScan.PAS

StrRScan (Funktion) (Unit Strings)

Liefert einen Zeiger auf das letzte Vorkommen eines Zeichens in einem String.

Deklaration

```
function StrRScan(Str: PChar; Chr: Char): PChar;
```

Rückgabewert

0 wenn Chr nicht in Str vorkommt.

Siehe auch

StrScan

Beispielcode

StrRScan.PAS

StrUpper (Funktion) (Unit Strings)

Wandelt einen String in Großbuchstaben um.

Deklaration

```
function StrUpper(Str: PChar): PChar;
```

Siehe auch

StrLower

Beispielcode

StrUpper.PAS

Succ (Funktion)

Gibt den Nachfolger des Arguments zurück.

Deklaration

```
function Succ(X): < derselbe Typ wie das Argument >;
```

Siehe auch:

Inc
Pred

Beispielcode

Pred.PAS

Swap (Funktion)

Vertauscht das niederwertige und das höherwertige Byte des Arguments.

Deklaration

```
function Swap(X): (Same type as parameter);
```

Siehe auch:

Hi
Lo

Beispielcode

Swap.PAS

TrackCursor (Prozedur) (Unit WinCrt)

Rollt nötigenfalls den Inhalt des CRT-Fensters, damit der Cursor sichtbar bleibt.

Deklaration

```
procedure TrackCursor;
```

Beispielcode

TrackCur.PAS

Trunc (Funktion)

Wandelt einen Realwert durch Abschneiden der Nachkommastellen in einen Integerwert um.

Deklaration

```
function Trunc(X: Real): Longint;
```

Siehe auch:

Int
Round

Beispielcode

Trunc.PAS

Truncate (Prozedur)

Schneidet eine Datei an der aktuellen Position ab.

Deklaration

```
procedure Truncate(var F);
```

Siehe auch:

Reset
Rewrite
Seek

Beispielcode

Truncate.PAS

UnpackTime (Prozedur) (Unit WinDos)

Konvertiert Datum und Uhrzeit aus einem gepackten Format in einen Record des Typs TDateTime.

Deklaration

```
procedure UnpackTime(Time: Longint; var DT: TDateTime);
```

Anmerkungen

Das gepackte Format (ein Longint-Wert) wird von den Funktionen GetFTime, FindFirst, und FindNext zurückgegeben.

Siehe auch

GetFTime
GetTime
PackTime
SetFTime
SetTime

Beispielcode

GetFTime.PAS

UpCase (Funktion)

Konvertiert Klein- in Großbuchstaben. Die deutschen Umlaute werden nicht berücksichtigt.

Deklaration

```
function UpCase(Ch: Char): Char;
```

Beispielcode

UpCase.PAS

Val (Prozedur)

Interpretiert die Zeichenfolge des Strings S als numerischen Wert

Deklaration

```
procedure Val(S; var V; var Code: Integer);
```

mit:

S	Variable vom Typ String. Muß eine Zeichenfolge sein, die eine ganze Zahl mit Vorzeichen ergibt
V	Integer- oder Real-Variable
Code	Integer-Variable.

Siehe auch:

Str

Beispielcode

Val.PAS

WhereX (Funktion) (Unit WinCrt)

Liefert die X-Koordinate der aktuellen Cursor-Position.

Deklaration

```
function WhereX: Integer;
```

Rückgabewert

Der Ursprung hat die Koordinaten (1,1). Der Wert entspricht also Cursor.X + 1.

Beispielcode

WhereX.PAS

WhereY (Funktion) (Unit WinCrt)

Liefert die Y-Koordinate der aktuellen Cursor-Position.

Deklaration

```
function WhereY: Integer;
```

Anmerkungen

Der Ursprung hat die Koordinaten (1,1). Der Wert entspricht also Cursor.Y + 1.

Beispielcode

WhereY.PAS

Write (Prozedur)

- Typisierte Dateien: Schreibt eine Variable in eine Dateikomponente
- Textdateien: Schreibt einen oder mehr Werte in die Datei

Deklaration

Typisierte Dateien:
```
procedure Write(F, V1 [, V2,...,Vn ] );
```
Textdateien:
```
procedure Write([var F: Text;] V1 [,V2,...,Vn ]);
```

Siehe auch:

Read
Readln
Writeln

Beispielcode

Eof.PAS

WriteBuf (Prozedur) (Unit WinCrt)

Schreibt einen Block von Zeichen in ein CRT-Fenster.

Deklaration

```
procedure WriteBuf(Buffer: PChar; Count: Word);
```

Anmerkungen

Buffer zeigt auf das erste Zeichen des Blocks. Count enthält die Anzahl der zu schreibenden Zeichen.

Wenn das sogenannte AutoTracking aktiviert ist , wird der Inhalt des CRT-Fensters nötigenfalls gerollt, damit der Cursor auch nach der Schreiboperation sichtbar ist.

Beispielcode

WriteBuf.PAS

WriteChar (Prozedur) (Unit WinCrt)

Schreibt ein einzelnes Zeichen in ein CRT-Fenster.

Deklaration

```
procedure WriteChar(Ch: Char);
```

Beispielcode

WriteChr.PAS

Writeln (Prozedur)

Ruft Write (für Textdateien) auf und schreibt dann einen Zeilenvorschub in Datei F.

Deklaration

```
procedure Writeln([ var F: Text; ] V1 [, V2, ...,Vn ] );
```

Siehe auch:

Write (Text)
Write (typisiert)

Beispielcode

ReadIn.PAS

AccessResource (Windows API Funktion)

Deklaration

```
function AccessResource(Instance, ResInfo: THandle): Integer;
```

Beschreibung

Diese Funktion öffnet die angegebene Ressourcendatei und stellt den Dateizeiger auf die gewünschte Ressource, um der Anwendung das Lesen dieser Ressource zu ermöglichen. Die Funktion AccessResource unterstützt ein DOS-Datei-Handle, das in nachfolgenden Dateiaufrufen zum Laden der Ressource verwendet werden kann. Die Datei wird dann nur zum Lesen geöffnet.

Parameter

Instance	Bezeichnet die Instanz des Moduls, in dessen ausführbarer Datei die Ressource enthalten ist.
ResInfo	Bezeichnet die gewünschte Ressource. Dieses Handle sollte mit Hilfe der Funktion **FindResource** erzeugt werden.

Rückgabewert

Der Rückgabewert bestimmt das DOS-Datei-Handle zur vorgegebenen Ressourcendatei. Er ist -1, wenn die Ressource nicht gefunden werden kann.

AddAtom (Windows API Funktion)

Deklaration

```
function AddAtom(Str: PChar): Atom;
```

Beschreibung

Die durch den Parameter Str bezeichnete Zeichenkette wird von der Funktion in die Atom-Tabelle eingefügt. Die Funktion AddAtom speichert nur eine einzige Kopie des vorgegebenen Strings in der Atom-Tabelle. Existiert der String bereits in der Tabelle, gibt die Funktion den zugehörigen Wert zurück und erhöht den Referenzzähler des Strings um den Wert 1.

Parameter

Str	Zeigt auf die in die Tabelle einzufügende null-terminierte Zeichenkette.

Rückgabewert

War die Funktion erfolgreich, wird das erzeugte Atom zurückgegeben. Andernfalls ist der Rückgabewert -1.

Siehe auch:

GetAtomName

AddFontResource (Windows API Funktion)

Deklaration

```
function AddFontResource(FileName: PChar): Integer;
```

Beschreibung

Diese Funktion fügt die Schriftressource aus der durch den Parameter Filename bezeichneten Datei in die Schriftentabelle von Windows ein.

Parameter

Filename Handle zum geladenen Modul enthält oder ein null-terminierte String.

Rückgabewert

Der Rückgabewert bestimmt die Anzahl der hinzugefügten Schriften. Wurden keine Schriften geladen, ist er 0.

Siehe auch:

wm_FontChange

AdjustWindowRect (Windows API Prozedur)

Deklaration

```
procedure AdjustWindowRect(var Rect: TRect; Style: Longint; Menu: Bool);
```

Beschreibung

Diese Funktion berechnet die Größe des Fensters auf derGrundlage der gewünschten Größe Rect des Client-Bereichs. Setzt ein einzeiliges Menü voraus, sofern ein Menü vorgesehen ist.

Parameter

Rect	Zeigt auf eine **TRect**-Datenstruktur, die die Koordinaten des Client-Rechtecks enthält.
Style	Bestimmt den Typ des Fensters, dessen Client-Rechteck konvertiert werden soll.
Menu	Bestimmt, ob das Fenster ein Menü hat.

Siehe auch:

CreateWindow

AdjustWindowRectEx (Windows API Prozedur)

Deklaration

```
procedure AdjustWindowRectEx(var Rect: TRect; Style: Longint; Menu: Bool;
ExStyle: Longint);
```

Beschreibung

Diese Funktion berechnet die benötigte Größe eines Fensterrechtecks des erweiterten Typs aufGrundlage der gewünschten Größe Rect des Client-Rechtecks.
Die Größe des berechneten Fensterrechtecks hängt vom Fenstertyp und davon ab, ob das Fenster ein Menü hat. Setzt ein einzeiliges Menü voraus, sofern vorhanden.

Parameter

Rect	Zeigt auf eine **TRect**-Datenstruktur, die die Koordinaten des Client-Rechtecks enthält.
Style	Bestimmt den Typ des Fensters, dessen Client-Rechteck konvertiert werden soll.
Menu	Bestimmt, ob das Fenster ein Menü hat.
ExStyle	Bestimmt weitere Eigenschaften des zu erzeugenden Fensters.

Siehe auch:

CreateWindowEx

AllocResource (Windows API Funktion)

Deklaration

```
function AllocResource(Instance, ResInfo: THandle; Size: Longint):
THandle;
```

Beschreibung

Diese Funktion reserviert nicht-initialisierten Speicher für die übergebene Ressource ResInfo.

Parameter

Instance	Bzeichnet die Instanz des Moduls, in dessen sführbarer Datei die Ressource enthalten ist.
ResInfo	Bezeichnet die gewünschte Ressource.
Size	Bestimmt eine neudefinierte Größe in Bytes, die Neudefinition wird ignoriert, wenn die Größe 0 ist.

Rückgabewert

Bezeichnet den Speicherblock, der für die Ressource reserviert wurde.

Siehe auch:

FindResource
LoadResource

AnimatePalette (Windows API Prozedur)

Deklaration

```
procedure AnimatePalette(Palette: HPalette; StartIndex: Word; NumEntries:
Word; var PaletteColors);
```

Beschreibung

Ersetzt die Einträge zwischen StartIndex und NumEntries in der durch Palette bezeichneten virtuellen Farbpalette durch PaletteColors.

Parameter

Palette	Bezeichnet die virtuelle Farbpalette.
StartIndex	Bestimmt den ersten Eintrag in der zu verändernden Farbpalette.
NumEntries	Bestimmt die Anzahl der Einträge in der zu verändernden Farbpalette.
PaletteColors	Zeigt auf das erste Element eines Arrays von **TPaletteEntry**-Datenstrukturen, durch die die Einträge ersetzt werden sollen.

Siehe auch:

CreatePalette

AnsiLower (Windows API Funktion))

Deklaration

```
function AnsiLower(Str: PChar): PChar;
```

Beschreibung

Diese Funktion konvertiert die vorgegebene Zeichenkette Str in Kleinbuchstaben. Die Konvertierung wird durch den Sprachtreiber aufGrundlage der aktuell ausgewählten Sprache durchgeführt.

Parameter

Str	Zeigt auf einen null-terminierten String oder bestimmt ein einzelnes Zeichen (im niederwertigen Byte des niederwertigen Word).

Rückgabewert

Der Rückgabewert zeigt auf die konvertierte Zeichenkette.

AnsiLowerBuff (Windows API Funktion))

Deklaration

```
function AnsiLowerBuff(Str: PChar; Length: Word): Word;
```

Beschreibung

Diese Funktion konvertiert die Zeichenkette Str in Kleinbuchstaben. Die Konvertierung wird durch den Sprachtreiber aufGrundlage der aktuell ausgewählten Sprache durchgeführt.

Parameter

Str	Zeigt auf einen Puffer, der ein oder mehr Zeichen enthält.
Length	Bestimmt die Anzahl der Zeichen im Puffer. Bei 0, beträgt die Länge 64 KByte.

Rückgabewert

Der Rückgabewert bestimmt die Länge des konvertierten Strings.

AnsiNext (Windows API Funktion))

Deklaration

```
function AnsiNext(CurrentChar: PChar): PChar;
```

Beschreibung

Bewegt den Zeiger CurrentChar auf das nächste Zeichen eines Strings, dessen Zeichen aus mindestens zwei Bytes bestehen.

Parameter

CurrentChar	Zeigt auf ein Zeichen in einem null-terminierten String.

Rückgabewert

Zeiger auf das nächste Zeichen im String.

AnsiPrev (Windows API Funktion)

Deklaration

```
function AnsiPrev(Start, CurrentChar: PChar): PChar;
```

Beschreibung

Setzt den Zeiger CurrentChar auf das vorhergehende Zeichen in einem String, dessen Zeichen aus mindestens zwei Bytes bestehen.

Parameter

Start	Beginn des null-terminierten Strings
CurrentChar	Zeigt auf ein Zeichen des Strings

Rückgabewert

Zeiger auf das vorhergehende Zeichen im String.

AnsiToOem (Windows API Funktion))

Deklaration

```
function AnsiToOem(AnsiStr, OemStr: PChar): Integer;
```

Beschreibung

Diese Funktion übersetzt den durch AnsiStr bezeichneten String vom ANSI-Zeichensatz in den OEM-Zeichensatz. Der String kann länger als 64 KByte sein.

Parameter

AnsiStr	Zeigt auf einen null-terminierten String aus dem ANSI-Zeichensatz.
OemStr	Zeigt auf das Ziel, zu dem der String kopiert wird, der Parameter kann auch mit AnsiStr identisch sein.

Rückgabewert

Der Rückgabewert ist immer -1.

AnsiToOemBuff (Windows API Prozedur)

Deklaration

```
procedure AnsiToOemBuff(AnsiStr, OemStr: PChar; Length: Integer);
```

Beschreibung

Diese Funktion übersetzt den String im durch AnsiStr bezeichneten Puffer vom ANSI-Zeichensatz in den OEM-Zeichensatz.

Parameter

AnsiStr	Zeigt auf einen Puffer, der Zeichen des ANSI-Zeichensatzes enthält.
OemStr	zeigt auf das Ziel, zu dem der String kopiert wird, kann mit dem Parameter AnsiStr identisch sein.
Length	Anzahl der Zeichen in AnsiStr, bei 0 beträgt die Länge 64 KByte.

AnsiUpper (Windows API Funktion)

Deklaration

```
function AnsiUpper(Str: PChar): PChar;
```

Beschreibung

Diese Funktion konvertiert die vorgegebene Zeichenkette Str in Großbuchstaben. Die Konvertierung wird durch den Sprachtreiber aufGrundlage der aktuell ausgewählten Sprache durchgeführt.

Parameter

Str Zeigt auf einen null-terminierten String oder bestimmt ein einzelnes Zeichen (im niederwertigen Byte des niederwertigen Word).

Rückgabewert

Der Rückgabewert zeigt auf die konvertierte Zeichenkette, wenn der Funktionsparameter eine Zeichenkette ist. Andernfalls ist der Rückgabewert ein 32-Bit-Wert, der das konvertierte Zeichen im niederwertigen Byte des niederwertigen Word enthält.

AnsiUpperBuff (Windows API Funktion))

Deklaration

```
function AnsiUpperBuff(Str: PChar; Length: Word): Word;
```

Beschreibung

Diese Funktion konvertiert die Zeichenkette Str im Puffer in Großbuchstaben. Die Konvertierung wird durch den Sprachtreiber aufGrundlage der aktuell ausgewählten Sprache durchgeführt.

Parameter

Str	Zeigt auf einen Puffer, der ein oder mehr Zeichen enthält.
Length	Bestimmt die Anzahl der Zeichen im Puffer, bei 0 beträgt die Länge 64 KByte.

Rückgabewert

Der Rückgabewert bestimmt die Länge des konvertierten Strings.

AnyPopup (Windows API Funktion)

Deklaration

```
function AnyPopup: Bool;
```

Beschreibung

Diese Funktion zeigt an, ob ein Pop-up-Fenster auf dem Bildschirm existiert. Die Funktion hat keine Parameter.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn ein Pop-up-Fenster existiert, sonst 0.

AppendMenu (Windows API Funktion)

Deklaration

```
function AppendMenu(Menu: HMenu; Flags, IDNewItem: Word; NewItem: PChar):
Bool;
```

Beschreibung

Diese Funktion fügt einen neuen Eintrag am Ende eines Menüs an. Anwendungen können den Status des Eintrags durch die Werte im Parameter Flags bestimmen.

Parameter

Menu	Bezeichnet das zu verändernde Menü.
Flags	Enthält Informationen über den Status des hinzugefügten Menüeintrags. Er besteht aus einem oder mehreren der folgenden **mf_xxx** Konstanten: mf_Bitmap mf_Checked mf_Disabled mf_Enabled mf_Grayed mf_MenuBarBreak mf_MenuBreak mf_OwnerDraw mf_Popup mf_Separator mf_String mf_Unchecked
IDNewItem	Bestimmt entweder die Befehls-ID des neuen Menüeintrags oder das Handle des Pop-up-Menüs.
NewItem	Bestimmt den Inhalt des neuen Menüeintrags, also den Zeiger auf einen String oder, falls ein Bitmap den Menüeintrag bildet, enthält das niederwertige Word das Handle des Bitmaps.

Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, ist er ungleich 0, andernfalls0.

Siehe auch:

DrawMenuBar
SetMenuItemBitmaps
wm_DrawItem
wm_MeasureItem

Arc (Windows API Funktion)

Deklaration

```
function Arc(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet einen elliptischen Kreisbogen. Der Mittelpunkt des Kreisbogens entspricht dem Mittelpunkt des umgebenden Rechtecks.

Parameter

DC	Bezeichnet den Gerätekontext.
X1, Y1	Bestimmt die virtuellen Koordinaten der oberen linken Ecke des umgebenden Rechtecks.
X2, Y2	Bestimmt die virtuellen Koordinaten der unteren rechten Ecke des umgebenden Rechtecks.
X3, Y3	Bestimmt die virtuellen Koordinaten des Startpunkts des Kreisbogens. Dieser Punkt muß nicht genau auf dem Kreisbogen liegen.
X4, Y4	Bestimmt die virtuellen Koordinaten des Endpunkts des Kreisbogens. Dieser Punkt muß nicht genau auf dem Kreisbogen liegen.

Rückgabewert

Ein Rückgabewert ungleich 0 gibt an, daß der Kreisbogen gezeichnet wird.

Hinweis

Die Breite des Rechtecks wird aus X2 - X1 berechnet und darf 32767 Einheiten nicht übersteigen. Diese Grenze gilt auch für die Höhe.

ArrangeIconicWindows (Windows API Funktion)

Deklaration

```
function ArrangeIconicWindows(Wnd: HWnd): Word;
```

Beschreibung

Diese Funktion ordnet die Symbole aller Fenster an, die durch den Parameter bezeichnet werden.

Parameter

Wnd Bezeichnet das Fenster.

Rückgabewert

Der Rückgabewert ist die Höhe einer Symbolzeile. Er ist 0, wenn keine Symbole gefunden werden.

Siehe auch:

GetDesktopWindow

BeginPaint (Windows API Funktion)

Deklaration

```
function BeginPaint(Wnd: HWnd; var Paint: TPaintStruct): HDC;
```

Beschreibung

Diese Funktion bereitet das bezeichnete Fenster zum Zeichnen vor. Anwendungen sollten die Funktion nur als Reaktion auf die Botschaft **wm_Paint** aufrufen. Dem Parameter Paint werden die Zeicheninformationen übergeben.

Parameter

Wnd	Bezeichnet das neu zu malende Fenster.
Paint	Zeigt auf eine **TPaintStruct** Datenstruktur, die die Zeicheninformationen aufnehmen soll.

Rückgabewert

Bezeichnet den Gerätekontext des vorgegebenen Fensters.

Siehe auch:

EndPaint
InvalidateRect
InvalidateRgn
wm_EraseBkgnd
wm_Paint

BitBlt (Windows API Funktion)

Deklaration

```
function BitBlt(DestDC: H
DC; X, Y, Width, Height: Integer; SrcDC: HDC; XSrc, YSrc: Integer; Rop:
Longint): Bool;
```

Beschreibung

Kopiert ein Bitmap von dem durch SrcDC bezeichneten Quellgerät zum durch DestDC vorgegeben Zielgerät.

Parameter

DestDC	Bezeichnet den Gerätekontext, der das Bitmap aufnehmen soll.
X, Y	Bestimmt die virtuellen Koordinaten der oberen linken Ecke des Zielrechtecks.
Width	Bestimmt die Breite des Zielrechtecks und des Quell-Bitmaps.
Height	Bestimmt die Höhe des Zielrechtecks und des Quell-Bitmaps.
SrcDC	Bezeichnet den Gerätekontext, aus dem das Bitmap kopiert wird. Hat den Wert 0, wenn eine Rasteroperation auf DestDC festlegt, die keine Quelle einschließt.
XSrc, YScr	Bestimmt die virtuellen Koordinaten der oberen linken Ecke des Quell-Bitmaps.
Rop	Konstante, die die durchzuführende **ternäre Rasteroperation** festlegt, und einen einfachen Kopiervorgang von Quell- nach Ziel-Bitmap ausführt.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Bitmap dargestellt wird. Andernfalls ist der Rückgabewert 0.

BringWindowToTop (Windows API Prozedur)

Deklaration

```
procedure BringWindowToTop(Wnd: HWnd);
```

Beschreibung

Bringt das durch Wnd bezeichnete untergeordnete Pop-up-Fenster in den Vordergrund eines Stapels überlappender Fenster. Zusätzlich werden Pop-up-Fenster und Hauptfenster aktiviert.

Parameter

Wnd	Bezeichnet das untergeordnete Pop-up-Fenster.

BuildCommDCB (Windows API Funktion)

Deklaration

```
function BuildCommDCB(Def: PChar; var DCB: TDCB): Integer;
```

Beschreibung

Diese Funktion übersetzt den durch Def bezeichneten Definitions-String in entsprechende Gerätesteuerblockcodes und kopiert diese in den durch DCB bezeichneten Block.

Parameter

Def	Zeigt auf einen null-terminierten String, die Steuerinformationen für ein Gerät enthält. Der String muß das Format des DOS MODE Befehlszeilenparameters haben.
DCB	Zeigt auf eine **TDCB**-Datenstruktur, die den übersetzten String aufnehmen soll.

Rückgabewert

Der Rückgabewert ist 0, wenn Def übersetzt wurde und negativ, wenn ein Fehler auftritt.

Siehe auch:

SetCommState

CallMsgFilter (Windows API Funktion)

Deklaration

```
function CallMsgFilter(var Msg: TMsg; Code: Integer): Bool;
```

Beschreibung

Diese Funktion übergibt die mit Msg gegebene Botschaft an die aktuelle Botschaftsfilterfunktion.

Parameter

Msg	Zeigt auf eine **TMsg**-Datenstruktur, die die zu filternde Botschaft enthält.
Code	Gibt einen Code an, der von der Filterfunktion benutzt wird.

Rückgabewert

Der Rückgabewert gibt den Zustand der Botschaftsbearbeitung an. Er ist 0, wenn die Botschaft bearbeitet werden soll, im anderen Fall ungleich 0.

Siehe auch:

SetWindowsHook

CallWindowProc (Windows API Funktion)

Deklaration

```
function CallWindowProc(PrevWndProc: TFarProc; Wnd: HWnd; Msg, wParam:
Word; lParam: Longint): Longint;
```

Beschreibung

Diese Funktion übergibt Botschaftsinformationen an dieRoutine, die durch den Parameter PrevWndProc angegeben wird, und erlaubt die Klassifizierung von Fenter-Unterklassen. Eine Unterklasse ist ein Fenster oder eine Menge von Fenstern, die zur selben Fensterklasse gehören, deren Botschaften aber von einer anderen Funktion abgefangen und bearbeitet werden, bevor sie an die Fensterfunktion dieser Klasse übergeben werden.

Parameter

PrevWndProc	Enthält die Adresse der Prozedurinstanz der vorhergehenden Fensterfunktion.
Wnd	Bezeichnet das Fenster, das die Botschaft empfängt.
Msg	Gibt die Botschaftsnummer an.
wParam	Gibt zusätzliche botschaftsabhängige Informationen an.
lPraram	Gibt zusätzliche botschaftsabhängige Informationen an.

Rückgabewert

Der Rückgabewert gibt das Ergebnis der Bearbeitung der Botschaft durch PrevWndProc an.

Siehe auch:

SetWindowLong

Catch (Windows API Funktion)

Deklaration

```
function Catch(var CatchBuf: TCatchBuf): Integer;
```

Beschreibung

Kopiert den Zustand aller Systemregister und des Befehlszählers in den Puffer, auf den der Parameter CatchBuf zeigt.

Parameter

CatchBuf Zeigt auf die **TCatchBuf** -Datenstruktur, die die Informationen zur Ausführungsumgebung aufnimmt.

Rückgabewert

Der Rückgabewert 0 gibt an, daß die Ausführungsumgebung in den Puffer kopiert wurde.

Siehe auch:

Throw

ChangeClipboardChain (Windows API Funktion)

Deklaration

```
function ChangeClipboardChain(Wnd, WndNext: HWnd): Bool;
```

Beschreibung

Diese Funktion entfernt das durch den Parameter Wnd angegebene Fenster aus der Kette der Clipboard-Viewer und setzt das Fenster, das durch den Parameter WndNext angegeben wird, anstelle des entfernten Fensters ein.

Parameter

Wnd	Bezeichnet das Fenster, das aus der Kette enfernt werden soll.
WndNext	Bezeichnet das Fenster, das Wnd in der Kette der Clipboard-Viewer folgt.

Rückgabewert

Der Rückgabewertungleich 0, wenn das Fenster gefunden und entfernt wurde. Andernfalls ist er 0.

Siehe auch:

SetClipboardViewer
wm_ChangeCBChain

ChangeSelector (Windows API Funktion)

Deklaration

```
function ChangeSelector(DestSelector, SourceSelector: Word): Word;
```

Beschreibung

Verändert die Attribute eines Selektors von Code in Daten oder umgekehrt. Der Wert des Selektors bleibt dabei unverändert.

Diese Funktion sollte nur in Ausnahmefällen verwendet werden. Der konvertierte Selektor sollte unmittelbar nach der Umwandlung verwendet werden, da es keine Möglichkeit zur Synchronisation des Quell- und des Zielselektors gibt.

Parameter

DestSelector	Bezeichnet den Selektor, der den konvertierten Selektor aufnimmt, muß zuvor mit **AllocSelector** zugewiesen worden sein.
SourceSelector	Der zu konvertierende Selektor.

Rückgabewert

Konvertierter Selektor oder 0, wenn die Konvertierung gescheitert ist.

CheckDlgButton (Windows API Prozedur)

Deklaration

```
procedure CheckDlgButton(Dlg: HWnd; IDButton: Integer; Check: Word);
```

Beschreibung

Diese Funktion markiert einen Schalter, entfernt eine Markierung oder ändert den Zustand eines dreistufigen Schalters.

Parameter

Dlg	Bezeichnet das Dialogfenster, das den Schalter enthält.
IDButton	Gibt den Schalter an, der modifiziert werden soll.
Check	Gibt die vorzunehmende Handlung an. Wenn der Parameter gleich 1 ist, wird der Schalter markiert, wenn er gleich 0 ist, wird die Markierung entfernt, ist er gleich 2, wird damit ein nicht-aktiver Schalter angezeigt.

CheckMenuItem (Windows API Funktion)

Deklaration

```
function CheckMenuItem(Menu: HMenu; IDCheckItem, Check: Word): Bool;
```

Beschreibung

Diese Funktion plaziert oder entfernt eine Markierung neben einer Menüoption in einem Pop-up-Menü.

Parameter

Menu	Bezeichnet das Menü.
IDCheckItem	Gibt die Menüoption an, die bearbeitet werden soll.
Check	Gibt an, wie die Menüoption markiert werden soll und wie die Position der Option im Menü festgelegt werden soll. Der Parameter kann eine Kombination der Flags **mf_xxx**sein: mf_ByCommand mf_ByPosition mf_Checked mf_Unchecked

Rückgabewert

Der Rückgabewert gibt den momentanen Zustand der Option an. Er ist -1, wenn die Menüoption nicht vorhanden ist.

CheckRadioButton (Windows API Prozedur)

Deklaration

```
procedure CheckRadioButton(Dlg: HWnd; IDFirstButton, IDLastButton,
IDCheckButton: Integer);
```

Beschreibung

Diese Funktion markiert das Schaltfeld, das durch den Parameter IDCheckButton angegeben wird und entfernt die Auswahlmarkierung von allen anderen Schaltfeldern in derGruppe von Schaltern, die durch die Parameter IDFirstButton und IDLastButton angegeben werden.

Parameter

Dlg	Bezeichnet das Dialogfenster.
IDFirstButton	Gibt den Integer-Bezeichner des ersten Schaltfelds in derGruppe an.
IDLastButton	Gibt den Integer-Bezeichner des letzten Schaltfelds in derGruppe an.
IDCheckButton	Gibt den Integer-Bezeichner des Schaltfelds an, das markiert werden soll.

ChildWindowFromPoint (Windows API Funktion)

Deklaration

```
function ChildWindowFromPoint(WndParent: HWnd; APoint: TPoint): HWnd;
```

Beschreibung

Diese Funktion überprüft, welches der durch WndParent bezeichneten Fenster den mit APoint angegebenen Punkt enthält.

Parameter

WndParent	Bezeichnet das übergeordnete Fenster.
APoint	Gibt in einer **TPoint**-Datenstruktur die Client-Koordinaten des Punktes an, der getestet werden soll.

Rückgabewert

Bezeichnet das untergeordnete Fenster, das den Punkt enthält. Er ist 0, wenn der übergebene Punkt außerhalb des übergeordneten Fensters liegt. Wenn der Punkt innerhalb des übergeordneten Fensters, aber nicht in einem der untergeordneten Fenster liegt, wird WndParent zurückgegeben.

Chord (Windows API Funktion)

Deklaration

```
function Chord(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet eine Region, die durch den Schnittpunkt einer von einem Rechteck umgebenen Ellipse und eines Liniensegments begrenzt wird.

Parameter

DC	Bezeichnet den Gerätekontext, in dem der elliptische Kreisbogen erscheinen wird.
X1, Y1	Gibt die Koordinaten der oberen linken Ecke des begrenzenden Rechtecks an.
X2, Y2	Gibt die Koordinaten der unteren rechten Ecke des begrenzenden Rechtecks an.
X3, Y3	Gibt die Koordinaten eines Endes des Liniensegments an.
X4, Y4	Gibt die Koordinaten eines Endes des Liniensegments an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der Bogen gezeichnet wurde. Andernfalls ist er 0.

ClearCommBreak (Windows API Funktion)

Deklaration

```
function ClearCommBreak(Cid: Integer): Integer;
```

Beschreibung

Diese Funktion stellt eine Zeichenübertragung wieder her und versetzt die Übertragungsleitung in einen nicht zu unterbrechenden Zustand.

Parameter

Cid — Gibt die Schnittstelle an, die angesprochen werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist negativ, wenn der Parameter Cid keine gültige Geräteeinheit ist.

Siehe auch:

OpenComm

ClientToScreen (Windows API Prozedur)

Deklaration

```
procedure ClientToScreen(Wnd: HWnd; var Point: TPoint);
```

Beschreibung

Diese Funktion wandelt die Client-Koordinaten des mit APoint gegebenen Punktes auf dem Bildschirm in Bildschirmkoordinaten um.

Parameter

Wnd	Bezeichnet das Fenster, dessen Client-Bereich für die Umwandlung verwendet wird.
APoint	Zeigt auf eine **TPoint**-Datenstruktur, die die umzuwandelnden Client-Koordinaten enthält.

ClipCursor (Windows API Prozedur)

Deklaration

```
procedure ClipCursor(Rect: LPRect);
```

Beschreibung

Diese Funktion schränkt den Cursor auf das Rechteck auf dem Bildschirm ein, das durch den Paramteter Rect übergeben wird. Wenn Rect **nil** ist, darf der Cursor auf dem Bildschirm frei bewegt werden.

Parameter

Rect	Zeigt auf eine **TRect**-Datenstruktur, die die Bildschirmkoordinaten der oberen linken und der unteren rechten Ecke des eingrenzenden Rechtecks enthält.

Siehe auch:

SetCursorPos

CloseClipboard (Windows API Funktion)

Deklaration

```
function CloseClipboard: Bool;
```

Beschreibung

Diese Funktion schließt die Zwischenablage. Sie ermöglicht dadurch anderen Anwendungen den Zugriff auf die Zwischenablage.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Zwischenablage geschlossen ist. Andernfalls ist er 0.

CloseComm (Windows API Funktion)

Deklaration

```
function CloseComm(Cid: Integer): Integer;
```

Beschreibung

Diese Funktion schließt die Schnittstelle, die durch den Parameter Cid angegeben wird, und gibt den Speicher frei, der für die Übertragungs- und Empfangsschlangen derGeräteeinheit belegt wurde. Alle Zeichen in der Ausgabeschlange werden gesendet, bevor die Schnittstelle geschlossen wird.

Parameter

Cid	Gibt die Geräteeinheit an, die geschlossen werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn die Schnittstelle geschlossen wurde. Er ist negativ, wenn ein Fehler aufgetreten ist.

Siehe auch:

OpenComm

CloseMetaFile (Windows API Funktion)

Deklaration

```
function CloseMetaFile(DC: THandle): THandle;
```

Beschreibung

Diese Funktion schließt den Metadatei-Gerätekontext DC und erzeugt ein Metadatei-Handle, das verwendet werden kann, um die Metadatei abzuspielen.

Parameter

DC	Bezeichnet den Metadatei-Gerätekontext, der geschlossen werden soll.

Rückgabewert

Bezeichnet der Metadatei, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist der Rückgabewert 0.

Siehe auch:

PlayMetaFile

CloseSound (Windows API Prozedur)

Deklaration

```
procedure CloseSound;
```

Beschreibung

Die Funktion löscht alle Warteschlangen, gibt alle Puffer frei, die reserviert wurden, und schließt den Zugriff auf den Lautsprecher.

CloseWindow (Windows API Prozedur)

Deklaration

```
procedure CloseWindow(Wnd: HWnd);
```

Beschreibung

Diese Funktion verkleinert das mit Wnd angegebene Fenster. Wenn das Fenster andere überlappt, werden Client-Bereich und Überschrift des geöffneten Fensters vom Bildschirm entfernt wird und das Symbol in den Symbolbereich des Bildschirms verschoben.

Parameter

Wnd	Bezeichnet das Fenster, das verkleinert werden soll.

CombineRgn (Windows API Funktion)

Deklaration

```
function CombineRgn(DestRgn, SrcRgn1, SrcRgn2: HRgn; CombineMode:
Integer): Integer;
```

Beschreibung

Diese Funktion erzeugt eine neue Region, indem sie die zwei mit SrcRgn1 und ScrRgn2 gegebenen Regionen verbindet. Wie diese Regionen kombiniert werden, wird durch den Parameter CombineMode angegeben.

Parameter

DestRgn	Bezeichnet eine Region, die durch die neue Region ersetzt wird.
SrcRgn1	Bezeichnet eine existierende Region.
SrcRgn2	Bezeichnet eine existierende Region.
CombineMode	Eine der **rgn_xxx Flags**

Rückgabewert

Der Rückgabewert gibt den Typ der entstandenen Region an, ist eine der Konstanten vom Typ **Region-Flags**.

CopyMetaFile (Windows API Funktion)

Deklaration

```
function CopyMetaFile(SrcMetaFile: THandle; FileName: PChar): THandle;
```

Beschreibung

Diese Funktion kopiert die mit ScrMetaFile gegebene Quell-Metadatei in die Datei, auf die der Parameter Filename zeigt.

Parameter

SrcMetaFile	Bezeichnet die Quell-Metadatei.
Filename	Zeigt auf einen null-terminierten String, der die Datei angibt, die die Metadatei aufnehmen soll, oder ist 0, wenn in den Speicher kopiert werden soll.

Rückgabewert

Bezeichnet die neue Metadatei.

CopyRect (Windows API Prozedur)

Deklaration

```
procedure CopyRect(var DestRect, SourceRect: TRect);
```

Beschreibung

Diese Funktion kopiert das Rechteck, auf das der Parameter SourceRect zeigt, in die Datenstruktur, auf die der Parameter DestRect zeigt.

Parameter

DestRect	Zeigt auf eine **TRect**-Datenstruktur.
SourceRect	Zeigt auf eine Trect-Datenstruktur.

CountClipboardFormats (Windows API Funktion)

Deklaration

```
function CountClipboardFormats: Integer;
```

Beschreibung

Diese Funktion liefert die Anzahl der Formate, die von der Zwischenablage verarbeitet werden können.

Rückgabewert

Der Rückgabewert gibt die Anzahl von Datenformaten in der Zwischenablage an.

CountVoiceNotes (Windows API Funktion)

Deklaration

```
function CountVoiceNotes(Voice: Integer): Integer;
```

Beschreibung

Diese Funktion liefert die Anzahl der Töne in der mit Voice angegebenen Warteschlange.

Parameter

Voice	Gibt die Warteschlange an.

Rückgabewert

Der Rückgabewert gibt die Anzahl der unterschiedlichen Töne in der gegebenen Warteschlange an.

Siehe auch:

SetVoiceNote

CreateBitmap (Windows API Funktion)

Deklaration

```
function CreateBitmap(Width, Height: Integer; Planes, BitCount: Byte;
Bits: Pointer): HBitmap;
```

Beschreibung

Diese Funktion erzeugt ein geräteabhängiges Bitmap, das für eine Speicheranzeige ausgewählt werden kann.

Parameter

Width	Gibt die Breite (in Pixel) des Bitmaps an.
Height	Gibt die Höhe (in Pixel) des Bitmaps an.
Planes	Gibt die Anzahl der Farbebenen im Bitmap an.
BitCount	Gibt die Anzahl von Farbbits je Bildschirmpixel an.
Bits	Zeigt auf ein Short-Integer-Array, das die ursprünglichen Bitwerte des Bitmaps enthält. Wenn er **nil** ist, wird das neue Bitmap nicht initialisiert.

Rückgabewert

Bezeichnet des Bitmaps, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Siehe auch:

BitBlt
SelectObject

CreateBitmapIndirect (Windows API Funktion)

Deklaration

```
function CreateBitmapIndirect(var Bitmap: TBitmap): HBitmap;
```

Beschreibung

Diese Funktion erzeugt ein Bitmap, dessen Breite, Höhe und Bitmuster durch die Datenstruktur, auf die der Parameter Bitmap zeigt, vorgegeben werden.

Parameter

Bitmap Zeigt auf eine **TBitmap**-Datenstruktur

Rückgabewert

Bezeichnet des Bitmaps, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Siehe auch:

BitBlt

CreateBrushIndirect (Windows API Funktion)

Deklaration

```
function CreateBrushIndirect(var LogBrush: TLogBrush): HBrush;
```

Beschreibung

Diese Funktion erzeugt einen virtuellen Pinsel, dessen Stil, Farbe und Muster durch die Datenstruktur vorgegeben werden, auf die der Parameter LogBrush zeigt.

Parameter

LogBrush Zeigt auf eine **TLogBrush**-Datenstruktur, die Informationen über den Pinsel enthält.

Rückgabewert

Bezeichnet des virtuellen Pinsels, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreateCaret (Windows API Prozedur)

Deklaration

```
procedure CreateCaret(Wnd: HWnd; ABitmap: HBitmap; Width, Height:
Integer);
```

Beschreibung

Diese Funktion erzeugt eine neue Form für das System-Caret.

Parameter

Wnd	Bezeichnet das Fenster, welches das neue Caret besitzt.
Bitmap	Legt die Form des Carets. Wenn der Parameter gleich 0, wird das Caret mit Farbe ausgefüllt; wenn er gleich 1 ist, ist das Caret nicht aktivierbar und wird grau dargestellt.
Width	Gibt die Breite des Carets an (in virtuellen Einheiten).
Height	Gibt die Höhe des Carets an (in virtuellen Einheiten).

Siehe auch:

CreateBitmap
CreateDIBitmap
GetSystemMetrics
LoadBitmap

CreateCompatibleBitmap (Windows API Funktion)

Deklaration

```
function CreateCompatibleBitmap(DC: HDC; Width, Height: Integer): HBitmap;
```

Beschreibung

Diese Funktion erzeugt ein Bitmap, das mit derGeräteeinheit, die durch den Parameter DC angegeben wird, kompatibel ist.

Parameter

DC	Bezeichnet den Gerätekontext.
Width	Gibt die Breite (in Bits) des Bitmaps an.
Height	Gibt die Höhe (in Bits) des Bitmaps an.

Rückgabewert

Bezeichner eines Bitmaps, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreateCompatibleDC (Windows API Funktion)

Deklaration

```
function CreateCompatibleDC(DC: HDC): HDC;
```

Beschreibung

Diese Funktion erzeugt einen Speichergerätekontext, der kompatibel mit derGeräteeinheit ist, die durch den Parameter DC angegeben wird.

Parameter

DC	Bezeichnet den Gerätekontext. Wenn der Parameter gleich 0 ist, erzeugt die Funktion einen Speichergerätekontext.

Rückgabewert

Bezeichnet des neuen Speichergerätekontexts, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

Siehe auch:

DeleteDC
GetDeviceCaps

CreateCursor (Windows API Funktion)

Deklaration

```
function CreateCursor(Instance: THandle; Xhotspot, Yhotspot, Width,
Height: Integer; ANDBitPlane, XORBitPlane: Pointer): HCursor;
```

Beschreibung

Diese Funktion erzeugt einen Cursor mit angegebener Breite, Höhe und Bitmuster.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, das den Cursor erzeugt.
Xhotspot, Yhotspot	Gibt die Position des Aktionspunktes des Cursors an.
Width	Gibt die Breite des Cursors in Pixeln an.
Height	Gibt die Höhe des Cursors in Pixeln an.
ANDbitPlane	Zeigt auf ein Byte-Array, das die Bitwerte für die **AND**-Maske des Cursors enthält.
XORbitPlane	Zeigt auf ein Byte-Array, das die Bitwerte für die **XOR**-Maske des Cursors enthält.

Rückgabewert

Bezeichnet des Cursors, wenn die Funktion erfolgreich ausgeführt wurde, andernfalls ist er 0.

CreateDC (Windows API Funktion)

Deklaration

```
function CreateDC(DriverName, DeviceName, Output: PChar; InitData:
Pointer): HDC;
```

Beschreibung

Diese Funktion erzeugt einen Gerätekontext für die mit DriverName angegebene Geräteeinheit.

Parameter

DriverName	Zeigt auf einen null-terminierten String, der den DOS-Dateinamen (ohne Namenserweiterung) desGerätetreibers angibt.
DeviceName	Zeigt auf einen null-terminierten String, der den Namen der Geräteeinheit angibt, die unterstützt werden soll.
Output	Zeigt auf einen null-terminierten String, der den DOS-Dateinamen oder Gerätenamen für das physikalische Ausgebemedium angibt.
InitData	Zeigt auf eine DEVMODE-Datenstruktur, die die Initialisierungsdaten für den Gerätetreiber enthält.

Rückgabewert

Bezeichner einesGerätekontextes für die angegebene Geräteeinheit, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateDialog (Windows API Funktion)

Deklaration

```
function (Instance: THandle; TemplateName: PChar; WndParent: HWnd;
DialogFunc: TFarProc): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein nicht-modales Dialogfenster, das die Größe, den Stil und die Steuerelemente besitzt, die durch die Dialogfenster-Ressource definiert werden, die vom Parameter TemplateName vorgegeben wird.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Ressource des Dialogfenster enthält.
TemplateName	Zeigt auf einen null-terminierten String, der die Ressource des Dialogfensters benennt.
WndParent	Bezeichnet das Fenster, welches das Dialogfenster besitzt.
DialogFunc	Adresse der Prozedurinstanz der Dialogfunktion oder **nil**, falls als Klasse definiert

Rückgabewert

Handle des Dialogfensters oder 0, wenn die Funktion das Dialogfenster nicht erzeugen kann.

Siehe auch:

DefDlgProc
MakeProcInstance
wm_InitDialog

CreateDialogIndirect (Windows API Funktion)

Deklaration

```
function CreateDialogIndirect(Instance: THandle; DialogTemplate: PChar;
Parent: HWnd; DialogFunc: TFarProc): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein nicht-modales Dialogfenster, dessen Größe, Stil und Steuerelemente durch die **Dialogfenstervorlage** definiert werden, die durch den Parameter DialogTemplate vorgegeben wird.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Ressource des Dialogfenster enthält.
DialogTemplate	Zeigt auf einen String, der den Namen der Dialogfenstervorlage enthält.
WndParent	Bezeichnet das Fenster, welches das Dialogfenster besitzt.
DialogFunc	Ist die Adresse der Prozedurinstanz der Dialogfunktion oder **nil**, wenn als Klasse definiert.

Rückgabewert

Handle des Dialogfensters oder 0, wenn die Funktion weder das Dialogfenster noch irgendein Steuerelement im Dialogfenster erzeugen kann.

Siehe auch:

DefDlgProc
MakeProcInstance
wm_InitDialog

CreateDialogIndirectParam (Windows API Funktion)

Deklaration

```
function CreateDialogIndirectParam(Instance: THandle;var DialogTemplate;
WndParent: HWnd; DialogFunc: TFarProc; InitParam: Longint): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein nicht-modales Dialogfenster, dessen Größe, Stil und Steuerelemente durch die **Dialogfenstervorlage** definiert werden. Im Unterschied zu CreateDialogIndirect erlaubt diese Funktion die Übergabe eines Parameters, InitParam, an die Callback-Funktion. Im übrigen ist diese Funktion identisch mit der Funktion CreateDialogIndirect.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Ressource des Dialogfensters enthält.
DialogTemplate	Zeigt auf einen String, der den Namen der Dialogfenstervorlage enthält.
WndParent	Bezeichnet das Fenster, welches das Dialogfenster besitzt.
DialogFunc	Ist die Adresse der Prozedurinstanz der Dialogfunktion oder **nil**, wenn als Klasse definiert.
InitParam	Ist ein 32-Bit Wert (lParam der **wm_InitDialog** Botschaft) , den die Funktion der Dialogfunktion übergibt, wenn sie das Dialogfenster erzeugt.

Rückgabewert

Handle des Dialogfensters oder 0, wenn die Funktion weder das Dialogfenster noch irgendein Steuerelement im Dialogfenster erzeugen kann.

Siehe auch:

DefDlgProc
MakeProcInstance
wm_InitDialog

CreateDialogParam (Windows API Funktion)

Deklaration

```
function CreateDialogParam(Instance: THandle; TemplateName: PChar;
WndParent: HWnd; DialogFunc: TFarProc; InitParam: Longint): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein nicht-modales Dialogfenster, dessen Eigenschaften durch die in TemplateName angegebene Datenstruktur definiert werden.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Ressource des Dialogfenster enthält.
TemplateName	Zeigt auf einen null-terminierten String, der den Namen der Dialogfenstervorlage enthält.
Parent	Bezeichnet das Fenster, welches das Dialogfenster besitzt.
DialogFunc	Ist die Adresse der Prozedurinstanz der Dialogfunktion oder **nil**, wenn als Klasse definiert.
InitParam	Ist ein 32-Bit Wert (lParam der **wm_InitDialog** Botschaft), den die Funktion der Dialogfunktion übergibt, wenn sie das Dialogfenster erzeugt.

Rückgabewert

Handle des Dialogfensters oder 0, wenn die Funktion weder das Dialogfenster noch irgendein Steuerelement im Dialogfenster erzeugen kann.

Siehe auch:

DefDlgProc
MakeProcInstance

CreateDIBitmap (Windows API Funktion)

Deklaration

```
function CreateDIBitmap(DC: HDC; var InfoHeader: TBitmapInfoHeader; Usage:
Longint; InitBits: PChar; var InitInfo: TBitmapInfo; Usage: Word):
HBitmap;
```

Beschreibung

Diese Funktion erzeugt ein gerätespezifisches Speicher-Bitmap aus einer durch InfoHeader und InitInfo gegebenen geräteunabhängigen Bitmap-Beschreibung und setzt optional Bits im Bitmap.

Parameter

DC	Bezeichnet den Gerätekontext.
InfoHeader	Zeigt auf eine **TBitmapInfoHeader**-Struktur, die Größe und Format des geräteunabhängigen Bitmaps beschreibt.
Usage	Wenn der Parameter auf cbm_Init gesetzt ist, wird das Bitmap mit den Bits, die von InitBits und InitInfo angegeben werden, initialisiert.
InitBits	Zeigt auf ein Byte-Array, das die ursprünglichen Bitmap-Werte enthält, das Format wird durch das Feld biBitCount von InitInfo bestimmt.
InitInfo	**TBitmapInfo**-Datenstruktur, die die Größe und das Farbformat beschreibt.
Usage	Eine der **DIB_xxx Konstanten**

Rückgabewert

Bezeichner eines Bitmaps wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateDIBPatternBrush (Windows API Funktion)

Deklaration

```
function CreateDIBPatternBrush(PackedDIB: THandle; Usage: Word): HBitmap;
```

Beschreibung

Diese Funktion erzeugt einen virtuellen Pinsel mit dem Muster, das vom durch den Parameter bezeichneten PackedDIB geräteunabhängigen Bitmap (DIB) definiert wird.

Parameter

PackedDIB	Bezeichnet ein globales Speicherobjekt, das eine **TBitmapInfo**-Datenstruktur und ein Byte-Array enthält, das die Pixel des Bitmaps definiert.
Usage	Eine der **DIB_xxx Konstanten**

Rückgabewert

Bezeichner eines virtuellen Pinsels, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch:

GetDeviceCaps

CreateDiscardableBitmap (Windows API Funktion)

Deklaration

```
function CreateDiscardableBitmap(DC: HDC; Width, Height: Integer):
HBitmap;
```

Beschreibung

Diese Funktion erzeugt ein verwerfbares Bitmap, das mit der durch den Parameter DC bezeichneten Geräteeinheit kompatibel ist.

Parameter

DC	Bezeichnet einen Gerätekontext.
Width	Gibt die Breite (in Bits) des Bitmap an.
Height	Gibt die Höhe (in Bits) des Bitmap an.

Rückgabewert

Bezeichner eines Bitmaps, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateEllipticRgn (Windows API Funktion)

Deklaration

```
function CreateEllipticRgn(X1, Y1, X2, Y2: Integer): HRgn;
```

Beschreibung

Diese Funktion erzeugt eine elliptische Region, die durch das bezeichnete Rechteck begrenzt wird.

Parameter

X1, Y1	Gibt die Koordinaten der oberen linken Ecke des begrenzenden Rechtecks der Ellipse an.
X2, Y2	Gibt die Koordinaten der unteren rechten Ecke des begrenzenden Rechtecks der Ellipse an.

Rückgabewert

Bezeichner einer neuen Region, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateEllipticRgnIndirect (Windows API Funktion)

Deklaration

```
function CreateEllipticRgnIndirect(var Rect: TRect): HRgn;
```

Beschreibung

Diese Funktion erzeugt eine elliptische Region, die durch das in ARect definierte Rechteck begrenzt wird.

Parameter

ARect — Zeigt auf eine **TRect**-Datenstruktur, die die Koordinaten der oberen linken und unteren rechten Ecke des Begrenzungsrechtecks der Ellipse enthält.

Rückgabewert

Bezeichner einer neuen Region, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateFont (Windows API Funktion)

Deklaration

```
function CreateFont(Height, Width, Escapement, Orientation, Weight:
Integer; Italic, Underline, StrikeOut, CharSet, OutputPrecision,
ClipPrecision, Quality, PitchAndFamily: Byte; FaceName: PChar): HFont;
```

Beschreibung

Diese Funktion erzeugt eine virtuelle Schrift, die aus den imGDI verfügbaren physikalischen Schriften entsprechend der angegebenen Merkmale ausgewählt wird.

Parameter

Height	Gibt die gewünschte Höhe (in virtuellen Einheiten) der Schrift an.
Width	Gibt die durchschnittliche Breite (in virtuellen Einheiten) von Zeichen in der Schrift an.
Escapement	Gibt den Winkel (in Zehntelgraden) jeder Textzeile an, die in der Schrift geschrieben wird (relativ zur unteren Grenze der Seite).
Orientation	Gibt den Winkel (in Zehntelgraden) der unteren Grenze jeden Zeichens an (relativ zur unteren Grenze der Seite).
Weight	Gibt die gewünschte Schriftstärke im Bereich von 0 bis 1000 an, oder setzt ein **fw_xxx Flag**, wie fw_Normal oder fw_Bold.
Italic	Gibt an, ob die Schrift kursiv ist.
Underline	Gibt an, ob die Schrift unterstrichen ist.
StrikeOut	Gibt an, ob die Zeichen der Schrift durchgestrichen dargestellt werden.
CharSet	Gibt den gewünschten Zeichensatz an,über eine der **Schriftzeichensatz-Flags**
OutputPrecision	Gibt die gewünschte Ausgabepräzision an über eine der **out_xxx Konstanten**.
ClipPrecision	Gibt die gewünschte Clipping- Präzision an über eine der **clip_xxx Konstanten**.
Quality	Gibt die gewünschte Ausgabequalität an über eine der **Schriftausgabequalitäts-Flags** .
PitchAndFamily	Eine der **Schrift-Pitch-Flags** Konstanten kombiniert mit einer der **ff_xxx Flags**
Facename	Zeigt auf die null-terminierte Zeichenkette, die den Schriftartnamen der gewünschten Schriften festlegt.

Rückgabewert

Bezeichner einer virtuellen Schrift, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch:

EnumFonts

CreateFontIndirect (Windows API Funktion)

Deklaration

```
function CreateFontIndirect(var LogFont: TLogFont): HFont;
```

Beschreibung

Diese Funktion erzeugt eine virtuelle Schrift mit den Merkmalen, die von der Datenstruktur, auf die der Parameter ALogFont zeigt, vorgegeben werden.

Parameter

ALogFont — Zeigt auf eine **TLogFont**-Datenstruktur, die die Merkmale der virtuellen Schrift bestimmt.

Rückgabewert

Bezeichnet der virtuellen Schrift, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateHatchBrush (Windows API Funktion)

Deklaration

```
function CreateHatchBrush(Index: Integer; Color: TColorRef): HBrush;
```

Beschreibung

Diese Funktion erzeugt einen virtuellen Pinsel mit der angegebenen Schraffur und Farbe.

Parameter

Index	Gibt den Schraffurstil an, eine **hs_xxx** Konstante
Color	Gibt die Farbe der Schraffur an, zeigt auf eine **TColorRef**-Datenstruktur.

Rückgabewert

Bezeichnet des virtuellen Pinsels, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateIC (Windows API Funktion)

Deklaration

```
function CreateIC(DriverName, DeviceName, Output, InitData: PChar): HDC;
```

Beschreibung

Diese Funktion erzeugt einen Informationskontext für die angegebene Geräteeinheit.

Parameter

DriverName	Zeigt auf einen null-terminierten String, der den DOS-Dateinamen (ohne Namenserweiterung) desGerätetreibers angibt.
DeviceName	Zeigt auf einen null-terminierten String, der den Namen der Geräteeinheit angibt, die unterstützt werden soll.
Output	Zeigt auf einen null-terminierten String, der den DOS-Dateinamen oder Gerätenamen für das physikalische Ausgabemedium angibt.
InitData	Zeigt auf gerätespezifische Initialisierungsdaten für den Gerätetreiber oder ist **nil**, wenn derGerätetreiber die voreingestellte Initialisierung verwenden soll.

Rückgabewert

Bezeichnet des Informationskontextes für die angegebene Geräteeinheit, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateIcon (Windows API Funktion)

Deklaration

```
function CreateIcon(Instance: THandle; Width, Height: Integer; Planes,
BitsPixel: Byte; ANDbits, XORbits: Pointer): HIcon;
```

Beschreibung

Diese Funktion erzeugt ein Symbol, das die angegebene Breite, Höhe und Bitmuster hat.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, das das Symbol erzeugt hat.
Width	Gibt die Breite des Symbols in Pixeln an.
Height	Gibt die Höhe des Symbols in Pixeln an.
Planes	Gibt die Anzahl der Ebenen in der **XOR**-Maske des Symbols an.
BitsPixel	Gibt die Anzahl der Bits pro Pixel in der **XOR**-Maske des Symbols an.
ANDbits	Zeigt auf ein Byte-Array, das die Bitwerte für die **AND**-Maske des Symbols enthält. Dieses Array muß eine monochrome Maske bezeichnen.
XORbits	Zeigt auf ein Byte-Array, das die Bitwerte für die **XOR**-Maske des Symbols enthält.

Rückgabewert

Bezeichnet des Symbols, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateMenu (Windows API Funktion)

Deklaration

```
function CreateMenu: HMenu;
```

Beschreibung

Diese Funktion erzeugt ein Menü, das anfangs leer ist.

Rückgabewert

Bezeichnet des neu erzeugten Menüs Er ist 0, wenn das Menü nicht erzeugt werden kann.

Siehe auch:

AppendMenu
InsertMenu

CreateMetaFile (Windows API Funktion)

Deklaration

```
function CreateMetaFile(FileName: PChar): THandle;
```

Beschreibung

Diese Funktion erzeugt einen Metadatei-Gerätekontext.

Parameter

Filename — Zeigt auf einen null-terminierten String, der den Namen der Metadatei angibt oder ist **nil**, wenn eine Speicher-Metadatei bezeichnet werden soll.

Rückgabewert

Bezeichnet des Metadatei-Gerätekontextes, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreatePalette (Windows API Funktion)

Deklaration

```
function CreatePalette(var LogPalette: TLogPalette): HPalette;
```

Beschreibung

Diese Funktion erzeugt eine virtuelle Farbpalette.

Parameter

LogPalette Zeigt auf eine **TLogPalette**-Datenstruktur, die Informationen über die Farben der virtuellen Palette enthält.

Rückgabewert

Bezeichner einer virtuellen Palette, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreatePatternBrush (Windows API Funktion)

Deklaration

```
function CreatePatternBrush(Bitmap: HBitmap): HBrush;
```

Beschreibung

Diese Funktion erzeugt einen virtuellen Pinsel, der das Muster hat, das vom Parameter Bitmap angegeben wird.

Parameter

Bitmap **HBitmap** Datenstruktur, die Bezeichner des Bitmaps enthält.

Rückgabewert

Bezeichner eines virtuellen Pinsels, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch:

CreateBitmap
CreateBitmapIndirect
LoadBitmap
CreateCompatibleBitmap
DeleteObject
GetDeviceCaps

CreatePen (Windows API Funktion)

Deklaration

```
function CreatePen(PenStyle, Width: Integer; Color: TColorRef): HPen;
```

Beschreibung

Diese Funktion erzeugt einen virtuellen Stift, der den Stil, die Breite und Farbe hat, die angegeben werden. Der Stift kann später als aktuellen Stift für jede Geräteeinheit ausgewählt werden.

Parameter

PenStyle	Gibt den Stil des Stifts an und nimmt den Wert einer der **ps_xxx Konstanten** an.
Width	Gibt die Breite des Stifts (in virtuellen Einheiten) an.
Color	Gibt in einer **TColorRef** die Farbe des Stifts an.

Rückgabewert

Bezeichner eines virtuellen Stifts, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreatePenIndirect (Windows API Funktion)

Deklaration

```
function CreatePenIndirect(var LogPen: TLogPen): HPen;
```

Beschreibung

Diese Funktion erzeugt einen virtuellen Stift, der den Stil, die Breite und Farbe hat, die von der Datenstruktur vorgegeben werden, auf die der Parameter LogPen zeigt.

Parameter

LogPen Zeigt auf die **TLogPen**-Datenstruktur, die Informationen über den virtuellen Stift enthält.

Rückgabewert

Bezeichner eines virtuelles Stiftobjektes wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreatePolygonRgn (Windows API Funktion)

Deklaration

```
function CreatePolygonRgn(var Points; Count, PolyFillMode: Integer): HRgn;
```

Beschreibung

Diese Funktion erzeugt eine Polygon-Region.

Parameter

Points — Zeigt auf ein Array von **TPoint**-Datenstrukturen. Jeder Punkt gibt die Koordinaten eines Scheitelpunktes des Polygons an.

Count — Gibt die Anzahl von Punkten im Array an.

PolyFillMode — Gibt den Polygon-Füllmodus an, der zum Füllen der Region benutzt werden soll, verwendet eine der **PolyFillModes**-Konstanten

Rückgabewert

Bezeichner einer neuen Region, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch:

SetPolyFillMode

CreatePopupMenu (Windows API Funktion)

Deklaration

```
function CreatePopupMenu: HMenu;
```

Beschreibung

Diese Funktion erzeugt ein leeres Popup-Menü.

Rückgabewert

Bezeichnet des Menüs, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch:

AppendMenu
InsertMenu
TrackPopupMenu

CreateRectRgn (Windows API Funktion)

Deklaration

```
function CreateRectRgn(X1, Y1, X2, Y2: Integer): HRgn;
```

Beschreibung

Diese Funktion erzeugt eine rechteckige Region, die durch das angegebene Rechteck begrenzt wird.

Parameter

X1, Y1	Gibt die Koordinaten der oberen linken Ecke des begrenzenden Rechtecks an.
X2, Y2	Gibt die Koordinaten der unteren rechten Ecke des begrenzenden Rechtecks an.

Rückgabewert

Bezeichner einer neue Region, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateRectRgnIndirect (Windows API Funktion)

Deklaration

```
function CreateRectRgnIndirect(var Rect: TRect): HRgn;
```

Beschreibung

Diese Funktion erzeugt eine rechteckige Region, die durch das in ARect gegebene Rechteckt begrenzt wird.

Parameter

ARect Zeigt auf eine TRect-Datenstruktur, die die Koordinaten der oberen linken und der unteren rechten Ecke der Region enthält.

Rückgabewert

Bezeichner einer neuen Region, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateRoundRectRgn (Windows API Funktion)

Deklaration

```
function CreateRoundRectRgn(X1, Y1, X2, Y2, X3, Y3: Integer): HRgn;
```

Beschreibung

Diese Funktion erzeugt eine rechteckige Region mit abgerundeten Ecken.

Parameter

X1, Y1	Gibt die Koordinaten der oberen Ecke der Region an.
X2, Y2	Gibt die Koordinaten der unteren rechten Ecke der Region an.
X3,	Gibt die Breite der Ellipse an, die verwendet wird, um die abgerundeten Ecken zu erzeugen.
Y3	Gibt die Höhe der Ellipse an, die verwendet wird, um die abgerundeten Ecken zu erzeugen.

Rückgabewert

Bezeichner einer neuen Region, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateSolidBrush (Windows API Funktion)

Deklaration

```
function CreateSolidBrush(Color: TColorRef): HBrush;
```

Beschreibung

Diese Funktion erzeugt einen virtuellen Pinsel, der die angegebene Zeichenfarbe hat. Der Pinsel kann später für jede Geräteeinheit als aktueller Pinsel gewählt werden.

Parameter

Color — Gibt als Datenstruktur **TColorRef** die Farbe des Pinsels an.

Rückgabewert

Bezeichner eines virtuellen Pinsels, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

CreateWindow (Windows API Funktion)

Deklaration

```
function CreateWindow(ClassName, WindowName: PChar; Style: Longint; X, Y,
Width, Height: Integer; WndParent: HWnd; Menu: HMenu; Instance: THandle;
Param: Pointer): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein überlappendes, ein Pop-up- oder ein untergeordnetes Fenster.

Parameter

ClassName eigt auf einen null-terminierten String, der die Fensterklasse benennt,
oder den Name einer vordefinierten Steuerelementeklasse.

WindowName Zeigt auf einen null-terminierten String, der den Fensternamen darstellt.

Style Gibt den Stil des Fensters an, das erzeugt werden soll. Er enthält eine oder eine beliebige Kombination der Fenster- oder Steuerelement-Stilkonstanten:

bs_xxx
cbs_xxx
ds_xxx
es_xxx
lbs_xxx
sbs_xxx
ss_xxx
ws_xxx

X, YGibt die ursprüngliche Position der linken, oberen Fensterecke (in Bildschirmkoordinaten) an oder hat den Wert **cw_UseDefault**.

Width Gibt die ursprüngliche Breite (in Geräteeinheiten) des Fensters an.

Height Gibt die ursprüngliche Höhe (in Geräteeinheiten) des Fensters an.

WndParent Bezeichnet das übergeordnete oder das besitzende Fenster.

Menu Bezeichnet ein Menü oder einen Bezeichner eines untergeordneten Fensters.

Instance Bezeichnet die Instanz des Moduls, das dem Fenster zugeordnet werden soll.

Param Zeigt auf einen Wert, der dem Fenster in der Datenstruktur **TCreateStruct** übergeben wird, auf die der Parameter lParam der Botschaft **wm_Create** zeigt. Es muß sich hierbei um einen Zeiger auf eine **TClientCreateStruct-Datenstruktur** zur Erzeugung eines MDI-Client-Fensters handeln.

Rückgabewert

Bezeichnet des neuen Fensters oder 0, wenn das Fenster nicht erzeugt ist.

Siehe auch:

RegisterClass
wm_Create
wm_GetMinMaxInfo
wm_NCCreate

CreateWindowEx (Windows API Funktion)

Deklaration

```
function CreateWindowEx(ExStyle: Longint; ClassName, WindowName: PChar;
Style: Longint; X, Y, Width, Height: Integer; WndParent: HWnd; Menu:
HMenu; Instance: THandle; Param: Pointer): HWnd;
```

Beschreibung

Diese Funktion erzeugt ein überlapptes, ein Pop-up- oder ein untergeordnetes Fenster mit einem erweiterten Stil, der im Parameter ExStyle angegeben wird.

Parameter

ExStyle	Gibt den erweiterten Stil des Fensters an, das erzeugt wird. Hat den Wert einer der **ws_ex_xxx** Konstanten.
ClassName	Zeigt auf einen null-terminierten String, der den Namen der Fensterklasse enthält.
WindowName	Zeigt auf einen nullterminierten String, der den Fensternamen darstellt.
Style	Gibt den Stil des Fensters an, das erzeugt werden soll. Er enthält eine oder eine beliebige Kombination der Fenster- oder Steuerelement-Stilkonstanten: **bs_xxx** **cbs_xxx** **ds_xxx** **es_xxx** **lbs_xxx** **sbs_xxx** **ss_xxx** **ws_xxx**
X, Y	Gibt die ursprüngliche Position der linken, oberen Fensterecke (in Bildschirmkoordinaten) an oder hat den Wert **cw_UseDefault**.
Width	Gibt die ursprüngliche Breite (in Geräteeinheiten) des Fensters an.
Height	Gibt die ursprüngliche Höhe (in Geräteeinheiten) des Fensters an.
WndParent	Bezeichnet das übergeordnete oder das besitzende Fenster.
Menu	Bezeichnet ein Menü oder einen Bezeichner eines untergeordneten Fensters.
Instance	ezeichnet die Instanz des Moduls, das dem Fenster zugeordnet werden soll.
Param	Zeigt auf einen Wert, der dem Fenster in der Datenstruktur **TCreateStruct** übergeben wird, auf die der Parameter lParam der Botschaft **wm_Create** zeigt. Es muß sich hierbei um einen Zeiger auf eine **TClientCreateStruct-Datenstruktur** zur Erzeugung eines MDI-Client-Fensters handeln.

Rückgabewert

Bezeichnet des neuen Fensters oder 0, wenn das Fenster nicht erzeugt wird.

Siehe auch:

CreateWindow
wm_ParentNotify

DebugBreak (Windows API Prozedur)

Deklaration

```
procedure DebugBreak;
```

Beschreibung

Erzwingt die Übergabe eines Break-Signals an den Debugger.

DefDlgProc (Windows API Funktion)

Deklaration

```
function DefDlgProc(Dlg: HWnd; Msg, wParam: Word; lParam: Longint):
Longint;
```

Beschreibung

Diese Funktion ermöglicht die Bearbeitung aller Windows-Botschaften, die von Dialogfenstern mit einer privaten Klassennicht bearbeitet werden.

Parameter

Dlg	Bezeichner des Dialogfensters.
Msg	Nummer der Botschaft.
wParam	16 Bits zusätzlicher, botschaftsabhängiger Informationen
lParam	32 Bits zusätzlicher, botschaftsabhängiger Informationen

Rückgabewert

Der Rückgabewert enthält das Ergebnis der bearbeiteten Botschaft.

DefFrameProc (Windows API Funktion)

Deklaration

```
function DefFrameProc(Wnd, MDIClient: HWnd; Msg, wParam: Word; lParam:
Longint): Longint;
```

Beschreibung

Diese Funktion ermöglicht die Bearbeitung von Windows-Botschaften, die von der Fensterfunktion eines MDI-Rahmenfensters nicht bearbeitet werden.

Parameter

Wnd	Bezeichner des MDI-Rahmenfensters
MDIClient	Bezeichner des MDI-Client-Fensters
Msg	Nummer der Botschaft
wParam	16 Bits zusätzlicher, botschaftsabhängiger Informationen.
lParam	32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert

Der Rückgabewert enthält das Ergebnis der bearbeiteten Botschaft.

DefHookProc (Windows API Funktion)

Deklaration

```
function DefHookProc(Code: Integer; wParam: Word; lParam: Longint;
NextHook: TFarProc): Longint;
```

Beschreibung

Diese Funktion ruft die folgende Funktion in einer Reihe von Filterverwaltungsfunktionen. Eine Filterverwaltungsfunktion bearbeitet Ereignisse, bevor sie an die botschaftsbearbeitende Schleife einer Anwendung in der WinMain-Funktion übergeben werden.

Parameter

Code	Bestimmt den Code der Windows- Filterverwaltungsfunktion (auch Botschaftsfilterfunktion).
wParam	Bestimmt den Word-Parameter der Botschaft, die von der Filterverwaltungsfunktion bearbeitet wird.
lParam	Bestimmt den long-Parameter der Botschaft, die von der Filterverwaltungsfunktion bearbeitet wird.
NextHook	Zeigt auf einen Speicherbereich, der die Struktur **TFarProc** enthält.

Rückgabewert

Der Rückgabewert bestimmt einen Wert, der im direkten Verhältnis zum Parameter code steht.

Siehe auch:

SetWindowsHook
UnhookWindowsHook

DefineHandleTable (Windows API Funktion)

Deklaration

```
function DefineHandleTable(Offset: Word): Bool;
```

Beschreibung

Erzeugt eine private Handle-Tabelle im Standarddatensegment.

In dieser Tabelle werden die von der Funktion **GlobalLock** zurückgegebenen Adressen globaler Speicherobjekte gespeichert. Im Real Modus aktualisiert Windows die zugehörige Adresse in der individuellen Handle-Tabelle, wenn das globale Speicherobjekt verschoben wird. Im Protected Mode (Standardmodus oder erweiterter 386-Modus) aktualisiert Windows die Adressen nicht.

Die Handle-Tabelle besteht aus zwei Werten vom Typ Word, denen ein Array mit Adressen folgt.

- Der erste Wert enthält die Anzahl von Tabelleneinträgen, die vor dem Aufruf von DefineHandleTable initialisiert werden müssen.
- Der zweite Wert enthält die Anzahl von Einträgen, die auf 0 gesetzt werden müssen, wenn Windows die Liste der zuletzt benutzten Speicherbereiche aktualisiert. Jeder dieser Werte kann jederzeit verändert werden.

Die Adressen in der Tabelle werden von GlobalLock zurückgegeben.

Parameter

Offset	Das Offset der Tabelle ab Beginn des Datensegments. Der Wert 0 zeigt an, daß Windows die Tabelle nicht mehr aktualisieren muß.

Rückgabewert

Bei erfolgreicher Ausführung ungleich 0, sonst 0.

DefMDIChildProc (Windows API Funktion)

Deklaration

```
function DefMDIChildProc(Wnd: HWnd; Msg, wParam: Word; lParam: Longint):
Longint;
```

Beschreibung

Diese Funktion unterstützt die normale Bearbeitung von Windows-Botschaften, die von der Fensterfunktion eines untergeordneten MDI-Fensters nicht bearbeitet werden. .

Parameter

Wnd	Bezeichnet das untergeordnete MDI-Fenster.
Msg	Bestimmt die Nummer der Botschaft.
wParam	Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.
lParam	Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert

Der Rückgabewert hängt von der übergebenen Botschaft ab. Er gibt das Ergebnis der bearbeiteten Botschaft an.

DefWindowProc (Windows API Funktion)

Deklaration

```
function DefWindowProc(Wnd: HWnd; Msg, wParam: Word; lParam: Longint):
Longint;
```

Beschreibung

Diese Funktion unterstützt die normale Bearbeitung von allenWindows-Botschaften, die von der gegebenen Anwendung nicht bearbeitet werden.

Parameter

Wnd	Bezeichnet das Fenster, das die Botschaft weitergibt.
Msg	Bestimmt die Nummer der Botschaft.
wParam	Bestimmt 16 Bits zusätzlicher, botschaftsabhängiger Informationen.
lParam	Bestimmt 32 Bits zusätzlicher, botschaftsabhängiger Informationen.

Rückgabewert

Der Rückgabewert hängt von der übergebenen Botschaft ab. Er zeigt das Ergebnis der bearbeiteten Botschaft an.

DeleteAtom (Windows API Funktion)

Deklaration

```
function DeleteAtom(AnAtom: Atom): Atom;
```

Beschreibung

Diese Funktion löscht ein Atom und - wenn der Referenzzähler des Atoms 0 ist - den zugehörigen String aus der Atom-Tabelle.

Parameter

AnAtom Bezeichnet das zu löschende Atom und die zugehörige Zeichenkette.

Rückgabewert

Nach erfolgreicher Ausführung ist er 0. Schlägt die Funktion fehl, ist der Rückgabewert gleich Atom. In diesem Fall wird das Atom nicht gelöscht.

DeleteDC (Windows API Funktion)

Deklaration

```
function DeleteDC(DC: HDC): Bool;
```

Beschreibung

Diese Funktion löscht den angegebenen Gerätekontext. Entspricht der Parameter DC dem letzten Gerätekontext des gegebenen Geräts, wird dies demGerät mitgeteilt. Alle Speicher- und Systemressourcen desGeräts werden freigegeben.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Er ist ungleich 0, wenn derGerätekontext erfolgreich gelöscht wurde, unabhängig davon, ob es sich um den letzten Kontext desGeräts handelt. Tritt ein Fehler auf, ist der Rückgabewert 0.

DeleteMenu (Windows API Funktion)

Deklaration

```
function DeleteMenu(Menu: HMenu; Position, Flags: Word): Bool;
```

Beschreibung

Diese Funktion löscht einen Eintrag aus dem durch Menu bezeichneten Menü.Gehört ein Pop-up-Menü zu diesem Eintrag, wird auch das zugehörige Handle zerstört und der verwendete Speicher wird freigegeben.

Parameter

Menu	Bezeichnet das zu ändernde Menü.
Position	Gibt den zu löschenden Menüeintrag oder die ID des Befehls an.
Flags	Legt fest, wie der Parameter Position interpretiert wird. Enthält eine der **mf_xxx Flags**-Konstanten: mf_ByPosition mf_ByCommand

Rückgabewert

Er ist nach erfolgreicher Ausführung ungleich 0, andernfalls 0.

DeleteMetaFile (Windows API Funktion)

Deklaration

```
function DeleteMetaFile(MF: THandle): Bool;
```

Beschreibung

Diese Funktion verhindert den Zugriff auf eine Metadatei durch Freigabe der zugehörigen Systemressourcen. Die Metadatei selbst wird nicht zerstört.

Parameter

MF Bezeichnet die zu löschende Metadatei.

Rückgabewert

Er ist ungleich 0, wenn die Systemressourcen freigegeben sind, gleich 0, wenn der Parameter MF kein gültiges Handle ist.

Siehe auch:

GetMetaFile

DeleteObject (Windows API Funktion)

Deklaration

```
function DeleteObject(Handle: THandle): Bool;
```

Beschreibung

Diese Funktion löscht das durch Handle bezeichnte Objekt (ein virtueller Stift, Pinsel, Bitmap, eine Region, Schrift oder Farbpalette) durch Freigabe des zugehörigen Systemspeichers.

Parameter

Handle	Bezeichnet das Handle eines virtuellen Stifts, Pinsels, Bitmaps, einer Region, Schrift oder Farbpalette.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Objekt gelöscht wird. Er ist 0, wenn der Parameter Handle kein gültiges Handle ist oder aktuell in einemGerätekontext benutzt wird.

DestroyCaret (Windows API Prozedur)

Deklaration

```
procedure DestroyCaret;
```

Beschreibung

Diese Funktion zerstört die Form des Carets, löst es vom aktuell verwaltenden Fenster und löscht es vom Bildschirm, wenn es sichtbar ist.

DestroyCursor (Windows API Funktion)

Deklaration

```
function DestroyCursor(Cursor: HCursor): Bool;
```

Beschreibung

Diese Funktion zerstört den durch Cursor bezeichneten Zeiger und gibt den belegten Speicherplatz frei.

Parameter

Cursor Bezeichnet den zu zerstörenden Cursor.

Rückgabewert

Nach erfolgreicher Ausführung ist der Rückgabewert ungleich 0, andernfalls ist er gleich 0.

Siehe auch:

CreateCursor

DestroyIcon (Windows API Funktion)

Deklaration

```
function DestroyIcon(Icon: HIcon): Bool;
```

Beschreibung

Diese Funktion zerstört das durch Icon bezeichnete Symbol und gibt den belegten Speicherplatz frei.

Parameter

Icon	Bezeichnet das zu zerstörende Symbol.

Rückgabewert

Nach erfolgreicher Ausführung ist der Rückgabewert ungleich 0, andernfalls ist er gleich 0.

Siehe auch:

CreateIcon

DestroyMenu (Windows API Funktion)

Deklaration

```
function DestroyMenu(Menu: HMenu): Bool;
```

Beschreibung

Diese Funktion zerstört das durch Menu bezeichnete Menü und gibt den zugehörigen Speicher frei.

Parameter

Menu Bezeichnet das zu zerstörende Menü.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Menü zerstört wurde, andernfalls gleich 0.

DestroyWindow (Windows API Funktion)

Deklaration

```
function DestroyWindow(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion zerstört das angegebene Fenster. Ist das gegebene Fenster anderen Fenstern übergeordnet, werden dessen untergeordneten Fenster ebenfalls zerstört. DestroyWindow zerstört auch nicht-modale Dialogfenster.

Parameter

Wnd	Bezeichnet das zu zerstörende Fenster.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Fenster zerstört wurde, sonst ist er 0.

Siehe auch:

CreateDialog
wm_Destroy
wm_NCDestroy

DialogBox (Windows API Funktion)

Deklaration

```
function DialogBox(Instance: THandle; TemplateName: PChar; WndParent:
HWnd; DialogFunc: TFarProc): Integer;
```

Beschreibung

Diese Funktion erzeugt ein modales Dialogfenster vom Typ, in derGröße und mit den Steuerelementen, die in der durch TemplateName gegebenen Dialogfenstervorlage festgelegt sind, und sendet eine **wm_InitDialog**-Botschaft, bevor das Dialogfenster angezeigt wird.

Parameter

Instance	Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.
TemplateName	Zeigt auf eine null-termininierte Zeichenkette, die die Dialogfenstervorlage benennt.
WndParent	Bezeichnet das Fenster, von dem das Dialogfenster verwaltet wird.
DialogFunc	Adresse der Prozedurinstanz der Dialogfunktion.

Rückgabewert

Der Rückgabewert enthält den Wert des Parameters nResult der Funktion EndDialog. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

DialogBoxIndirect (Windows API Funktion)

Deklaration

```
function DialogBoxIndirect(Instance, DialogTemplate: THandle; WndParent:
HWnd; DialogFunc: TFarProc): Integer;
```

Beschreibung

Diese Funktion erzeugt ein modales Dialogfenster vom Typ, in derGröße und mit den Steuerelementen, die in der **Dialogfenstervorlage** festgelegt sind, und sendet eine **wm_InitDialog**-Botschaft, bevor das Dialogfenster angezeigt wird.

Parameter

Instance	Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.
DialogTemplate	Bezeichnet einen Block im globalen Speicher, der eine Dialogfenstervorlagen-Datenstruktur enthält.
WndParent	Bezeichnet das Fenster, zu dem das Dialogfenster gehört.
DialogFunc	Adresse der Prozedurinstanz der Dialogfunktion.

Rückgabewert

Der Rückgabewert enthält den Wert des Parameters nResult der Funktion EndDialog. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

DialogBoxIndirectParam (Windows API Funktion)

Deklaration

```
function DialogBoxIndirectParam(Instance, DialogTemplate: THandle;
WndParent: HWnd; DialogFunc: TFarProc; InitParam: Longint): Integer;
```

Beschreibung

Diese Funktion erzeugt ein durch die **Dialogfenstervorlage** definiertes modales Dialogfenster, übergibt vor dessen Darstellung die Botschaft **wm_InitDialog** an die Dialogfunktion. Die Funktion ermöglicht auch die Übergabe eines Initialisierungsparameters an die Callback-Funktion.

Parameter

Instance	Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.
DialogTemplate	Bezeichnet einen Block im globalen Speicher, der eine DLGTEMPLATE-Datenstruktur enthält.
WndParent	Bezeichnet das Fenster, zu dem das Dialogfenster gehört.
DialogFunc	st die Adresse der Prozedurinstanz der Dialogfunktion.
InitParam	Ein 32-Bit-Wert, der von lParam der Botschaft **wm_InitDialog** an die Dialogfunktion übergeben wird, wenn diese das Dialogfenster erstellt.

Rückgabewert

Enthält den Wert des Parameters nResult der Funktion EndDialog. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

DialogBoxParam (Windows API Funktion)

Deklaration

```
function DialogBoxParam(Instance: THandle; TemplateName: PChar; Parent:
HWnd; DialogFunc: TFarProc; InitParam: Longint): Integer;
```

Beschreibung

Diese Funktion erzeugt ein gemäß TemplateName definiertes modales Dialogfenster, übergibt vor dessen Darstellung die Botschaft **wm_InitDialog** an die Dialogfunktion und InitParam als Parameter lParam dieser Botschaft. Die Botschaft erlaubt es der Dialogfunktion, die Steuerelemente des Dialogfensters zu initialisieren.

Parameter

Instance	Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Dialogfenstervorlage enthält.
TemplateName	Zeigt auf einen null-terminierten String, die die Dialogfenstervorlage benennt.
WndParent	Bezeichnet das Fenster, zu dem das Dialogfenster gehört.
DialogFunc	Ist die Adresse der Prozedurinstanz der Dialogfunktion.
InitParam	Ein 32-Bit-Wert, der in lParam der Botschaft **wm_InitDialog** an die Dialogfunktion übergeben wird, wenn diese das Dialogfenster erstellt.

Rückgabewert

Der Rückgabewert enthält den Wert des Parameters nResult der Funktion EndDialog. Der Rückgabewert ist -1, wenn die Funktion das Dialogfenster nicht erstellen konnte.

DispatchMessage (Windows API Funktion)

Deklaration

```
function DispatchMessage(var Msg: TMsg): Longint;
```

Beschreibung

Diese Funktion übergibt die Botschaft in der durch Msg bezeichneten MSG-Struktur an die Fensterfunktion des vorgegebenen Fensters.

Parameter

Msg	Zeigt auf eine **TMsg**-Datenstruktur.

Rückgabewert

Der Rückgabewert enthält den von der Fensterfunktion zurückgegebenen Wert. Er hängt von der übergebenen Botschaft ab, wird jedoch meist ignoriert.

DlgDirList (Windows API Funktion)

Deklaration

```
function DlgDirList(Dlg: HWnd; PathSpec: PChar; IDListBox, IDStaticPath:
Integer; Filetype: Word): Integer;
```

Beschreibung

Diese Funktion füllt ein Listenfenster mit einer Datei- oder Verzeichnisliste. Das Listenfenster wird durch den Parameter IDListBox bezeichnet und mit den Namen der Dateien gefüllt, die dem in PathSpec vorgegebenen Pfadnamen entsprechen.

Parameter

Dlg	Bezeichnet das Dialogfenster, welches das Listenfenster enthält.
PathSpec	Zeigt auf einen null-terminierten String mit dem Pfadnamen.
IDListBox	Bezeichner einer Liste.
IDStaticPath	Bestimmt den Bezeichner des statischen Textfeldes, der für die Darstellung des aktuellen Laufwerks und Verzeichnisses verwendet wird.
Filetype	Bestimmt die darzustellenden DOS-Dateiattribute: $0000 (Lesen\\Schreiben), $0001 (Nur Lesen), $0002 (Verborgen), $0004 (System), $0010 (Unterverzeichnis), $0020 Archive), $2000 (**lb_Dir**), $4000(Laufwerke), $8000 (exklusive)

Rückgabewert

Er ist ungleich 0, wenn eine Liste dargestellt wurde, auch wenn es sich dabei um eine leere Liste handelt. Der Rückgabewert 0 zeigt an, daß der Eingabestring keinen gültigen Suchpfad enthielt.

Siehe auch:

lb_ResetContent
lb_Dir

DlgDirListComboBox (Windows API Funktion)

Deklaration

```
function DlgDirListComboBox(Dlg: HWnd; PathSpec: PChar; IDComboBox,
IDStaticPath: Integer; Filetype: Word): Integer;
```

Beschreibung

Diese Funktion füllt das Listenfenster desdurch den Parameter IDComboBox bezeichneten Kombinationsfensters mit einer Datei- oder Verzeichnisliste, die der Dateinamensvorgabe PathSpec entsprechen.

Parameter

Dlg	Bezeichnet das Dialogfenster, welches das Kombinationsfenster enthält.
PathSpec	Zeigt auf einen null-terminierten String mit dem Verzeichnisnamen.
IDComboBox	Bezeichner eines Kombinationsfensters.
IDStaticPath	Bestimmt den Bezeichner des statischen Textfeldes, das für die Darstellung des aktuellen Laufwerks und Verzeichnisses verwendet wird.
Filetype	Bestimmt die darzustellenden DOS-Dateiattribute: $0000 (Lesen\\Schreiben), $0001 (Nur Lesen), $0002 (Verborgen), $0004 (System), $0010 (Unterverzeichnis), $0020 Archive), $2000 (**lb_Dir**), $4000(Laufwerke), $8000 (exklusive)

Rückgabewert

Er ist ungleich 0, wenn eine Liste dargestellt wurde, auch wenn es sich dabei um eine leere Liste handelt. Der Rückgabewert 0 zeigt an, daß der Eingabestring keinen gültigen Suchpfad enthielt.

Siehe auch:

cb_ResetContent
cb_Dir

DlgDirSelect (Windows API Funktion)

Deklaration

```
function DlgDirSelect(Dlg: HWnd; Str: PChar; IDListBox: Integer): Bool;
```

Beschreibung

Diese Funktion ermittelt die aktuelle Auswahl in einem Listenfenster und kopiert die Auswahl in den durch Str vorgegebenen Puffer.

Parameter

Dlg	Bezeichnet das Dialogfenster, welches das Listenfenster enthält.
Str	Zeigt auf einen Puffer, der den ausgewählten Pfadnamen empfangen soll.
IDListBox	Bezeichner der Listen-ID in einem Dialogfenster.

Rückgabewert

Der Rückgabewert gibt den Status der aktuellen Auswahl im Listenfenster an. Handelt es sich dabei um einen Verzeichnisnamen, ist der Rückgabewert ungleich, andernfalls gleich 0.

Siehe auch:

DlgDirList
lb_GetCurSel
lb_GetText

DlgDirSelectComboBox (Windows API Funktion)

Deklaration

```
function DlgDirSelectComboBox(Dlg: HWnd; Str: PChar; IDComboBox: Integer):
Bool;
```

Beschreibung

Diese Funktion ermittelt die aktuelle Auswahl im Listenfenster eines einfaches Kombinationsfensters (**cbs_Simple**) und kopiert die Auswahl in den durch Str vorgegebenen Puffer.

Parameter

Dlg	Bezeichnet das Dialogfenster, welches das Kombinationsfenster enthält.
Str	Zeigt auf einen Puffer, der den ausgewählten Pfadnamen empfangen soll.
IDListBox	Bezeichnet die Integer-ID des Kombinationsfensters in einem Dialogfenster.

Rückgabewert

Der Rückgabewert gibt den Status der aktuellen Auswahl im Kombinationsfenster an. Handelt es sich dabei um einen Verzeichnisnamen, ist der Rückgabewert ungleich, andernfalls gleich 0.

Siehe auch:

DlgDirListComboBox
cb_GetCurSel
cb_GetLBTText

DOS3Call (Windows API Pozdur)

Deklaration

```
procedure DOS3Call;
```

Beschreibung

Ruft die DOS Interrupt-Funktion 21h auf.

DOS3Call sollte nur von Assembler-Routinen aufgerufen werden, da die Register für den INT 21h Aufruf vorbereitet werden müssen.

Unter Windows ist die Verwendung von DOS3Call etwas schneller als ein direkter Aufruf des Software-Interrupts.

DPtoLP (Windows API Funktion)

Deklaration

```
function DPtoLP(DC: HDC; var Points; Count: Integer): Bool;
```

Beschreibung

Diese Funktion konvertiert die Punkte einesGeräts in virtuelle Punkte.

Parameter

DC	Bezeichnet den Gerätekontext.
Points	Zeigt auf ein Array von Punkten. Jeder Punkt muß eine **TPoint**-Datenstruktur sein.
Count	Bestimmt die Anzahl der Punkte im Array.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn alle Punkte konvertiert wurden. Andernfalls ist der Rückgabewert 0.

DrawFocusRect (Windows API Prozedur)

Deklaration

```
procedure DrawFocusRect(DC: HDC; var Rect: TRect);
```

Beschreibung

Diese Funktion führt eine **XOR**-Operation aus und zeichnet ein Rechteck vom Typ, der den Fokus anzeigt.

Parameter

DC	Bezeichnet den Gerätekontext.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die die Koordinaten des zu zeichnenden Rechtecks festlegt.

DrawIcon (Windows API Funktion)

Deklaration

```
function DrawIcon(DC: HDC; X, Y: Integer; Icon: HIcon): Bool;
```

Beschreibung

Diese Funktion zeichnet ein Symbol im angegebenen Gerät.

Parameter

DC Bezeichnet den Gerätekontext.

X, YBestimmt die virtuellen Koordinaten der oberen linken Ecke des Symbols.

Icon Bezeichnet das zu zeichnende Symbol.

Rückgabewert

Nach erfolgreicher Ausführung der Funktion ist der Rückgabewert ungleich 0, andernfalls gleich 0.

DrawMenuBar (Windows API Prozedur)

Deklaration

```
procedure DrawMenuBar(Wnd: HWnd);
```

Beschreibung

Diese Funktion stellt eine Windows-Menüleiste wieder her.

Parameter

Wnd	Bezeichnet das Fenster, dessen Menü wiederhergestellt werden muß.

DrawText (Windows API Funktion)

Deklaration

```
function DrawText(DC: HDC; Str: PChar; Count: Integer; var Rect: TRect;
Format: Word): Integer;
```

Beschreibung

Diese Funktion zeichnet formatierten Text. Die Art der Formatierung wird durch den Parameter Format festgelegt. Solange das dt_NoClip-Format nicht verwendet wird, kürzt DrawText den Text so, daß er nicht außerhalb des begrenzenden Rechtecks erscheint.

Parameter

DC	Bezeichnet den Gerätekontext.
Str	Zeigt auf den darzustellenden String. Hat der Parameter Count den Wert -1, muß der String null-terminiert sein.
Count	Anzahl der Bytes im String.
Rect	Zeigt auf eine **TRect** -Datenstruktur, die das Rechteck in virtuellen Koordinaten enthält, in dem der Text formatiert werden soll.
Format	Bestimmt die Art der Textformatierung. Möglich ist jede Kombination der **dt_xxx Flags**

Rückgabewert

Höhe des Textes.

Ellipse (Windows API Funktion)

Deklaration

```
function Ellipse(DC: HDC; X1, Y1, X2, Y2: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet eine Ellipse. Der Mittelpunkt der Ellipse wird durch den Mittelpunkt des eingrenzenden Rechtecks festgelegt. Der Rahmen der Ellipse wird mit dem aktuellen Stift gezeichnet, die Innenfläche mit dem aktuellen Pinsel ausgefüllt.

Parameter

DC	Bezeichnet den Gerätekontext.
X1, Y1	Legt die virtuellen Koordinaten der oberen linken Ecke des eingrenzenden Rechtecks fest.
X2, Y2	Legt die virtuellen Koordinaten der unteren rechten Ecke des eingrenzenden Rechtecks fest.

Rückgabewert

Am Rückgabewert ist abzulesen, ob die Ellipse gezeichnet wird. Wird sie nicht gezeichnet, ist der Rückgabewert 0. Andernfalls ist er ungleich 0.

EmptyClipboard (Windows API Funktion)

Deklaration

```
function EmptyClipboard: Bool;
```

Beschreibung

Diese Funktion leert die Zwischenablage und gibt die Handles auf die entsprechenden Daten frei. Außerdem wird der Besitz der Zwischenablage dem aufrufenden Fenster zugewiesen.

Rückgabewert

Der Rückgabewert meldet den Status der Zwischenablage. Ist er ungleich 0, ist die Zwischenablage leer, andernfalls liegt ein Fehler vor.

EnableHardwareInput (Windows API Funktion)

Deklaration

```
function EnableHardwareInput(EnableInput: Bool): Bool;
```

Beschreibung

Diese Funktion schaltet Maus- und Tastatureingaben aus. Die Eingaben werden gespeichert, wenn der Parameter EnableInput den Wert True hat. Ist der Wert False, werden die Eingaben verworfen.

Parameter

EnableInput — Erlaubt die Speicherung der Eingaben, wenn der Parameter einen Wert ungleich 0 annimmt. Andernfalls werden die Eingaben verworfen.

Rückgabewert

Der Rückgabewert zeigt an, ob die Eingabe über Maus und Tastatur abgeschaltet ist. Er ist ungleich 0 (Standard), wenn die Eingabe vorher eingeschaltet war. Andernfalls ist er gleich 0.

EnableMenuItem (Windows API Funktion)

Deklaration

```
function EnableMenuItem(Menu: HMenu; IDEnableItem, Enable: Word): Bool;
```

Beschreibung

Diese Funktion aktiviert, deaktiviert oder versetzt eine Menüoption in nicht aktivierbaren Zustand.

Parameter

Menu	Bezeichnet das Menü.
IDEnableItem	ID oder Position eines Menüeintrags oder eines Pop-Up-Menüs.
Enable	Legt die zugehörige Aktion fest. Dies kann eine Kombination der **mf_xxx Flag** Konstanten mf_Command oder mf_ByPosition mit mf_Disabled, mf_Enabled, or mf_Grayed sein. Diese Werte können mit Hilfe des bitorientierten OR-Operators kombiniert werden.

Rückgabewert

Der Rückgabewert definiert den vorhergehenden Menüstatus. Der Rückgabewert ist -1, wenn der Menüpunkt nicht existiert.

EnableWindow (Windows API Funktion)

Deklaration

```
function EnableWindow(Wnd: HWnd; Enable: Bool): Bool;
```

Beschreibung

Diese Funktion aktiviert bzw. deaktiviert die Eingabe über Maus und Tastatur in das festgelegte Fenster oder Steuerelement.

Parameter

Wnd	Bezeichnet das betroffene Fenster.
Enable	Legt fest, ob das angegebene Fenster aktiviert oder deaktiviert werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Fenster korrekt aktiviert bzw. deaktiviert wurde. Ergibt sich ein Fehler, ist der Wert 0.

EndDialog (Windows API Prozedur)

Deklaration

```
procedure EndDialog(Dlg: HWnd; Result: Integer);
```

Beschreibung

Die Funktion beendet ein modales Dialogfenster und übergibt den mit Result gegebenen Wert an die Funktion **DialogBox**, die das Dialogfenster erzeugt hat.

Parameter

Dlg	Bezeichnet das zu schließende Dialogfenster.
Result	Enthält den Wert, der vom Dialogfenster an die aufrufende Funktion DialogBox zurückgegeben wird.

EndPaint (Windows API Prozedur)

Deklaration

```
procedure EndPaint(Wnd: HWnd; var Paint: TPaintStruct);
```

Beschreibung

Die Funktion legt das Ende des Zeichnens im durch Wnd bezeichneten Fenster fest.

Parameter

Wnd	Legt das neu zu zeichnende Fenster fest.
Paint	Weist auf eine **TPaintStruct**-Datenstruktur, in der die durch **BeginPaint** ermittelten Zeichnungsinformationen abgelegt sind.

EnumChildWindows (Windows API Funktion)

Deklaration

```
function EnumChildWindows(WndParent: HWnd; EnumFunc: TFarProc; lParam:
Longint): Bool;
```

Beschreibung

Die Funktion listet die untergeordneten Fenster des festgelegten übergeordneten Fensters auf, indem das Handle jedes untergeordneten Fensters und lParam an die anwendungsunterstützte Callback-Funktion übergeben wird. Die Funktion EnumChildWindows zählt die untergeordneten Fenster, bis die aufgerufene Funktion den Wert 0 zurückgibt oder das letzte untergeordnete Fenster aufgelistet wurde.

Parameter

WndParent	Legt das übergeordnete Fenster fest, dessen untergeordnetes Fenster aufgelistet werden sollen.
EnumFunc	Ist die Adresse der Callback-Funktion.
lParam	Legt den Wert fest, der an die Callback-Funktion zur Verwendung in der Anwendung übergeben wird.

Rückgabewert

Wurden alle untergeordneten Fenster aufgelistet, ist der Rückgabewert ungleich 0; in allen anderen Fällen ist er 0.

EnumClipboardFormats (Windows API Funktion)

Deklaration

```
function EnumClipboardFormats(Format: Word): Word;
```

Beschreibung

Die Funktion listet die zur Zwischenablage gehörigen verfügbaren Formate auf.

Parameter

Format — Gibt ein bekanntes Format an oder hat den Wert 0 für das erste Format in der Liste, die Formate werden durch **cf_xxx** bezeichnet.

Rückgabewert

Der Rückgabewert kennzeichnet das nächste bekannte Datenformat der Zwischenablage. Er ist 0, wenn Format den letzten Eintrag in der Liste bezeichnet oder die Zwischenablage nicht geöffnet ist.

Siehe auch:

OpenClipboard

EnumFonts (Windows API Funktion)

Deklaration

```
function EnumFonts(DC: HDC; FaceName: PChar; FontFunc: TFarProc; Data:
Pointer): Integer;
```

Beschreibung

Die Funktion listet die auf einem bestimmten Gerät zur Verfügung stehenden Schriften auf. Für jede Schrift, deren Schriftartname durch den Parameter FaceName festgelegt wurde, findet die Funktion die zugehörigen **TLogFont**, **TTextMetric**, FontType und Data. Informationen und übergibt sie an die Funktion, auf die der Parameter FontFunc verweist. Die Auflistung wird fortgesetzt, bis keine Schriften mehr gefunden werden oder die Callback-Funktion 0 zurückgibt.

Parameter

DC	Bezeichnet den Gerätekontext.
FaceName	Zeigt auf die null-terminierte Zeichenkette, die den Schriftartnamen der gewünschten Schriften festlegt.
FontFunc	FARPROC ist die Adresse der Prozedurinstanz der Callback-Funktion.
Data	Zeigt auf die von der Anwendung zur Verfügung gestellten Daten. Die Daten werden zusammen mit den Schriftinformationen an die Callback-Funktion übergeben.

Rückgabewert

Der Rückgabewert ist der letzte Rückgabewert der Callback-Funktion. Seine Bedeutung ist anwenderdefiniert.

EnumMetaFile (Windows API Funktion)

Deklaration

```
function EnumMetaFile(DC: HDC; MF: THandle; CallbackFunc: TFarProc;
ClientData: LPByte): Bool;
```

Beschreibung

Diese Funktion listet die GDI-Aufrufe in einer Metadatei auf, indem sie folgende Daten an eine Callback-Funktion übergibt:

- DC
- einen Zeiger auf die Objekt-Handle-Tabelle der Metadatei
- einen Zeiger auf einen Record in der Metadatei
- die Anzahl von Objekten, die den in der Tabelle verzeichneten Handles zugeordnet sind und
- ClientData.

Die Auflistung wird fortgesetzt, bis keine GDI-Aufrufe mehr vorhanden sind oder die Callback-Funktion den Wert 0 zurückgibt.

Parameter

DC	Bezeichnet das der Metadatei zugeordnete Gerät.
MF	Bezeichnet die Metadatei.
CallbackFunc	Ist die Adresse der Prozedurinstanz der Callback-Funktion.
ClientData	Zeigt auf die Daten, die der Callback-Funktion übergeben werden.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Callback-Funktion alle GDI-Aufrufe in einer Metadatei aufgelistet hat, andernfalls ist er 0.

EnumObjects (Windows API Funktion)Deklaration

Deklaration

```
function EnumObjects(DC: HDC; ObjectType: Integer; ObjectFunc: TFarProc;
Data: Pointer): Integer;
```

Beschreibung

Diese Funktion listet die verfügbaren Objekttypen einesGeräts auf, indem **TLogPen** oder **TLogBrush** und Data an die Callback-Funktion übergeben werden. Die Callback-Funktion wird solange aufgerufen, bis keine weiteren Objekte auffindbar sind oder sie den Rückgabewert 0 hat.

Parameter

DC	Bezeichnet den Gerätekontext.
ObjectType	Bezeichnet den Objekttyp.Kann den Wert einer der **obj_xxx Konstanten** haben: obj_Brush oder obj_Pen
ObjectFunc	Adresse der Prozedurinstanz der Callback-Funktion.
Data	Zeiger auf die Daten, die der Callback-Funktion übergeben werden.

Rückgabewert

Der Rückgabewert ist der letzte Rückgabewert der Callback-Funktion. Dessen Bedeutung wird vom Benutzer definiert.

EnumProps (Windows API Funktion)

Deklaration

```
function EnumProps(Wnd: HWnd; EnumFunc: TFarProc): Integer;
```

Beschreibung

Diese Funktion listet alle Einträge der Eigenschaftsliste des betroffenen Fensters auf. Die Auflistung erfolgt durch Übergabe Wnd, nDummy, PSTR und hData an die Callback-Funktion. EnumProps läuft, bis alle Einträge aufgelistet wurden oder die Callback-Funktion 0 zurückgibt.

Parameter

Wnd	Legt das Fenster fest, dessen Eigenschaftsliste aufgelistet werden soll.
EnumFunc	Adresse der Prozedurinstanz der Callback-Funktion.

Rückgabewert

Der Rückgabewert ist der letzte Rückgabewert der Callback-Funktion. Er ist -1, wenn die Funktion keine Liste mit Eigenschaften findet.

EnumTaskWindows (Windows API Funktion)

Deklaration

```
function EnumTaskWindows(Task: THandle; EnumFunc: TFarProc; lParam:
Longint): Bool;
```

Beschreibung

Diese Funktion listet aller Fenster einer Task auf, indem das Fenster-Handle und lParam an die Callback-Funktion übergeben werden. Die Auflistung wird beendet, wenn die Callback-Funktion den Wert 0 zurückgibt.

Parameter

Task	Bezeichnet die Task.
EnumFunc	Ist die Adresse der Prozedurinstanz der Callback-Funktion des Fensters.
lParam	Legt den 32-Bit-Wert fest, der zusätzliche Parameter an die Callback-Funktion übergibt.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn alle zu einer Task gehörenden Fenster aufgelistet wurden. In jedem anderen Fall ist der Rückgabewert 0.

Siehe auch:

GetCurrentTask

EnumWindows (Windows API Funktion)

Deklaration

```
function EnumWindows(EnumFunc: TFarProc; lParam: Longint): Bool;
```

Beschreibung

Diese Funktion listet alle übergeordneten Fenster auf dem Bildschirm auf durch Übergabe des zugehörigen Handles auf jedes Fenster und lParam an die Callback-Funktion. Die Funktion fährt mit der Auflistung fort, bis die aufgerufene Funktion den Wert 0 zurückgibt oder das letzte Fenster gelistet wurde.

Parameter

EnumFunc Ist die Adresse der Prozedurinstanz der Callback-Funktion.
lParam Legt den Übergabewert an die Callback-Funktion für die Anwendung fest.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn alle Fenster gelistet wurden. Andernfalls ist der Wert 0.

EqualRect (Windows API Funktion)

Deklaration

```
function EqualRect(var Rect1, Rect2: TRect): Bool;
```

Beschreibung

Diese Funktion überprüft die Übereinstimmung zweier Rechtecke durch Vergleich der Koordinaten der oberen linken und der unteren rechten Ecken.

Parameter

Rect1, Rect2	Geben die Koordinaten der oberen linken und der unteren rechten Ecke der zu vergleichenden Rechtecke enthält.

Rückgabewert

Der Rückgabewert zeigt an, ob die Rechtecke identisch sind. In diesem Fall ist der Rückgabewert ungleich 0, andernfalls ist er gleich 0.

EqualRgn (Windows API Funktion)

Deklaration

```
function EqualRgn(SrcRgn1, SrcRgn2: HRgn): Bool;
```

Beschreibung

Diese Funktion überprüft, ob zwei vorgegebene Regionen übereinstimmen.

Parameter

SrcRgn1 Bezeichnet die erste Region.
SrcRgn2 Bezeichnet die zweite Region.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Regionen identisch sind. Andernfalls ist der Rückgabewert 0.

Escape (Windows API Funktion)

Deklaration

```
function Escape(DC: HDC; Escape, Count: Integer; InData, OutData:
Pointer): Integer;
```

Beschreibung

Diese Funktion erlaubt es der Anwendung, auf die Fähigkeiten einer bestimmten Geräteeinheit zuzugreifen, die über die grafische Geräteschnittstelle (GDI) nicht direkt verfügbar sind.

Parameter

DC	Legt den Gerätekontext fest
Count	Bestimmt die Anzahl der Datenbytes, auf die der Parameter InData zeigt.
InData	Zeigt auf die von dieser Escape-Funktion benötigte Struktur mit Eingabedaten.
OutData	Zeigt auf die Datenstruktur, die zum Empfang der Ausgabedaten dieser Escape-Funktion bestimmt ist, oder ist **nil**, wenn keine Daten zurückgegeben werden.

Rückgabewert

Der Rückgabewert ist positiv, wenn die Funktion erfolgreich ausgeführt wurde. Der Rückgabewert ist 0, wenn die Escape-Funktion nicht implementiert wurde. Ein negativer Wert zeigt einen Fehler an. Im Falle eines Fehlers, kann einer **sp_xxx Spoolerfehlercodes**ausgegeben werden.

EscapeCommFunction (Windows API Funktion)

Deklaration

```
function EscapeCommFunction(Cid, Func: Integer): Integer;
```

Beschreibung

Diese Funktion weist die serielle Schnittstelle an, die durch den Parameter Func bezeichnete erweiterte Funktion auszuführen.

Parameter

Cid	Legt die serielle Schnittstelle für die Ausführung der erweiterten Funktion fest.
Func	Legt den Funktionscode der erweiterten Funktion fest und hat den Wert einer der **Escape comm** -Konstanten.

Rückgabewert

Der Rückgabewert ist 0 nach erfolgreicher Ausführung. Wurde durch den Parameter Func ein ungültiger Funktionscode festgelegt, ist der Rückgabewert negativ.

Siehe auch:

OpenComm

ExcludeClipRect (Windows API Funktion)

Deklaration

```
function ExcludeClipRect(DC: HDC; X1, Y1, X2, Y2: Integer): Integer;
```

Beschreibung

Diese Funktion erzeugt eine neue Clipping-Region, die aus der existierenden Clipping-Region abzüglich des festgelegten Rechtecks besteht.

Parameter

DC	Bezeichnet den Gerätekontext.
X1, Y1	Legt die virtuellen Koordinaten der oberen linken Ecke des Rechtecks fest.
X2, Y2	Legt die virtuellen Koordinaten der unteren rechten Ecke des Rechtecks fest.

Rückgabewert

Der Rückgabewert gibt den Typ der neuen Clipping-Region an und hat den Wert einer der **Region-Flags**.

ExcludeUpdateRgn (Windows API Funktion)

Deklaration

```
function ExcludeUpdateRgn(DC: HDC; Wnd: HWnd): Integer;
```

Beschreibung

Diese Funktion verhindert das Zeichnen innerhalb ungültiger Bereiche eines Fensters durch den Ausschluß einer aktualisierten Region im Fenster aus der Clipping-Region.

Parameter

DC	Bezeichnet den zur Clipping-Region gehörenden Gerätekontext.
Wnd	Bezeichnet das zu aktualisierende Fenster.

Rückgabewert

Der Rückgabewert gibt den Typ der neuen Clipping-Region an und hat den Wert einer der **Region-Flags**.

ExtFloodFill (Windows API Funktion)

Deklaration

```
function ExtFloodFill(DC: HDC; X, Y: Integer; Color: TColorRef; FillType:
Word): Bool;
```

Beschreibung

Diese Funktion füllt einen Bereich der Bildschirmoberfläche mit dem aktuellen Pinsel in der durch FillType bezeichneten Weise.

Parameter

DC	Bezeichnet den Gerätekontext.
X, Y	Bestimmt die virtuellen Koordinaten des Ausgangspunktes.
Color	**TColorRef** der Eingrenzung bzw. zu füllenden Fläche.
FillType	Eine der **Füllmusterstilflags**

Rückgabewert

Bei erfolgreicher Ausführung der Funktion ist der Wert ungleich 0, andernfalls gleich 0.

Siehe auch:

FloodFill

ExtTextOut (Windows API Funktion)

Deklaration

```
function ExtTextOut(DC: HDC; X, Y: Integer; Options: Word; Rect: LPRect;
Str: PChar; Count: Word; Dx: LPInteger): Bool;
```

Beschreibung

Diese Funktion schreibt eine Zeichenkette in der aktuellen Schrift in einen durch Rect bezeichneten rechteckigen Bereich des Bildschirms.

Parameter

DC	Bezeichnet den Gerätekontext.
X, Y	Bestimmt die virtuellen Koordinaten für den Ursprung des ersten Zeichens der Zeichenkette.
Options	Bestimmt den Typ des Rechtecks. Der Parameter kann eine Kombination der Werte **eto_xxx** sein.
Rect	Zeigt auf eine Datenstruktur vom Typ TRect oder ist **nil**.
Str	Zeigt auf die angegebene Zeichenkette.
Count	Gibt die Anzahl der Zeichen in der Zeichenkette an.
Dx	Zeigt auf ein Array von Werten, welche den Abstand zwischen den Ursprüngen benachbarter Buchstaben festlegen.

Rückgabewert

Der Rückgabewert zeigt an, ob die Zeichenkette dargestellt wurde. In diesem Fall ist der Wert ungleich 0, andernfalls ist der Wert 0.

Siehe auch:

SetTextAlign

FatalExit (Windows API Prozedur)

Deklaration

```
procedure FatalExit(Code: Integer);
```

Beschreibung

Zeigt den Fehlercode Code an und eine symbolische Stack-Darstellung, die die Ausführung bis zu diesem Punkt andeutet. Der Bentuzer wird nach Anweisungen zu weiteren Bearbeitung abgefragt. Diese Prozedur sollte nur zum Testen verwendet werden.

Parameter

Code	Angezeigter Fehlercode

FillRect (Windows API Funktion)

Deklaration

```
function FillRect(DC: HDC; var Rect: TRect; Brush: HBrush): Integer;
```

Beschreibung

Diese Funktion füllt ein gegebenes Rechteck mit dem in Brush angegebenen Pinsel aus. Die Funktion füllt das gesamte Rechteck einschließlich des linken und oberen Randes aus, aber sie füllt nicht den rechten und unteren Rand.

Parameter

DC	Bezeichnet den Gerätekontext.
Rect	Zeigt auf eine **TRect-**Datenstruktur, die die virtuellen Koordinaten des auszufüllenden Rechtecks enthält.
Brush	Bezeichnet den Pinsel, der zum Füllen benutzt wird.

Rückgabewert

Obwohl der Rückgabewert der Funktion vom Typ Integer ist, wird der Rückgabewert nicht benutzt und hat keine Bedeutung.

Siehe auch:

CreateHatchBrush
CreatePatternBrush
CreateSolidBrush
GetStockObject

FillRgn (Windows API Funktion)

Deklaration

```
function FillRgn(DC: HDC; Rgn: HRgn; Brush: HBrush): Bool;
```

Beschreibung

Diese Funktion füllt eine Region mit dem Pinsel, der durch den Parameter Brush bezeichnet wird.

Parameter

DC	Bezeichnet den Gerätekontext.
Rgn	Bezeichnet die Region, die ausgefüllt werden soll.
Brush	Bezeichnet den Pinsel, der benutzt werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

FindAtom (Windows API Funktion)

Deklaration

```
function FindAtom(Str: PChar): Atom;
```

Beschreibung

Diese Funktion durchsucht die Atomtabelle nach der Zeichenkette, auf die der Parameter Str zeigt und ruft das Atom ab, das dem String zugeordnet ist.

Parameter

Str Zeigt auf die (null-terminierte) Zeichenkette, nach der gesucht werden soll.

Rückgabewert

Bezeichnet das Atom, das dem String zugeordnet ist. Er ist 0, wenn der String nicht in der Tabelle ist.

FindResource (Windows API Funktion)

Deklaration

```
function FindResource(Instance: THandle; Name, ResType: PChar): THandle;
```

Beschreibung

Diese Funktion bestimmt die Position einer Ressource in der angegebenen Ressourcendatei.

Parameter

Instance	Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.
Name	Zeigt auf einen null-terminierten String, die den Namen der Ressource darstellt.
ResType	Zeigt auf eine Integer-ID, einen null-terminierten String, die die Typbezeichnung der Ressource darstellt. Für vordefinierte Ressourcentypen nimmt der Parameter einen der Werte der **rt_xxx** Konstanten an.

Rückgabewert

Bezeichnet die benannte Ressource. Er ist 0, wenn die angeforderte Ressource nicht gefunden werden kann.

FindWindow (Windows API Funktion)

Deklaration

```
function FindWindow(ClassName, WindowName: PChar): HWnd;
```

Beschreibung

Diese Funktion gibt das Handle des übergeordneten Fensters zurück, dessen Klasse mit dem Parameter ClassName angegeben ist und dessen Fenstername oder Titelleiste durch WindowName angegeben wird.

Parameter

ClassName — Zeigt auf einen null-terminierten String, die die Klassenbezeichnung des Fensters angibt. Wenn ClassName den Wert **nil** hat, dann gelten alle Klassenbezeichnungen.

WindowName — Zeigt auf einen null-terminierten String, die den Fensternamen angibt, oder hat den Wert 0 für alle Fensternamen.

Rückgabewert

Bezeichnet das Fenster, das den angegebenen Klassennamen und den angegebenen Fensternamen hat. Er ist 0, wenn kein solches Fenster gefunden wurde.

FlashWindow (Windows API Funktion)

Deklaration

```
function FlashWindow(Wnd: HWnd; Invert: Bool): Bool;
```

Beschreibung

Diese Funktion läßt ein angegebenes Fenster oder Symbol einmal aufblinken. (Eine inaktive Titelleiste wird zu einer aktiven; eine aktive Titelleiste wird zu einer inaktiven.)

Parameter

Wnd	Bezeichnet das Fenster, das aufblinken soll.
Invert	Gibt an, ob das Fenster aufleuchten oder wieder in seinen ursprünglichen Zustand versetzt werden soll. Das Fenster wird von einem Zustand in den anderen versetzt, wenn der Parameter ungleich 0 ist. Wenn der Parameter 0 ist, dann wird das Fenster wieder in seinen ursprünglichen Zustand versetzt (entweder aktiv oder inaktiv).

Rückgabewert

Der Rückgabewert gibt den Status des Fensters vor dem Aufruf der Funktion FlashWindow an. Er ist ungleich 0, wenn das Fenster vor dem Aufruf aktiv war. Andernfalls ist er 0.

FloodFill (Windows API Funktion)

Deklaration

```
function FloodFill(DC: HDC; X, Y: Integer; Color: TColorRef): Bool;
```

Beschreibung

Diese Funktion füllt einen durch Color begrenzten Bereich auf der Bildschirmfläche mit dem aktuellen Pinsel aus.

Parameter

DC	Bezeichnet den Gerätekontext.
X, Y	Gibt die virtuellen Koordinaten des Punktes an, von dem aus das Ausfüllen beginnt.
Color	Gibt die Farbe derGrenze an und ist eine **TColorRef**-Datenstruktur.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist 0, wenn das Ausfüllen nicht zu Ende geführt werden konnte, der gegebene Punkt die von Color bezeichnete Farbe hat oder der Punkt außerhalb der Clipping-Region liegt.

FlushComm (Windows API Funktion)

Deklaration

```
function FlushComm(Cid, Queue: Integer): Integer;
```

Beschreibung

Diese Funktion leert die Sende- oder Empfangswarteschlange der Schnittstelle, die durch den Parameter Cid angegeben ist. Der Parameter Queue gibt an, welche Warteschlange geleert werden soll.

Parameter

Cid	Gibt die Schnittstelle an, deren Warteschlange geleert werden soll.
Queue	Gibt die Schlange an, die geleert werden soll. Wenn Queue 0 ist, dann wird die Sendeschlange geleert. Wenn er 1 ist, dann wird die Empfangsschlange geleert.

Rückgabewert

Der Rückgabewert ist 0, wenn sie erfolgreich ausgeführt wurde. Er ist negativ, wenn Cid ein ungültigesGerät bezeichnet oder wenn Queue keine gültige Schlange ist.

Siehe auch:

OpenComm

FrameRect (Windows API Prozedur)

Deklaration

```
procedure FrameRect(DC: HDC; var Rect: TRect; Brush: HBrush;
```

Beschreibung

Diese Funktion zeichnet eine Umrandung um das Rechteck. Die Breite und Höhe der Umrandung beträgt immer eine virtuelle Einheit.

Parameter

DC	Bezeichnet den Gerätekontext des Fensters.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die die virtuellen Koordinaten der oberen linken und der unteren rechten Ecke des Rechtecks enthält.
Brush	Bezeichnet den Pinsel, der zum Einrahmen des Rechtecks benutzt werden soll.

Siehe auch:

CreateHatchBrush
CreatePatternBrush
CreateSolidBrush

FrameRgn (Windows API Funktion)

Deklaration

```
function FrameRgn(DC: HDC; Rgn: HRgn; Brush: HBrush; Width, Height:
Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet eine Umrandung um eine Region.

Parameter

DC	Bezeichnet den Gerätekontext.
Rgn	Bezeichnet die Region, die in eine Umrandung eingeschlossen werden soll.
Brush	Bezeichnet den Pinsel, der zum Zeichnen der Umrandung benutzt werden soll.
Width	Gibt die Breite in vertikalen Pinselstrichen an (in virtuellen Einheiten).
Height	Gibt die Höhe in horizontalen Pinselstrichen an (in virtuellen Einheiten).

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

FreeLibrary (Windows API Prozedur)

Deklaration

```
procedure FreeLibrary(LibModule: THandle);
```

Beschreibung

Diese Funktion verringert den Referenzzähler des geladenen Bibliothekmoduls LibModule um eins. Wenn der Referenzzähler 0 erreicht, wird der vom Modul belegte Speicherplatz freigegeben.

Parameter

LibModule Bezeichnet das geladene Bibliotheksmodul.

FreeModule (Windows API Funktion)

Deklaration

```
function FreeModule(Module: THandle): Bool;
```

Beschreibung

Diese Funktion verringert den Referenzzähler des geladenen Moduls Module um eins. Wenn der Referenzzähler 0 erreicht, wird der vom Modul belegte Speicherplatz freigegeben.

Parameter

Module Bezeichnet das geladene Modul.

Rückgabewert

Wird nicht verwendet.

FreeProcInstance (Windows API Prozedur)

Deklaration

```
procedure FreeProcInstance(Proc: TFarProc);
```

Beschreibung

Gibt die Adresse der Prozedurinstanz einer Funktion frei.

Parameter

Proc	Ist die Adresse der Prozedurinstanz freizugebenden Funktion.

Siehe auch:

MakeProcInstance

FreeResource (Windows API Funktion)

Deklaration

```
function FreeResource(ResData: THandle): Bool;
```

Beschreibung

Verwirft den Bezeichner ResData und gibt den zugehörigen Speicherplatz frei, wenn die Ressource nicht mehr referenziert wird.

Parameter

ResData Bezeichnet die Daten, die der Ressource zugeordnet sind.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion fehlgeschlagen ist und die Ressource nicht freigegeben wird. Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde.

Siehe auch:

LoadResource

FreeSelector (Windows API Funktion)

Deklaration

```
function FreeSelector(Selector: Word): Word;
```

Beschreibung

Gibt einen Selektor frei, der von **AllocSelector** oder **AllocDStoCSAlias** reserviert wurde.

Parameter

Selector Freizugebender Selektor

Rückgabewert

0 bei erfolgreicher Ausführung, ansonsten wird der Selektor zurückgegeben.

GetActiveWindow (Windows API Funktion)

Deklaration

```
function GetActiveWindow: HWnd;
```

Beschreibung

Diese Funktion liefert das Handle des aktiven Fensters, das den aktuellen Eingabefokus besitzt.

Rückgabewert

Bezeichnet des aktiven Fensters.

Siehe auch:

SetActiveWindow

GetAspectRatioFilter (Windows API Funktion)

Deklaration

```
function GetAspectRatioFilter(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ermittelt die aktuellen Einstellungen für das Verhältnis von Bildschirmhöhe und -breite. Dieses Verhältnis wird durch die Breite und Höhe der Pixel eines bestimmten Geräts bestimmt.

Parameter

DC	Bezeichnet den Gerätekontext, der das angegebene Höhen-/Seitenverhältnis enthält.
Returns	Die x-Koordinate des Höhen-/Seitenverhältnis ist im höherwertigen Word, die y- Koordinate im niederwertigen Word abgelegt.

Siehe auch:

SetMapperFlags

GetAsyncKeyState (Windows API Funktion)

Deklaration

```
function GetAsyncKeyState(Key: Integer): Integer;
```

Beschreibung

Diese Funktion überprüft, ob im Augenblick des Funktionsaufrufs eine Taste gedrückt ist.

Parameter

Key	Enthält einen der 256 möglichen virtuellen Tastencodes

Rückgabewert

Ist das höchstwertige Bit (MSB) gesetzt, wird die Taste gerade gedrückt. Ist das niederwertigste Bit (LSB) gesetzt, wurde die Taste nach dem vorherigen Aufruf dieser Funktion betätigt.

GetAtomHandle (Windows API Funktion)

Deklaration

```
function GetAtomHandle(AnAtom: Atom): THandle;
```

Beschreibung

Die Funktion liefert ein Handle (relativ zum lokalen Heap) für den String, der zu dem angegebenen Atom gehört.

Parameter

AnAtom — Bezeichnet durch ein vorzeichenlos Integer das Atom, dessen Handle ermittelt werden soll.

Rückgabewert

Lokales Handle des zum Atom gehörigen String. Er ist 0, wenn kein solches Atom existiert.

GetAtomName (Windows API Funktion)

Deklaration

```
function GetAtomName(AnAtom: Atom; Buffer: PChar; Size: Integer): Word;
```

Beschreibung

Diese Funktion erstellt eine Kopie der zum Atom gehörigen Zeichenkette in dem Puffer, auf den der Parameter Buffer zeigt.

Parameter

AnAtom	Bezeichnet die Zeichenkette.
Buffer	Zeigt auf den Puffer, der die Zeichenkette aufnehmen soll.
Size	Bestimmt die maximale Größe des Puffers in Bytes.

Rückgabewert

Der Rückgabewert gibt die Anzahl von Bytes an, die in den Puffer kopiert wurden. Er ist 0, wenn das festgelegte Atom ungültig ist.

GetBitmapBits (Windows API Funktion)

Deklaration

```
function GetBitmapBits(Bitmap: HBitmap; Count: Longint; Bits: Pointer):
Longint;
```

Beschreibung

Diese Funktion kopiert die Bits des angegebenen Bitmaps in den Puffer, auf den der Parameter Bits zeigt.

Parameter

Bitmap	Bezeichnet das Bitmap.
Count	Legt die Anzahl der zu kopierenden Bytes fest.
Bits	Zeiger auf den Puffer, der das Bitmap speichern soll. Das Bitmap ist ein Array von Bytes. Dieses Array entspricht einem horizontalen Raster, das ein Vielfaches von 16 Bit darstellt.

Rückgabewert

Der Rückgabewert gibt die aktuelle Anzahl von Bytes in dem Bitmap an. Liegt ein Fehler vor, ist er 0.

GetBitmapDimension (Windows API Funktion)

Deklaration

```
function GetBitmapDimension(Bitmap: HBitmap): Longint;
```

Beschreibung

Diese Funktion gibt die Breite und Höhe eines Bitmaps zurück.

Parameter

Bitmap Bezeichnet das Bitmap.

Rückgabewert

Der Rückgabewert gibt Breite und Höhe des Bitmaps in Zehntel-Millimetern an. Der Wert der Höhe liegt im höherwertigen, der der Breite im niederwertigen Word. Wurden diese Werte nicht zuvor gesetzt, ist der Rückgabewert 0.

Siehe auch:

SetBitmapDimension

GetBkColor (Windows API Funktion)

Deklaration

```
function GetBkColor(DC: HDC): Longint;
```

Beschreibung

Die Funktion gibt die aktuelle Hintergrundfarbe des angegebenen Geräts zurück.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt den RGB-Farbwert der aktuellen Hintergrundfarbe an.

GetBkMode (Windows API Funktion)

Deklaration

```
function GetBkMode(DC: HDC): Integer;
```

Beschreibung

Diese Funktion gibt den Hintergrundmodus des festgelegten Geräts zurück. Der Hintergrundmodus wird für Texte, schraffierende Pinsel und Stiftarten verwendet, die keine durchgehende Linie ziehen.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt den aktuellen Hintergrundmodus an und hat den Wert einer der **Hintergrundmodus**-Konstanten.

GetBrushOrg (Windows API Funktion)

Deklaration

```
function GetBrushOrg(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ermittelt den aktuelle Ursprung des Pinsels im gegebenen Gerätekontext.

Parameter

DC Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt den aktuellen Ursprung des Pinsels an. Die x-Koordinate ist im niederwertigen Word, die y-Koordinate im höherwertigen Word abgelegt. Die Koordinaten werden in Geräteeinheiten angegeben.

GetCapture (Windows API Funktion)

Deklaration

```
function GetCapture: HWnd;
```

Beschreibung

Diese Funktion stellt ein Handle bereit, welches das Fenster bezeichnet, das aktuell Mauseingaben akzeptiert.

Rückgabewert

Bezeichnet das Fenster, das Mauseingaben empfängt. Er ist 0, wenn kein Fenster geöffnet ist.

Siehe auch:

SetCapture

GetCaretBlinkTime (Windows API Funktion)

Deklaration

```
function GetCaretBlinkTime: Word;
```

Beschreibung

Diese Funktion liefert die Blinkrate des Carets. Die Rate bestimmt den Zeitraum zwischen dem Aufblinken des Carets in Millisekunden.

Rückgabewert

Der Rückgabewert gibt die Blinkrate in Millisekunden an.

GetCaretPos (Windows API Prozedur)

Deklaration

```
procedure GetCaretPos(var Point: TPoint);
```

Beschreibung

Diese Funktion ermittelt die aktuelle Position des Carets in Form von Bildschirmkoordinaten und kopiert sie in die Datenstruktur, auf die der Parameter Point zeigt.

Parameter

Point	Zeigt auf eine **TPoint** -Struktur, die die Bildschirmkoordinaten des Carets aufnehmen soll.

GetCharWidth (Windows API Funktion)

Deklaration

```
function GetCharWidth(DC: HDC; FirstChar, LastChar: Word; var Buffer):
Bool;
```

Beschreibung

Diese Funktion ermittelt die Breite einzelner Zeichen in einer bestimmten Gruppe von fortlaufenden Zeichen der aktuellen Schrift.

Parameter

DC	Bezeichnet den Gerätekontext.
FirstChar	Gibt das erste Zeichen einerGruppe von fortlaufenden Zeichen an.
LastChar	Gibt das letzte Zeichen einerGruppe von fortlaufenden Zeichen an.
Buffer	Zeigt auf einen Puffer, der die Werte der Breite einerGruppe von fortlaufenden Zeichen der aktuellen Schrift aufnimmt.

Rückgabewert

Der Rückgabewert ist nach erfolgreicher Ausführung ungleich 0, andernfalls gleich 0.

GetClassInfo (Windows API Funktion)

Deklaration

```
function GetClassInfo(Instance: THandle; ClassInfo: PChar; var WndClass:
TWndClass): Bool;
```

Beschreibung

Diese Funktion ermittelt Informationen über die bezeichnete Fensterklasse. Die Felder ClassName, MenuName und Instance der Struktur **TWndClass** werden nicht zurückgegeben.

Parameter

Instance	Bezeichnet die Instanz der die Fensterklasse erzeugenden Anwendung. Zur Ermittlung von vordefinierten Windows-Klassen (z.B. Aktionsschalter oder Listen), muß Instance auf 0 gesetzt werden.
ClassName	Zeigt auf einen null-terminierten String, die den Namen der gesuchten Klasse enthält, oder auf die ID der Klasse.
WndClass	Zeigt auf die Struktur vom Typ TWndClass, in welche die Informationen über die Klasse kopiert werden.

Rückgabewert

Der Rückgabewert hat den Wert ungleich 0, wenn die Funktion eine passende Klasse gefunden und die Daten erfolgreich kopiert hat. Er ist 0, wenn keine entsprechende Klasse gefunden wurde.

GetClassLong (Windows API Funktion)

Deklaration

```
function GetClassLong(Wnd: HWnd; Index: Integer): Longint;
```

Beschreibung

Diese Funktion ermittelt den durch Index festgelegten Wert vom Typ long aus der **TWndClass**-Struktur des bezeichneten Fensters. Um auf einen bei der Erstellung der Fensterklassenstruktur zusätzlich reservierten Vier-Byte-Wert zuzugreifen, muß ein positiver Byte-Offset für den durch Index angegebenen Index verwendet werden.

Parameter

Wnd Bezeichnet das Fenster.

Index Legt den Byte-Offset des zu ermittelnden Wertes fest oder enthält den Wert der Konstante **gcl_WndProc**

Rückgabewert

Der Rückgabewert enthält den ermittelten Wert.

GetClassName (Windows API Funktion)

Deklaration

```
function GetClassName(Wnd: HWnd; ClassName: PChar; MaxCount: Integer):
Integer;
```

Beschreibung

Diese Funktion ermittelt den Klassennamen des durch den Parameter Wnd bezeichneten Fensters.

Parameter

Wnd	Bezeichnet das Fenster, dessen Klassenname ermittelt werden soll.
ClassName	Zeigt auf einen Puffer, in dem der Klassenname abgelegt wird.
MaxCount	Legt die Größe des Puffers fest.

Rückgabewert

Der Rückgabewert gibt die tatsächlich in ClassName kopierte Zeichenzahl an. Er ist 0, wenn der Klassenname ungültig ist.

GetClassWord (Windows API Funktion)

Deklaration

```
function GetClassWord(Wnd: HWnd, Index: Integer): Word;
```

Beschreibung

Diese Funktion stellt den durch Index bezeichneten Wert aus der **TWndClass** -Struktur des bezeichneten Fensters dar Um auf Zwei-Byte-Werte zuzugreifen, die bei der Erstellung der Fensterklassenstruktur zusätzlich reserviert wurden, wird ein positiver Byte-Offset, wie der durch den Parameter Index festgelegte Index verwendet. Der erste Zwei-Byte-Wert liegt im zusätzlich belegten Speicher bei 0, der folgende bei 2, usw.

Parameter

Wnd	Bezeichnet das Fenster.
Index	Bestimmt den Byte-Offset des darzustellenden Wertes oder ist eine der **gcw_xxx** Konstanten.

Rückgabewert

Der Rückgabewert gibt den gefundenen Wert an.

Siehe auch

SetClassWord

GetClientRect (Windows API Prozedur)

Deklaration

```
procedure GetClientRect(Wnd: HWnd; var Rect: TRect);
```

Beschreibung

Diese Funktion kopiert die Client-Koordinaten des Client-Bereichs eines Fensters in die Datenstruktur, auf die der Parameter Rect zeigt.

Parameter

Wnd	Bezeichnet das dem Client-Bereich zugeordnete Fenster.
Rect	Zeigt auf eine **TRect**-Datenstruktur.

GetClipboardData (Windows API Funktion)

Deklaration

```
function GetClipboardData(Format: Word): THandle;
```

Beschreibung

Diese Funktion liest Daten aus der Zwischenablage in dem durch den Parameter Format angegebenen Format. Die Zwischenablage muß vorher geöffnet worden sein.Das von der Funktion zurückgegebene Daten-Handle wird von der Zwischenablage, nicht von der Anwendung kontrolliert.

Parameter

Format	Gibt das Datenformat an, ist eine der **cf_xxx** Konstanten.

Rückgabewert

Bezeichnet des Speicherblocks mit den Daten der Zwischenablage. Liegt ein Fehler vor, ist der Rückgabewert 0.

Siehe auch:

SetClipboardData

GetClipboardFormatName (Windows API Funktion)

Deklaration

```
function GetClipboardFormatName(Format: Word; FormatName: PChar; MaxCount:
Integer): Integer;
```

Beschreibung

Diese Funktion ermittelt den Namen des registrierten Formats aus der Zwischenablage.

Parameter

Format	Bestimmt den Typ des darzustellenden Formats und hat den Wert einer der **cf_xxx**-Konstanten.
FormatName	Zeigt auf einen Puffer, in dem der Formatname abgelegt wird.
MaxCount	Gibt die Puffergröße an

Rückgabewert

Der Rückgabewert gibt die tatsächliche Länge des in den Puffer kopierten Strings an. Er ist 0, wenn das gewünschte Format nicht existiert oder zu den vordefinierten gehört.

GetClipboardOwner (Windows API Funktion)

Deklaration

```
function GetClipboardOwner: HWnd;
```

Beschreibung

Diese Funktion ermittelt das Handle des Fensters, das aktueller Besitzer der Zwischenablage ist.

Rückgabewert

Bezeichnet des Fensters, das die Zwischenablage besitzt. Ist die Zwischenablage derzeit keinem Fenster zugeordnet, ist der Wert 0.

GetClipboardViewer (Windows API Funktion)

Deklaration

```
function GetClipboardViewer: HWnd;
```

Beschreibung

Diese Funktion ermittelt das Handle des ersten Fensters in der Kette des Clipboard-Viewer.

Rückgabewert

Bezeichner des aktuell für die Darstellung der Zwischenablage verantwortlichen Fensters. Existiert kein solches Fenster, ist er 0.

GetClipBox (Windows API Funktion)

Deklaration

```
function GetClipBox(DC: HDC; var Rect: TRect): Integer;
```

Beschreibung

Diese Funktion liefert die Größe des kleinstmöglichen, die aktuelle Clipping-Grenze umgebenden Rechtecks.

Parameter

DC	Bezeichnet den Gerätekontext.
Rect	Zeigt auf eine **TRect**-Datenstruktur, in der die Größe des Rechtecks abgelegt wird.

Rückgabewert

Der Rückgabewert gibt den Typ der Clipping-Region an und enthält den Wert einer der **Regions-Flags-Konstanten**:

GetCodeHandle (Windows API Funktion)

Deklaration

```
function GetCodeHandle(Proc: TFarProc): THandle;
```

Beschreibung

Diese Funktion ermittelt, welches Code-Segment die Funktion enthält, auf die der Parameter Proc zeigt (lädt dieses, falls nötig).

Parameter

Proc	Adresse der Prozedurinstanz.

Rückgabewert

Bezeichnet das Code-Segment, das die bezeichnete Funktion enthält.

GetCommError (Windows API Funktion)

Deklaration

```
function GetCommError(Cid: Integer; var Stat: TComStat): Integer;
```

Beschreibung

Tritt ein Schnittstellenfehler auf, sperrt Windows die Schnittstelle, bis der Fehler durch die Funktion GetCommError behoben wird. Die Funktion füllt den Statuspuffer, auf den der Parameter Stat zeigt, mit dem aktuellen Status der Schnittstelle, die über den Parameter Cid angegeben ist.

Parameter

Cid	Legt fest, welche Schnittstelle überprüft werden soll.
Stat	Zeigt auf eine **TComStat**-Datenstruktur, die den Gerätestatus aufnehmen soll oder ist **nil**.

Rückgabewert

Der Rückgabewert bestimmt den Fehlercode der zuletzt ausgeführten Schnittstellenfunktion. Dabei kann es sich um einen einzelnen oder eine Kombination mehrerer Werte aus der Tabelle **ce_xxx Schnittstellenfehlercodes**, handeln.

Siehe auch:

OpenComm

GetCommEventMask (Windows API Funktion)

Deklaration

```
function GetCommEventMask(Cid, EvtMask: Integer): Word;
```

Beschreibung

Ermittelt die akutelle Ereignismaske einesGeräts und löscht diese danach.

Parameter

Cid	Legt das zu untersuchende Schnittstellengerät fest.
EvtMask	Legt fest, welche Ereignisse unterstützt werden.

Rückgabewert

Der Rückgabewert gibt den aktuellen Wert der Ereignismaske an. Jedes Bit der Ereignismaske gibt an, ob ein vorgegebenes Ereignis vorgekommen ist. In diesem Fall ist das Bit auf 1 gesetzt.

Siehe auch:

OpenComm
SetCommEventMask

GetCommState (Windows API Funktion)

Deklaration

```
function GetCommState(Cid: Integer; var DCB: TDCB): Integer;
```

Beschreibung

Parameter

Cid	Bestimmt die zu überprüfende Geräteschnittstelle.
DCB	Zeigt auf die **TDCB** -Datenstruktur, die den aktuellen Gerätesteuerblock aufnehmen soll.

Rückgabewert

Bei erfolgreicher Ausführung gleich 0, im Fehlerfall ist der Rückgabewert negativ.

Siehe auch:

OpenComm

GetCurrentPosition (Windows API Funktion)

Deklaration

function GetCurrentPosition(DC: HDC): Longint;

Beschreibung

Lädt die logischen Koordinaten der aktuellen Position.

Parameter

Der Rückgabewert gibt die aktuelle Position an. Die y-Koordinate liegt im höherwertigen Word, die x-Koordinate im niederwertigen.

GetCurrentTask (Windows API Funktion)

Deklaration

```
function GetCurrentTask :THandle;
```

Beschreibung

Lädt den Handle der aktuellen Task.

Rückgabewert

Bezeichnet die Task, wenn die Funktion erfolgreich ausgeführt wurde. Sonst ist er 0.

GetCurrentTime (Windows API Funktion)

Deklaration

```
function GetCurrentTime: Longint;
```

Beschreibung

Diese Funktion ermittelt die aktuelle Windows-Zeit., d.h. die Anzahl von Millisekunden, die seit dem Systemstart vergangen sind.

Rückgabewert

Der Rückgabewert gibt die aktuelle Zeit in Millisekunden an.

GetCursorPos (Windows API Prozedur)

Deklaration

```
procedure GetCursorPos(var Point: TPoint);
```

Beschreibung

Diese Funktion ermittelt die aktuelle Zeigerposition in Form von Bildschirmkoordinaten.

Parameter

Point	Zeigt auf die **TPoint**-Struktur, die die Bildschirmkoordinaten des Zeigers aufnehmen soll.

GetDC (Windows API Funktion)

Deklaration

```
function GetDC(Wnd: HWnd): HDC;
```

Beschreibung

Diese Funktion ermittelt ein Handle für einen Bildschirmkontext für den Client-Bereich eines gegebenen Fensters.

Parameter

Wnd Bezeichnet das Fenster, dessen Bildschirmkontext bereitgestellt werden soll.

Rückgabewert

Bezeichnet den Bildschirmkontext des Client-Bereichs des gegebenen Fensters, wenn die Funktion erfolgreich war; andernfalls ist er 0.

Siehe auch:

ReleaseDC

GetDCOrg (Windows API Funktion)

Deklaration

```
function GetDCOrg(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ermittelt den Ursprung des Bildschirmkontextes. Dieser Ursprung bestimmt den von Windows verwendeten Offset zur Übersetzung von Gerätekoordinaten in Client-Koordinaten von Punkten in einem Anwendungsfenster.

Parameter

DC	Bezeichnet den Gerätekontext, dessen Ursprung ermittelt werden soll.

Rückgabewert

Der Rückgabewert gibt den Ursprung in Gerätekoordinaten an. Die y-Koordinate liegt im höherwertigen Word, die x-Koordinate im niederwertigen.

GetDesktopWindow (Windows API Funktion)

Deklaration

```
function GetDesktopWindow: HWnd;
```

Beschreibung

Diese Funktion liefert das Fenster-Handle für das Desktop-Fenster von Windows.

Rückgabewert

Bezeichnet das Desktop-Fenster von Windows.

GetDeviceCaps (Windows API Funktion)

Deklaration

```
function GetDeviceCaps(DC: HDC; Index: Integer): Integer;
```

Beschreibung

Diese Funktion stellt gerätespezifische Informationen über einen gegebenen Bildschirm dar.

Parameter

DC	Bezeichnet den Gerätekontext.
Index	Bestimmt die zurückzugebende Information. Dabei kann es sich um einen beliebigen Wert aus der Tabelle **GDI-Informationsindizes** handeln.

Rückgabewert

Der Rückgabewert gibt den Wert der gewünschten Information an.

GetDialogBaseUnits (Windows API Funktion)

Deklaration

```
function GetDialogBaseUnits: Longint;
```

Beschreibung

Diese Funktion gibt die von Windows zur Erstellung von Dialogfenstern verwendeten Dialogbasiseinheiten zurück. Diese Werte sollten zur Berechnung der durchschnittlichen Zeichenbreite einer Schrift von der Anwendung verwendet werden.

Rückgabewert

Der Rückgabewert gibt die Dialogbasiseinheiten an. Das höherwertige Word enthält die Höhe in Pixel, abgeleitet aus der Höhe der Systemschrift. Das niederwertige Word enthält die Breite in Pixel, abgeleitet aus der Breite der Systemschrift.

GetDIBits (Windows API Funktion)

Deklaration

```
function GetDIBits(DC: HDC; Bitmap: THandle; StartScan, NumScans: Word;
Bits: Pointer; var BitInfo: TBitmapInfo; Usage: Word): Integer;
```

Beschreibung

Diese Funktion ermittelt die Bits des angegebenen Bitmaps und kopiert sie in geräteunabhängigem Format in den durch Bits bezeichneten Puffer.

Parameter

DC	Legt den Gerätekontext fest.
Bitmap	Legt das Bitmap fest.
StartScan	Bestimmt die Anordnung der ersten Scan-Zeile im bezeichneten Bitmap.
NumScans	Bestimmt die Anzahl der zu kopierenden Zeilen.
Bits	Zeigt auf den Puffer, der die Bitmap-Daten im geräteunabhängigen Format aufnimmt.
BitsInfo	Zeigt auf die Datenstruktur **TBitmapInfo**, die die Informationen zum Farbformat und die Dimensionierung des Bitmaps enthält.
Usage	Enthält eine **Dib_xxx**-Konstanten.

Rückgabewert

Der Rückgabewert bestimmt die Anzahl der Scan-Zeilen, die aus dem Bitmap kopiert wurden. Er ist 0, wenn ein Fehler aufgetreten ist.

GetDlgCtrlID (Windows API Funktion)

Deklaration

```
function GetDlgCtrlID(Wnd: HWnd): Integer;
```

Beschreibung

Diese Funktion gibt den ID-Wert des durch den Parameter Wnd bezeichneten Fensters zurück.

Parameter

Wnd Bezeichnet das untergeordnete Fenster.

Rückgabewert

Ist die Funktion erfolgreich, entspricht der Rückgabewert der numerischen Identifikation des untergeordneten Fensters. Im Fehlerfall ist der Rückgabewert 0.

GetDlgItem (Windows API Funktion)

Deklaration

```
function GetDlgItem(Dlg: HWnd; IDDlgItem: Integer): HWnd;
```

Beschreibung

Diese Funktion liefert das Handle eines im Dialogfenster enthaltenen Steuerelements. Das Dialogfenster wird durch den Parameter Dlg bestimmt.

Parameter

Dlg	Bezeichnet das Dialogfenster, in dem das Steuerelement enthalten ist.
IDDlgItem	Gibt die Integer-ID des zu ermittelnden Elements an.

Rückgabewert

Bezeichnet das vorgegebene Steuerelement. Er ist 0, wenn kein Steuerelement mit der durch IDDlgItem gegebenen Integer-ID existiert.

GetDlgItemInt (Windows API Funktion)

Deklaration

```
function GetDlgItemInt(Dlg: HWnd; IDDlgItem: Integer; Translate: LPBool;
Signed: Bool): Word;
```

Beschreibung

Diese Funktion übersetzt den Text eines Steuerelements des gegebenen Dialogfensters in einen Integer. Sie übersetzt den Text ohne Berücksichtigung etwaiger Leerzeichen am Anfang des Textes.

Parameter

Dlg	Bezeichnet das Dialogfenster.
IDDlgItem	Bestimmt den Integer-Bezeichner des zu übersetzenden Objekts des Dialogfensters.
Translate	Zeigt auf eine Variable vom Typ **Bool**, die das übersetzte Flag aufnimmt.
Signed	Legt fest, ob der zu ermittelnde Wert vom Typ Signed ist.

Rückgabewert

Der Rückgabewert gibt den übersetzten Wert des Textes des Dialogfensterelements an.

Siehe auch:

wm_GetText

GetDlgItemText (Windows API Funktion)

Deklaration

```
function GetDlgItemText(Dlg: HWnd; IDDlgItem: Integer; Str: PChar;
MaxCount: Integer): Integer;
```

Beschreibung

Diese Funktion ermittelt den zu einem Steuerelement in einem Dialogfenster gehörigen Text.

Parameter

Dlg	Bezeichnet das Dialogfenster mit dem gewünschten Steuerelement.
IDDlgItem	Gibt den Integer-Bezeichner des Objekts an
Str	Zeigt auf den Puffer, der den Text aufnimmt.
MaxCount	Gibt die Puffergröße an.

Rückgabewert

Der Rückgabewert gibt die Anzahl der tatsächlich in den Puffer kopierten Zeichen an. Er ist 0, wenn kein Text kopiert wurde.

Siehe auch:

wm_GetText

GetDriveType (Windows API Funktion)

Deklaration

```
function GetDriveType(Drive: Integer): Word;
```

Beschreibung

Diese Funktion stellt fest, ob in einem Laufwerk austauschbare oder fest installierte Speichermedien verwendet werden, oder ob es sich um ein Netzlaufwerk handelt.

Parameter

Drive — Bestimmt das Laufwerk, dessen Typ ermittelt werden soll. Laufwerk A: hat den Wert 0, B: den Wert 1, C: den Wert 2 usw.

Rückgabewert

Der Rückgabewert gibt den Laufwerkstyp an.(**drive_xxx**-Konstante).
Der Rückgabewert ist 0, wenn die Funktion den Laufwerkstyp nicht ermitteln kann. Er ist 1, wenn das angegebene Laufwerk nicht existiert.

GetEnvironment (Windows API Funktion)

Deklaration

```
function GetEnvironment(PortName, Environ: PChar; MaxCount: Word):
Integer;
```

Beschreibung

Diese Funktion ermittelt die aktuelle Umgebung der an die bezeichnete Systemschnittstelle angeschlossenen Geräteeinheit.

Parameter

PortName	Zeigt auf den null-terminierten String, der den Namen der gewünschten Schnittstelle enthält.
Environ	Zeigt auf den Puffer, der die Umgebungsbezeichnung aufnehmen soll (das erste Feld muß den Gerätenamen enthalten) oder **nil**, wenn die erforderliche Größe zurückgegeben wird.
MaxCount	Bestimmt die maximale Größe des Puffers.

Rückgabewert

Der Rückgabewert gibt die Anzahl der in den Puffer kopierten Bytes. Kann die Umgebung nicht gefunden werden, ist der Rückgabewert 0.

GetFocus (Windows API Funktion)

Deklaration

```
function GetFocus: HWnd;
```

Beschreibung

Diese Funktion ermittelt das Handle des Fensters, das aktuell den Fokus besitzt.

Rückgabewert

Bezeichner des Fensters, das akuell den Fokus besitzt. Konnte die Funktion nicht erfolgreich ausgeführt werden, ist der Rückgabewert 0.

GetFreeSpace (Windows API Funktion)

Deklaration

```
function GetFreeSpace(Flag: Word): Longint;
```

Beschreibung

Diese Funktion untersucht den globalen Heap und gibt den zur Verfügung stehenden Speicher in Bytes zurück.

Parameter

Flags Legt fest, ob der Heap über oder unter der EMS-Speicherabgrenzung in Large-Frame- und Small-Frame-EMS-Systemen geprüft wird. Ist der Parameter auf **gmem_Not_Banked** gesetzt, gibt die Funktion die Größe des verfügbaren Speichers unterhalb der Abgrenzung zurück. Der Wert oberhalb der Abgrenzung wird zurückgegeben, wenn Flags den Wert 0 hat. In Systemen ohne EMS wird dieser Parameter ignoriert.

Rückgabewert

Der Rückgabewert entspricht derGröße des verfügbaren Speichers in Byte.

Siehe auch:

GlobalCompact

GetInputState (Windows API Funktion)

Deklaration

```
function GetInputState: Bool;
```

Beschreibung

Diese Funktion untersucht, ob in der Systemwarteschlange Maus-, Tastatur- oder Timer-Ereignisse zur Bearbeitung vorliegen.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn Maus-, Tastatur- oder Timer-Ereignisse vorliegen, ansonsten ist er 0.

GetInstanceData (Windows API Funktion)

Deklaration

```
function GetInstanceData(Instance: THandle; Data, Count: Word): Integer;
```

Beschreibung

Diese Funktion kopiert Daten einer vorhergehenden Instanz der Anwendung in den Datenbereich der aktuellen Instanz.

Parameter

Instance	Bezeichnet eine vorhergehende Instanz der Anwendung.
Data	Zeigt auf einen Puffer in der aktuellen Instanz.
Count	Legt die Anzahl der zu kopierenden Bytes fest.

Rückgabewert

Der Rückgabewert gibt die Anzahl der kopierten Bytes an.

GetKBCodePage (Windows API Funktion)

Deklaration

```
function GetKBCodePage: Integer;
```

Beschreibung

Diese Funktion ermittelt, welche OEM/ANSI-Tabellen von Windows geladen sind.

Rückgabewert

Der Rückgabewert gibt die momentan von Windows geladene Code-Seite an.
Möglich sind folgende Werte:

437	Standard (USA)
850	International
860	Portugal
861	Island
863	Franko-Kanadisch
865	Norwegen/Dänemark

GetKeyboardState (Windows API Prozedur)

Deklaration

```
procedure GetKeyboardState(var KeyState: Byte);
```

Beschreibung

Diese Funktion kopiert den Status der 256 möglichen Tasten der virtuellen Tastatur in den durch KeyState bezeichneten Puffer. Das obere Bit eines jeden Bytes wird auf 1 gesetzt, wenn die Taste gedrückt ist, auf 0, wenn die Taste nicht betätigt wird. Das untere Bit wird auf 1 gesetzt, wenn die Taste seit dem Startup mehrmals betätigt wurde.

Parameter

KeyState Zeigt auf den 256-Byte-Puffer der virtuellen Tasten-Codes.

GetKeyboardType (Windows API Funktion)

Deklaration

```
function GetKeyboardType(TypeFlag: Integer): Integer;
```

Beschreibung

Diese Funktion ermittelt den Typ der vom System verwendeten Tastatur.

Parameter

TypeFlag Legt fest, ob der Rückgabewert der Funktion den Typ oder den Untertyp der Tastatur anzeigt. Folgende Werte sind möglich:

0	Funktion gibt den Typ der Tastatur zurück.
1	Funktion gibt den Untertyp der Tastatur zurück.
2	Funktion gibt die Anzahl der Funktionstasten der Tastatur zurück.

Rückgabewert

1	IBM PC/XT oder kompatible Tastatur (83 Tasten)
2	Olivetti M24 "ICO" Tastatur (102 Tasten)
3	IBM AT oder vergleichbare Tastatur (84 Tasten)
4	IBM Erweiterte Tastatur (101 oder 102 Tasten)
5	Nokia 1050 und vergleichbare Tastaturen
6	Nokia 9140 und vergleichbare Tastaturen

GetKeyNameText (Windows API Funktion)

Deklaration

```
function GetKeyNameText(lParam: Longint; Buffer: PChar; Size: Integer): Integer;
```

Beschreibung

Diese Funktion ermittelt einen String, der den Namen einer Taste enthält.

Parameter

lParam	Enthält den 32-Bit-Parameter der **wm_KeyDown** Tastaturbotschaft
Buffer	Bestimmt einen Puffer, der den Tastennamen aufnehmen soll.
Size	Gibt die Größe des Puffers an.

Rückgabewert

Der Rückgabewert entspricht der tatsächlichen Länge des in Buffer kopierten Strings.

GetKeyState (Windows API Funktion)

Deklaration

```
function GetKeyState(VirtKey: Integer): Integer;
```

Beschreibung

Diese Funktion ermittelt den Status der durch den Parameter VirtKey bestimmten virtuellen Taste. Der Status zeigt an, ob die Taste gedrückt oder arretiert ist.

Parameter

VirtKey Gibt eine virtuelle Taste an.

Rückgabewert

Der Rückgabewert gibt den Status der vorgegebenen virtuellen Taste an. Ist das höherwertige Bit 1, wird die Taste gedrückt. Ist das niederwertige Bit 1, ist die Taste arretiert.

GetLastActivePopup (Windows API Funktion)

Deklaration

```
function GetLastActivePopup(WndOwner: HWnd): HWnd;
```

Beschreibung

Diese Funktion ermittelt, welches zu dem durch WndOwner bezeichneten Fenster gehörige Pop-up-Fenster zuletzt aktiviert war.

Parameter

WndOwnerBezeichnet das besitzende Fenster.

Rückgabewert

Bezeichnet das zuletzt aktivierte Pop-up-Fenster. Der Rückgabewert ist identisch mit dem Parameter WndOwner.

GetMapMode (Windows API Funktion)

Deklaration

```
function GetMapMode(DC: HDC): Integer;
```

Beschreibung

Diese Funktion ermittelt den aktuellen Abbildungsmodus.

Parameter

DC Bezeichnet den Gerätekontext.

Rückgabewert

Eine Konstante vom Typ **mm_xxx** an.

Siehe auch:

SetMapMode

GetMenu (Windows API Funktion)

Deklaration

```
function GetMenu(Wnd: HWnd): HMenu;
```

Beschreibung

Diese Funktion ermittelt das Handle des Menüs im gegebenen Fenster.

Parameter

Wnd Bezeichnet das Fenster, dessen Menü untersucht werden soll.

Rückgabewert

Bezeichnet das Menü. Er ist 0, wenn das Fenster kein Menü besitzt. Wenn Wnd ein untergeordnetes Fenster bezeichnet, ist der Rückgabewert nicht definiert.

GetMenuCheckMarkDimensions (Windows API Funktion)

Deklaration

```
function GetMenuCheckMarkDimensions: Longint;
```

Beschreibung

Diese Funktion gibt die Größe des Bitmaps der Standardauswahlmarkierung zurück. Windows stellt dieses Bitmap direkt neben den markierten Menüeinträgen dar.

Rückgabewert

Der Rückgabewert gibt Höhe und Breite des Bitmaps der Standardauswahlmarkierung an. Das höherwertige Word enthält die Höhe in Pixel, das niederwertige die Breite.

Siehe auch:

SetMenuItemBitmaps

GetMenuItemCount (Windows API Funktion)

Deklaration

```
function GetMenuItemCount(Menu: HMenu): Word;
```

Beschreibung

Diese Funktion stellt die Anzahl der Einträge in dem durch Menu bezeichneten Menü fest. Es kann sich um ein Pop-up-Menü oder ein Hauptmenü handeln.

Parameter

Menu Bezeichnet das Handle des zu untersuchenden Menüs.

Rückgabewert

Der Rückgabewert gibt die Anzahl der Menüeinträge im bezeichneten Menü an. Wurde die Funktion nicht erfolgreich ausgeführt, ist der Rückgabewert -1.

GetMenuItemID (Windows API Funktion)

Deklaration

```
function GetMenuItemID(Menu: HMenu; Pos: Integer): Word;
```

Beschreibung

Diese Funktion ermittelt den Bezeichner des an der durch Pos vorgegebenen Position befindlichen Menüeintrags.

Parameter

Menu	Bezeichnet das Handle des Pop-up-Menüs, das den Eintrag enthält, dessen Bezeichner ermittelt wird.
Pos	Gibt die Position des zum Bezeichner gehörenden Menüeintrags (ausgehend von 0) an.

Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, gibt der Rückgabewert die ID eines Eintrags in einem Pop-up-Menü an. Hat der Parameter Menu den Wert 0 oder ist der betreffende Eintrag selbst ein Pop-up-Menü, ist der Rückgabewert -1.

GetMenuState (Windows API Funktion)

Deklaration

```
function GetMenuState(Menu: HMenu; ID, Flags: Word): Word;
```

Beschreibung

Diese Funktion ermittelt die Anzahl der Einträge im Pop-up-Menü, das mit dem durch ID bezeichneten Menüeintrag verbunden ist.

Parameter

Menu	Bezeichnet das Menü.
ID	Gibt die ID des Menüeintrags an.
Flags	Eine **mf_xxx**-Konstante: MF_BYPOSITION oder MF_BYCOMMAND.

Rückgabewert

Der Rückgabewert ist -1, wenn der angegebene Eintrag nicht existiert. Bezeichnet ID ein Pop-up-Menü, enthält der Rückgabewert die Anzahl der Menüeinträge im höherwertigen Byte und die zum Menü gehörigen Flags **mf_xxx**im niederwertigen Byte.

MF_CHECKED
MF_DISABLED
MF_ENABLED
MF_GRAYED
MF_MENUBARBREAK
MF_MENUBREAK
MF_SEPARATOR
MF_UNCHECKED

GetMenuString (Windows API Funktion)

Deklaration

```
function GetMenuString(Menu: HMenu; IDItem: Word; Str: PChar; MaxCount:
Integer; Flag: Word): Integer;
```

Beschreibung

Diese Funktion kopiert das Label des angegebenen Menüeintrags in den Parameter String. Das kopierte Label ist null-terminiert.

Parameter

Menu	Bezeichnet das Menü.
IDItem	Bestimmt den Bezeichner (Integer) des Menüeintrags
Str	Zeigt auf den Puffer, der das Label aufnimmt.
MaxCount	Bestimmt die maximale Länge des zu kopierenden Labels.
Flag	Eine **mf_xxx**-Konstante: MF_BYPOSITION oder MF_BYCOMMAND

Rückgabewert

Der Rückgabewert entspricht der Anzahl der tatsächlich in den Puffer kopierten Bytes.

GetMessage (Windows API Funktion)

Deklaration

```
function GetMessage(var Msg: TMsg; Wnd: HWnd; MsgFilterMin, MsgFilterMax:
Word): Bool;
```

Beschreibung

Diese Funktion ruft eine Botschaft von der Anwendungsschlange ab und legt diese Botschaft in der Datenstruktur ab, auf die der Parameter Msg zeigt. Wenn keine Botschaft verfügbar ist, dann übergibt die Funktion die Steuerung an andere Anwendungen, bis eine Botschaft verfügbar wird oder **wm_Paint** oder **wm_Timer** die nächste Meldung ist.

Parameter

Msg	Zeigt auf eine Datenstruktur vom Typ TMsg.
Wnd	Bezeichnet das Fenster, an das Botschaften gesendet werden, oder 0 für alle Fenster der Anwendung.
MsgFilterMin	Gibt den Integerwert des niedrigsten abzurufenden Botschaftswerts an.
MsgFilterMax	Gibt den Integerwert des höchsten abzurufenden Botschaftswerts an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn eine andere Botschaft als **wm_Quit** abgerufen wurde, andernfalls ist er 0.

GetMessagePos (Windows API Funktion)

Deklaration

```
function GetMessagePos: Longint;
```

Beschreibung

Diese Funktion gibt einen long-Wert zurück, der die Zeigerposition in Bildschirmkoordinaten zu dem Zeitpunkt darstellt, zu dem die letzte von der Funktion **GetMessage** abgerufene Botschaft eintraf.

Rückgabewert

Der Rückgabewert gibt die x- und y-Koordinaten der Zeigerposition zurück. Die x-Koordinate liegt im niederwertigen Word und die y-Koordinate im höherwertigen.

GetMessageTime (Windows API Funktion)

Deklaration

```
function GetMessageTime: Longint;
```

Beschreibung

Diese Funktion gibt die Zeit der Erzeugung der letzten von der Funktion **GetMessage** abgerufenen Botschaft an.

Rückgabewert

Der Rückgabewert gibt die Zeit der Erzeugung der Botschaft an (in Millisekunden).

GetMetaFile (Windows API Funktion)

Deklaration

```
function GetMetaFile(FileName: PChar): THandle;
```

Beschreibung

Diese Funktion erzeugt ein Handle für die Metadatei, die vom Parameter Filename benannt wird.

Parameter

Filename — Zeigt auf den null-terminierten String, der den DOS-Dateinamen der Metadatei angibt.

Rückgabewert

Bezeichnet eine Metadatei, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

GetMetaFileBits (Windows API Funktion)

Deklaration

```
function GetMetaFileBits(MF: THandle): THandle;
```

Beschreibung

Diese Funktion gibt ein Handle auf einen globalen Speicherblock zurück, der die angegebene Metadatei als eine Kollektion von Bits enthält. Der Speicherblock kann benutzt werden, um die Größe der Metadatei zu bestimmen und die Metadatei abzuspeichern.

Parameter

MF	Bezeichnet die Speicher-Metadatei, wird nach dem Aufruf ungültig.

Rückgabewert

Bezeichnet den globalen Speicherblock, der die Metadatei enthält. Wenn ein Fehler auftritt, ist der Rückgabewert 0.

GetModuleFileName (Windows API Funktion)

Deklaration

```
function GetModuleFileName(Module: THandle; FileName: PChar; Size:
Integer): Integer;
```

Beschreibung

Diese Funktion ruft den vollständigen Pfadnamen der ausführbaren Datei ab, aus der das angegebene Modul geladen wurde. Die Funktion kopiert den null-terminierten Dateinamen in den Puffer, auf den der Parameter Filename zeigt.

Parameter

Module	Bezeichnet das Modul oder die Instanz des Moduls.
Filename	Zeigt auf den Puffer, der den Dateinamen empfangen soll.
Size	Gibt die maximale Anzahl von Zeichen an, die kopiert werden sollen.

Rückgabewert

Der Rückgabewert gibt die tatsächliche Länge des in den Puffer kopierten Strings an.

GetModuleHandle (Windows API Funktion)

Deklaration

```
function GetModuleHandle(ModuleName: PChar): THandle;
```

Beschreibung

Diese Funktion ruft das Modul-Handle des angegebenen Moduls ab.

Parameter

ModuleName Zeigt auf einen null-terminierten String, der das Modul bezeichnet.

Rückgabewert

Bezeichnet das Modul, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

GetModuleUsage (Windows API Funktion)

Deklaration

```
function GetModuleUsage(Module: THandle): Integer;
```

Beschreibung

Diese Funktion gibt den Referenzzähler eines angegebenen Moduls zurück.

Parameter

Module Bezeichnet das Modul oder die Instanz des Moduls.

Rückgabewert

Bezeichnet den Referenzzähler des Moduls.

GetNearestColor (Windows API Funktion)

Deklaration

```
function GetNearestColor(DC: HDC; Color: TColorRef): Longint;
```

Beschreibung

Diese Funktion gibt die vom angegebenen Gerät darstellbare virtuelle Farbe zurück, die der angegebenen virtuellen Farbe am ehesten entspricht.

Parameter

DC	Bezeichnet den Gerätekontext.
Color	**TColorRef**-Datenstruktur, die die Farbe angibt, der entsprochen werden soll.

Rückgabewert

Der Rückgabewert zeigt einen RGB-Farbwert an, der eine Flächenfarbe bezeichnet,

GetNearestPaletteIndex (Windows API Funktion)

Deklaration

```
function GetNearestPaletteIndex(Palette: HPalette; Color: TColorRef):
Word;
```

Beschreibung

Diese Funktion gibt den Index des Eintrags in einer virtuellen Palette zurück, der dem Farbwert Color am ehesten entspricht.

Parameter

Palette	Bezeichnet die virtuelle Palette.
Color	**TColorRef**-Datenstruktur, die die Farbe angibt, der entsprochen werden soll.

Rückgabewert

Der Rückgabewert ist der Index des Eintrags in einer virtuellen Palette. Der Eintrag enthält die Farbe, die der angegebenen Farbe am nächsten kommt.

GetNextDlgGroupItem (Windows API Funktion)

Deklaration

```
function GetNextDlgGroupItem(Dlg: HWnd; Ctrl: HWnd; Previous: Bool): HWnd;
```

Beschreibung

Diese Funktion sucht ab Ctrl zyklisch nach dem nächsten (oder vorherigen) Steuerelement innerhalb einerGruppe von Steuerelementen mit dem Stil **ws_Group** in dem vom Parameter Dlg bezeichneten Dialogfenster.

Parameter

Dlg	Bezeichnet das Dialogfenster.
Ctrl	Bezeichnet das Steuerelement im Dialogfenster, bei dem die Suche beginnt.
Previous	Wenn dieser Parameter 0 ist, dann sucht die Funktion nach dem vorhergehenden Steuerelement in derGruppe, sonst nach dem nächsten.

Rückgabewert

Bezeichnet das nächste oder vorangegangene Steuerelement in derGruppe.

GetNextDlgTabItem (Windows API Funktion)

Deklaration

```
function GetNextDlgTabItem(Dlg: HWnd; Ctrl: HWnd; Previous: Bool): HWnd;
```

Beschreibung

Diese Funktion sucht zyklisch nach dem ersten Steuerelement, das den Stil **ws_TabStop** hat und dem vom Parameter Ctrl bezeichneten Steuerelement vorangeht oder darauf folgt.

Parameter

Dlg	Bezeichnet das Dialogfenster.
Ctrl	Bezeichnet das Steuerelement, das den Startpunkt der Suche bildet.
Previous	Wenn dieser Parameter 0 ist, sucht die Funktion das vorhergehende Steuerelement im Dialogfenster. Ist Previous ungleich 0, das nächste.

Rückgabewert

Bezeichnet das vorherige (oder nächste) Steuerelement.

GetNextWindow (Windows API Funktion)

Deklaration

```
function GetNextWindow(Wnd: HWnd; Flag: Word): HWnd;
```

Beschreibung

Diese Funktion sucht nach einem Handle, das ausgehend von Wnd das nächste (oder vorherige) Fenster in der Liste des Fenster-Managers bezeichnet.

Parameter

Wnd	Bezeichnet das aktuelle Fenster.
Flag	**gw_xxx Konstante**: GW_HWNDNEXT GW_HWNDPREV

Rückgabewert

Bezeichnet das nächste (oder vorherige) Fenster in der Liste des Fenster-Managers.

GetNumTasks (Windows API Funktion)

Deklaration

```
function GetNumTasks: Word;
```

Beschreibung

Diese Funktion gibt die Anzahl von Tasks zurück, die gerade im System ablaufen. Eine Task ist eine einzelne Instanz einer Windows-Anwendung.

Rückgabewert

Der Rückgabewert gibt die Anzahl der Tasks an.

GetObject (Windows API Funktion)

Deklaration

```
function GetObject(hObject: THandle; Count: Integer; ObjectPtr: Pointer):
Integer;
```

Beschreibung

Diese Funktion füllt einen Puffer mit den virtuellen Daten, die das bezeichnete virtuelle Objekt definieren.

Parameter

Object	Bezeichnet ein virtuelles Objekt
Count	Gibt die Anzahl von Bytes an, die in den Puffer kopiert werden sollen.
ObjectPtr	Zeigt auf den Puffer, der die Information empfangen soll. **TLogPen**, **TLogBrush**, **TLogFont**, **TBitmap** oder ein Integerwert.

Rückgabewert

Der Rückgabewert gibt die tasächliche Anzahl kopierter Bytes an. Er ist 0, wenn ein Fehler auftritt.

Siehe auch

GetBitmapBits
GetPaletteEntries

GetPaletteEntries (Windows API Funktion)

Deklaration

```
function GetPaletteEntries(Palette: HPalette; StartIndex, NumEntries:
Word; var PaletteEntries): Word;
```

Beschreibung

Diese Funktion ruft einen Bereich von Paletteneinträgen in einer virtuellen Palette ab und kopiert sie in PaletteEntries.

Parameter

Palette	Bezeichnet die virtuelle Palette.
StartIndex	Bezeichnet den ersten Eintrag in der virtuellen Palette,
NumEntries	Gibt die Zahl der Einträge in der virtuellen Palette an.
PaletteEntries	Zeigt auf ein Array von **TPaletteEntry** Datenstrukturen, die die Paletteneinträge aufnehmen sollen.

Rückgabewert

Der Rückgabewert ist die Anzahl von Einträgen, die aus der virtuellen Palette abgerufen wurden. Er ist 0, wenn die Funktion fehlschlägt.

GetParent (Windows API Funktion)

Deklaration

```
function GetParent(Wnd: HWnd): HWnd;
```

Beschreibung

Diese Funktion ruft das Fenster-Handle des übergeordneten Fensters (falls vorhanden) des angegebenen Fensters ab.

Parameter

Wnd	Bezeichnet das Fenster, für welches das Handle eines übergeordneten Fensters abgerufen werden soll.

Rückgabewert

Bezeichnet das übergeordnete Fenster. Er ist 0, wenn das Fenster kein übergeordnetes Fenster hat.

GetPixel (Windows API Funktion)

Deklaration

```
function GetPixel(DC: HDC; X, Y; Integer): Longint;
```

Beschreibung

Diese Funktion ruft den RGB-Farbwert des Pixels an dem von den Parametern X und Y bezeichneten Punkt ab.

Parameter

DC Bezeichnet den Gerätekontext.
X, YZu prüfender Punkt

Rückgabewert

Bezeichner einen RGB-Farbwert für die Farbe des angegebenen Punktes. Er ist -1, wenn die Koordinaten keinen Punkt in der Clipping-Region bezeichnen.

GetPolyFillMode (Windows API Funktion)

Deklaration

```
function GetPolyFillMode(DC: HDC): Integer;
```

Beschreibung

Diese Funktion ruft den aktuellen Polygon-Füllmodus ab.

Parameter

DC Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt den Polygon-Füllmodus an. Er kann eine der **PolyFüllmodus-Konstanten** als Wert annehmen.

GetPriorityClipboardFormat (Windows API Funktion)

Deklaration

```
function GetPriorityClipboardFormat(var PriorityList; Count: Integer):
Integer;
```

Beschreibung

Diese Funktion gibt das erste Zwischenablageformat einer Liste zurück, für das Daten existieren.

Parameter

PriorityList Zeigt auf ein Integer-Array, das eine Liste von Zwischenablageformaten vom Typ **cf_xxx** enthält.

Count Gibt die Anzahl der Einträge in PriorityList an.

Rückgabewert

Der Rückgabewert ist das Zwischenablageformat höchster Priorität in der Liste, für das Daten vorhanden sind. Wenn in der Zwischenablage keine Daten vorhanden sind, dann gibt diese Funktion 0 zurück. Wenn in der Zwischenablage Daten vorhanden sind, die zu keinem Format in der Liste passen, dann ist der Rückgabewert -1.

GetPrivateProfileInt (Windows API Funktion)

Deklaration

```
function GetPrivateProfileInt(ApplicationName, KeyName: PChar; Default:
Integer; FileName: PChar): Word;
```

Beschreibung

Diese Funktion kopiert eine Schlüsselanweisung aus der Windows-Initialisierungsdatei WIN.INI.

Parameter

ApplicationName	Zeigt auf einen String in FileName, der die Anwendung bezeichnet.
KeyName	Zeigt auf einen String in FileName, der eine Schlüsselanweisung bezeichnet.
Default	Setzt den voreingestellten Wert für die angegebene Schlüsselanweisung, falls KeyName nicht gefunden wird an, wenn die Schlüsselanweisung nicht gefunden werden kann.
FileName	Name der Initialisierungsdatei im Windows-Verzeichnis.

Rückgabewert

Schlüsselwert,
0, wenn negativ oder kein Integer.

GetPrivateProfileString (Windows API Funktion)

Deklaration

```
function GetPrivateProfileString(ApplicationName, KeyName, Default,
ReturnedString: PChar; Size: Integer; FileName: PChar): Integer;
```

Beschreibung

Diese Funktion kopiert eine Zeichenkette aus der angegebenen Initialisierungsdatei in einen Puffer, auf den der Parameter ReturnedString zeigt.

Parameter

ApplicationName	Name einer Windows-Anwendung in FileName
KeyName	Zeigt auf einen Schlüsselnamen in FileName oder ist **nil** und liefert dann eine Liste der Schlüsselnamen
Default	Gibt den Standardwert an, wenn die KeyName nicht gefunden werden kann.
ReturnedString	Zeigt auf den Puffer, der die Zeichenkette empfängt.
Size	Gibt maximale Anzahl von Zeichen an, die in den Puffer kopiert werden.
FileName	Initialisierungsdatei

Rückgabewert

Der Rückgabewert gibt die Anzahl der in den Puffer kopierten Zeichen an.

GetProcAddress (Windows API Funktion)

Deklaration

```
function GetProcAddress(Module: THandle; ProcName: PChar): TFarProc;
```

Beschreibung

Diese Funktion ruft die Speicheradresse einer exportierten Bibliotheks-Funktion ab, auf deren Bezeichnung der Parameter ProcName zeigt.

Parameter

Module	Bibliotheksmodul
ProcName	Funktionsname

Rückgabewert

Eintrittsadresse, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

GetProfileInt (Windows API Funktion)

Deklaration

```
function GetProfileInt(AppName, KeyName: PChar; Default: Integer):
Integer;
```

Beschreibung

Diese Funktion ruft den Wert einer Integerschlüsselanweisung aus der Windows-Initialisierungsdatei WIN.INI ab.

Parameter

AppName	Zeigt auf die Bezeichnung einer Windows-Anwendung
KeyName	Zeigt auf einen Schlüsselnamen
Default	Standardwert , wenn die Schlüsselanweisung KeyName nicht gefunden werden kann.

Rückgabewert

Der Rückgabewert ist 0, wenn der zur angegebenen Schlüsselbezeichnung gehörige Wert kein Integerwert oder ein negativer Integerwert ist.

GetProfileString (Windows API Funktion)

Deklaration

```
function GetProfileString(AppName, KeyName, Default, ReturnedString:
PChar; Size: Integer): Integer;
```

Beschreibung

Diese Funktion kopiert eine Zeichenkette aus der Windows- Initialisierungsdatei WIN.INI in den Puffer, auf den der Parameter ReturnedString zeigt.

Parameter

AppName	Name der Anwendung
KeyName	Zeigt auf eine Schlüsselanweisung oder ist **nil** und liefert dann alle Schlüsselnamen, die AppName zugeordnet sind.
Default	Gibt den voreingestellten Wert an, wenn KeyName nicht gefunden werden kann.
ReturnedString	Zeigt auf den Puffer, der die Zeichenkette empfängt.
Size	Gibt die Anzahl der Zeichen an, die in den Puffer kopiert werden.

Rückgabewert

Der Rückgabewert gibt die Anzahl der in den Puffer kopierten Zeichen an.

GetProp (Windows API Funktion)

Deklaration

```
function GetProp(Wnd: HWnd; Str: PChar): THandle;
```

Beschreibung

Diese Funktion ruft ein Daten-Handle aus der Besitzerliste des angegebenen Fensters ab.

Parameter

Wnd	Bezeichnet das Fenster, dessen Besitzerliste durchsucht werden soll.
Stri	Zeigt auf einen null-terminierten String oder ein Atom.

Rückgabewert

Bezeichnet das zugehörige Daten-Handle, wenn die Besitzerliste den angegebenen String enthält. Andernfalls ist er 0.

Siehe auch

AddAtom

GetROP2 (Windows API Funktion)

Deklaration

```
function GetROP2(DC: HDC): Integer;
```

Beschreibung

Diese Funktion ruft den aktuellen Zeichenmodus ab.

Parameter

DC Bezeichnet den Gerätekontext für ein rasterorientiertesGerät.

Rückgabewert

Der Rückgabewert gibt den Zeichenmodus an, bezeichnet eine der **r2_ binären Rasteroperationen**.

Siehe auch

SetROP2

GetScrollPos (Windows API Funktion)

Deklaration

```
function GetScrollPos(Wnd: HWnd; Bar: Integer): Integer;
```

Beschreibung

Diese Funktion ruft die aktuelle Position einer Bildlaufleistenpositionsmarke ab. Die aktuelle Position ist ein relativer Wert, der vom aktuellen Bildlaufbereich abhängt.

Parameter

Wnd	Fenster, das Standardbildlaufleisten oder Bildlaufleistenelement enthält.
Bar	Eine **sb_xxx Konstanten**.

Rückgabewert

Der Rückgabewert gibt die aktuelle Position der Positionsmarke an.

GetScrollRange (Windows API Prozedur)

Deklaration

```
procedure GetScrollRange(Wnd: HWnd; Bar: Integer; var MinPos, MaxPos:
Integer);
```

Beschreibung

Diese Funktion liefert die aktuellen maximalen und minimalen Bildlaufleistenpositionen für die angegebene Bildlaufleiste.

Parameter

Wnd	Fenster mit Bildlaufleistenelement
Bar	Eine **sb_xxx Konstante**
MinPos	Zeigt auf die Integervariable, die die minimale Position empfangen soll.
MaxPos	Zeigt auf die Integervariable, die die maximale Position empfangen soll.

GetStockObject (Windows API Funktion)

Deklaration

```
function GetStockObject(Index: Integer): THandle;
```

Beschreibung

Diese Funktion ruft ein Handle zu einem der vordefinierten Stifte, Pinsel oder Schriften ab.

Parameter

Index Eine Konstante vom Typ **Stock logischer Objekte**

Rückgabewert

Bezeichnet das gewünschte virtuelle Objekt, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

GetStretchBltMode (Windows API Funktion)

Deklaration

```
function GetStretchBltMode(DC: HDC): Integer;
```

Beschreibung

Diese Funktion ruft den aktuellen Dehnungsmodus für Bitmaps ab.

Parameter

DC Bezeichnet den Gerätekontext.

Rückgabewert

Eine der **StretchBlt-Modus-Konstanten**.

Siehe auch

SetStretchBltMode

GetSubMenu (Windows API Funktion)

Deklaration

```
function GetSubMenu(Menu: HMenu; Pos: Integer): HMenu;
```

Beschreibung

Diese Funktion ruft das Menü-Handle eines Pop-Up-Menüs ab.

Parameter

Menu	Bezeichnet das Menü.
Pos	Bezeichnet die Position des Pop-Up-Menüs im angegebenen Menü.

Rückgabewert

Bezeichnet das gegebene Pop-Up-Menü. Er ist 0, wenn an der angegebenen Position kein Pop-Up-Menü existiert.

GetSysColor (Windows API Funktion)

Deklaration

```
function GetSysColor(Index: Integer): Longint;
```

Beschreibung

Diese Funktion ruft die aktuelle Farbe des Bildschirmelements ab, das vom Parameter Index bezeichnet wird.

Parameter

Index Gibt das Bildschirmelement an, dessen Farbe abgerufen werden soll.

Rückgabewert

Der Rückgabewert gibt einen RGB-Farbwert an, der die Farbe des angegebenen Elements bezeichnet.

Siehe auch

SetSysColors

GetSysModalWindow (Windows API Funktion)

Deklaration

```
function GetSysModalWindow: HWnd;
```

Beschreibung

Diese Funktion gibt das Handle eines systemmodalen Fensters zurück, wenn ein solches vorhanden ist.

Rückgabewert

Bezeichnet das systemmodale Fenster, falls ein solches vorhanden ist. Sonst ist der Rückgabewert 0.

GetSystemDirectory (Windows API Prozedur)

Deklaration

```
procedure GetSystemDirectory(Buffer: PChar; Size: Word);
```

Beschreibung

Diese Funktion ermittelt den Pfadnamen des Windows-Systemunterverzeichnisses. Das Systemunterverzeichnis enthält Dateien wie Bibliotheken, Treiber und Schriftdateien.

Parameter

Buffer	Puffer, der den null-terminierten String empfangen soll
Size	Größe des Puffers (in Bytes) an, mindestens 144

Rückgabewert

Der Rückgabewert gibt die Länge des kopierten Strings, das abschließende Nullzeichen nicht eingeschlossen.

GetSystemMenu (Windows API Funktion)

Deklaration

```
function GetSystemMenu(Wnd: HWnd; Revert: Bool): HMenu;
```

Beschreibung

Diese Funktion erlaubt es der Anwendung, auf das Systemmenü zuzugreifen, um es zu kopieren und /oder zu verändern.

Parameter

Wnd	Fenster, das eine Kopie des Systemmenüs aufnehmen soll.
Revert	Wenn 0: GetSystemMenu gibt ein Handle zu einer Kopie des aktuellen Systemmenüs zurück. Ungleich 0: Handle wird zum Original-Systemmenü zurückgegeben.

Rückgabewert

Bezeichnet das Systemmenü, wenn Revert ungleich 0 ist und das Systemmneü verändert wurde. Wenn Revert ungleich 0 ist und das Systemmenü nicht verändert wurde, ist der Rückgabewert 0.
Wenn Revert 0 ist, dann bezeichnet der Rückgabewert eine Kopie des Systemmenüs.

Siehe auch

AppendMenu
InsertMenu
ModifyMenu

GetSystemMetrics (Windows API Funktion)

Deklaration

```
function GetSystemMetrics(Index: Integer): Integer;
```

Beschreibung

Diese Funktion ruft die Systemmaße ab, d.h. die Breiten und Höhen verschiedener Bildschirmelemente des Windows-Bildschirms. Die Funktion GetSystemMetrics kann auch Flags zurückgeben, die anzeigen, ob die aktuelle Version eine Debugger-Version ist, ob eine Maus angeschlossen ist, oder ob die Bedeutung der linken und rechten Maustaste ausgetauscht worden ist.

Parameter

Index Eine **sm_xxx-**-Konstante.

Rückgabewert

Der Rückgabewert gibt die angeforderten Systemmaßeinheiten an.

GetSystemPaletteEntries (Windows API Funktion)

Deklaration

```
function GetSystemPaletteEntries(DC: HDC; StartIndex, NumEntries: Word;
var PaletteEntries: TPaletteEntry): Word;
```

Beschreibung

Diese Funktion ruft einen Bereich von Paletteneinträgen aus der Systempalette ab.

Parameter

DC — Bezeichnet den Gerätekontext.

StartIndex — Gibt den ersten Eintrag in der Systempalette an, der abgerufen werden soll.

NumEntries — Gibt die Anzahl von Einträgen in der Systempalette an, die abgerufen werden sollen.

PaletteEntries — Zeigt auf ein Array von **TPaletteEntry** -Datenstrukturen.

Rückgabewert

Der Rückgabewert ist die Anzahl von Einträgen, die aus der Systempalette abgerufen wurden. Er ist 0, wenn die Funktion fehlschlägt.

GetSystemPaletteUse (Windows API Funktion)

Deklaration

```
function GetSystemPaletteUse(DC: HDC; B: Word): Word;
```

Beschreibung

Diese Funktion legt fest, ob eine Anwendung auf die ganze Systempalette zugreifen kann.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert bestimmt die aktuelle Verwendung der Systempalette. Er hat einen der **syspal_xxx-** Werte.

Siehe auch

SetSystemPaletteUse

GetTabbedTextExtent (Windows API Funktion)

Deklaration

```
function GetTabbedTextExtent(DC: HDC; Str: PChar; Count, TabPositions:
Integer; var TabStopPositions): Longint;
```

Beschreibung

Diese Funktion berechnet Breite und Höhe der Textzeile, auf die der Parameter Str zeigt. Wenn der String ein oder mehrere Tabulatorzeichen enthält, dann richtet sich die Breite des Strings nach den Tabulatorpositionen, die vom Parameter TabStopPositions vorgegeben sind.

Parameter

DC	Bezeichnet den Gerätekontext.
Str	Zeigt auf einen Textstring.
Count	Gibt die Anzahl von Zeichen im Textstring an.
TabPositions	Gibt die Anzahl von Tabulatorstoppositionen in dem Array an, auf das der Parameter TabStopPositions zeigt.
TabStopPositions	Zeigt auf ein Array von Integerwerten, das die Tabulatorstoppositionen in Pixeln enthält. Die Tabulatorstops müssen in aufsteigender Reihenfolge sortiert werden; Rückwärtstabulatoren sind nicht erlaubt.

Rückgabewert

Der Rückgabewert gibt die Abmessungen des Strings an; die Höhe liegt im höherwertigen, die Breite im niederwertigen Word.

GetTempDrive (Windows API Funktion)

Deklaration

```
function GetTempDrive(DriveLetter: Char): Char;
```

Beschreibung

Diese Funktion nimmt eine Laufwerksbezeichnung oder 0 entgegen und gibt einen Kennbuchstaben zurück, der das optimale Laufwerk für eine temporäre Datei bezeichnet.

Parameter

DriveLetter — Bestimmt eine Laufwerksbezeichnung.

Rückgabewert

Der Rückgabewert zeigt das optimale Laufwerk für temporäre Dateien an.

GetTempFileName (Windows API Funktion)

Deklaration

```
function GetTempFileName(DriveLetter: Char; PrefixString: PChar; Unique:
Word; TempFileName: PChar): Integer;
```

Beschreibung

Diese Funktion erzeugt einen temporären Dateinamen .

Parameter

DriveLetter Gibt das für die temporäre Datei empfohlene **tf_ForceDrive** Laufwerk an. Wenn DriveLetter gleich 0 ist, dann wird das Standardlaufwerk benutzt.

PrefixString Zeigt auf einen null-terminierten String, die als Namenspräfix der temporären Datei verwendet werden soll.

Unique Gibt einen vorzeichenlosen short-Integerwert an.

TempFileName Zeigt auf den Puffer, der den temporären Dateinamen empfangen soll. (mindestens 144 Bytes lang)

Rückgabewert

Der Rückgabewert gibt einen eindeutigen numerischen Wert an, der im temporären Dateinamen benutzt wird.

GetTextAlign (Windows API Funktion)

Deklaration

```
function GetTextAlign(DC: HDC): Word;
```

Beschreibung

Diese Funktion ruft den Status der Textausrichtungsflags ab.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert ist eine Kombination aus einem oder mehreren der **ta_xxx**-Flags.

Siehe auch

TextOut
ExtTextOut

GetTextCharacterExtra (Windows API Funktion)

Deklaration

```
function GetTextCharacterExtra(DC: HDC): Integer;
```

Beschreibung

Diese Funktion ruft den aktuellen Zeichenzwischenraum ab. Der Zeichenzwischenraum definiert den zusätzlichen Raum (in virtuellen Einheiten), den die Funktionen **TextOut** und **ExtTextOut** an jedes Zeichen anhängen, wenn sie eine Zeile schreiben.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt den aktuellen Zeichenzwischenraum an.

Siehe auch

TextOut
ExtTextOut

GetTextColor (Windows API Funktion)

Deklaration

```
function GetTextColor(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ruft die aktuelle Textfarbe ab. Die Textfarbe definiert die Vordergrundfarbe der Zeichen, die mit der Funktion **TextOut** oder **ExtTextOut** dargestellt werden.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt die aktuelle Textfarbe als RGB-Farbwert an.

Siehe auch

TextOut
ExtTextOut

GetTextExtent (Windows API Funktion)

Deklaration

```
function GetTextExtent(DC: HDC; Str: PChar; Count: Integer): Longint;
```

Beschreibung

Diese Funktion berechnet Breite und Höhe der Textzeile, auf die der Parameter Str zeigt. Die Funktion GetTextExtent benutzt die aktuell ausgewählte Schrift, um die Abmessungen des Strings zu berechnen.

Parameter

DC	Bezeichnet den Gerätekontext.
Str	Zeigt auf einen Textstring.
Count	Gibt die Anzahl von Zeichen in dem Textstring an.

Rückgabewert

Der Rückgabewert gibt die Abmessungen des Strings an. Die Höhe ist im niederwertigen, die Breite im höherwertigen Word.

GetTextFace (Windows API Funktion)

Deklaration

```
function GetTextFace(DC: HDC; Count: Integer; Facename: PChar): Integer;
```

Beschreibung

Diese Funktion kopiert die Schriftartbezeichnung der ausgewählten Schrift in einen Puffer, auf den der Parameter Facename zeigt.

Parameter

DC	Bezeichnet den Gerätekontext.
Count	Gibt die Größe des Puffers in Byte an.
Facename	Zeigt auf den Puffer, der die Schriftartbezeichnung empfangen soll.

Rückgabewert

Der Rückgabewert gibt die Anzahl der tatsächlich in den Puffer kopierten Zeichen an. Er ist 0, wenn ein Fehler auftritt.

GetTextMetrics (Windows API Funktion)

Deklaration

```
function GetTextMetrics(DC: HDC; var Metrics: TTextMetric): Bool;
```

Beschreibung

Diese Funktion füllt den Puffer, auf den der Parameter Metrics zeigt, mit den Maßen für die ausgewählte Schrift.

Parameter

DC	Bezeichnet den Gerätekontext.
Metrics	Zeigt auf die **TTextMetric**-Datenstruktur, die Maße empfangen soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

GetThresholdEvent (Windows API Funktion)

Deklaration

```
function GetThresholdEvent: LPInteger;
```

Beschreibung

Diese Funktion ruft ein Flag ab, das ein kürzlich eingegetretenes Schwellenereignis bezeichnet. Ein Schwellenereignis ist jeder Übergang desTon-Warteschlangenwerts von n auf n-1, wobei n der Schwellenwert in Tönen ist.

Rückgabewert

Der Rückgabewert ist ein Zeiger auf ein Schwellenereignis.

GetThresholdStatus (Windows API Funktion)

Deklaration

```
function GetThresholdStatus: Integer;
```

Beschreibung

Diese Funktion ruft den Schwellenereignisstatus für jeden Ton ab. Jedes Bit im Status stellt einen Ton dar. Wenn ein Bit gesetzt ist, dann liegt der Ton-Warteschlangenwert gegenwärtig unter der Schwelle.

Rückgabewert

Der Rückgabewert gibt die Statusflags des aktuellen Schwellenereignisses an.

GetTickCount (Windows API Funktion)

Deklaration

```
function GetTickCount: Longint;
```

Beschreibung

Diese Funktion ermittelt die Anzahl von Millisekunden, die vergangen sind, seit das System gestartet wurde.

Rückgabewert

Der Rückgabewert gibt die Anzahl von Millisekunden an, die vergangen sind, seit das System gestartet wurde.

GetTopWindow (Windows API Funktion)

Deklaration

```
function GetTopWindow(Wnd: HWnd): HWnd;
```

Beschreibung

Diese Funktion sucht nach dem Handle zum obersten untergeordneten Fenster, das zum übergeordneten Fenster gehört.

Parameter

Wnd — Bezeichnet das übergeordnete Fenster.

Rückgabewert

Bezeichnet die ID eines untergeordnetes Fenster. Wenn kein untergeordnetes Fenster existiert, dann ist er 0.

GetUpdateRect (Windows API Funktion)

Deklaration

```
function GetUpdateRect(Wnd: HWnd; var Rect: TRect; Erase: Bool): Bool;
```

Beschreibung

Diese Funktion ruft die Koordinaten des kleinsten Rechtecks ab, das die Aktualisierungsregion des angegebenen Fensters vollständig umschließt.

Parameter

Wnd	Bezeichnet das Fenster, dessen Aktualisierungsregion abgerufen werden soll.
Rect	Zeigt auf die **TRect**-Datenstruktur, die die Client-Koordinaten des umschließenden Rechtecks empfangen soll.
Erase	Gibt an, ob der Hintergrund in der Aktualisierungsregion gelöscht werden soll.

Rückgabewert

Der Rückgabewert gibt den Status der Aktualisierungsregion des angegebenen Fensters an. Er ist ungleich 0, wenn die Aktualisierungsregion nicht leer ist. Andernfalls ist er 0.

Siehe auch

wm_EraseBkgnd

GetUpdateRgn (Windows API Funktion)

Deklaration

```
function GetUpdateRgn(Wnd: HWnd; Rgn: HRgn; Erase: Bool): Integer;
```

Beschreibung

Diese Funktion kopiert die Aktualisierungsregion eines Fensters in die Region, die durch den Parameter Rgn bezeichnet wird. Die Koordinaten dieser Region werden relativ zur oberen linken Ecke des Fensters (in Client-Koordinaten) gebildet.

Parameter

Wnd	Bezeichnet das Fenster, das die zu aktualisierende Region enthält.
Rgn	Bezeichnet die Aktualisierungsregion.
Erase	Gibt an, ob der Fensterhintergrund gelöscht werden soll oder nicht.

Rückgabewert

Der Rückgabewert ist eine der **Bereichsflags**-Konstanten, die den Typ der resultierenden Region anzeigt.

GetVersion (Windows API Funktion)

Deklaration

```
function GetVersion: Word;
```

Beschreibung

Diese Funktion ermittelt die aktuelle Versionsnummer von Windows.

Rückgabewert

Der Rückgabewert gibt die Haupt- und Nebenversionsnummern von Windows an. Das höherwertige Word gibt die Nebenversionsnummer (Revision), das niederwertige die Hauptversionsnummer an.

GetViewportExt (Windows API Funktion)

Deklaration

```
function GetViewportExt(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ruft die x- und y-Abmessungen desGrafikfensters desGerätekontexts ab.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt die x- und y-Abmessungen an (in gerätebezogenen Einheiten). Die y-Abmessung liegt im höherwertigen, die x-Abmessung im niederwertigen Word.

GetViewportOrg (Windows API Funktion)

Deklaration

```
function GetViewportOrg(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ruft die x- und y-Koordinaten des Ursprungs desGrafikfensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert zeigt den Ursprung desGrafikfensters an (in gerätebezogenen Koordinaten). Die y-Koordinate liegt im höherwertigen, die x-Koordinate im niederwertigen Word.

GetWindow (Windows API Funktion)

Deklaration

```
function GetWindow(Wnd: HWnd; Cmd: Word): HWnd;
```

Beschreibung

Diese Funktion sucht nach einem Handle zu einem Fenster aus der Liste des Fenster-Managers. Der Parameter Cmd gibt die Beziehung zwischen dem vom Parameter Wnd bezeichneten Fenster und dem Fenster dessen Handle zurückgegeben wird, an.

Parameter

Wnd	Bezeichnet das ursprüngliche Fenster.
Cmd	Eine **gw_xxx-**-Konstante.

Rückgabewert

Bezeichnet ein Fenster. Er ist 0, wenn die Funktion das Ende der Liste des Fenster-Managers erreicht hat oder wenn der Parameter Cmd ungültig ist.

GetWindowDC (Windows API Funktion)

Deklaration

```
function GetWindowDC(Wnd: HWnd): HDC;
```

Beschreibung

Diese Funktion ruft den Bildschirmkontext für das gesamte Fenster, einschließlich Titelzeile, Menüs und Bildlaufleisten ab.

Parameter

Wnd	Bezeichnet das Fenster, dessen Bildschirmkontext abgerufen werden soll.

Rückgabewert

Bezeichnet den Bildschirmkontext für das angegebene Fenster, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch

ReleaseDC

GetWindowExt (Windows API Funktion)

Deklaration

```
function GetWindowExt(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ruft die x- und y-Abmessungen des Fensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt die x- und y-Abmessungen in virtuellen Einheiten an. Die y-Abmessung liegt im höherwertigen, die x-Abmessung im niederwertigen Word.

GetWindowLong (Windows API Funktion)

Deklaration

```
function GetWindowLong(Wnd: HWnd; Index: Integer): Longint;
```

Beschreibung

Diese Funktion ruft Informationen über das Fenster ab, das vom Parameter Wnd bezeichnet wird.

Parameter

Wnd	Bezeichnet das Fenster.
Index	Gibt den Byte-Offset des abzurufenden Wertes oder einen der **gwl_xxx-Offsets** aus

Rückgabewert

Der Rückgabewert liefert Informationen über das angegebene Fenster.

GetWindowOrg (Windows API Funktion)

Deklaration

```
function GetWindowOrg(DC: HDC): Longint;
```

Beschreibung

Diese Funktion ruft die x- und y-Koordinaten des Ursprungs des Fensters ab, das dem angegebenen Gerätekontext zugeordnet ist.

Parameter

DC	Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert gibt den Ursprung des Fensters an (in virtuellen Koordinaten). Die y-Koordinate liegt im höherwertigen, die x-Koordinate im niederwertigen Word.

GetWindowRect (Windows API Prozedur)

Deklaration

```
procedure GetWindowRect(Wnd: HWnd; var Rect: TRect);
```

Beschreibung

Diese Funktion kopiert die Abmessungen des umgrenzenden Rechtecks des angegebenen Fensters in die Struktur, auf die der Parameter Rect zeigt. Die Abmessungen sind in Bildschirmkoordinaten angegeben.

Parameter

Wnd	Bezeichnet das Fenster.
Rect	Zeigt auf eine **TRect**-Datenstruktur.

GetWindowsDirectory (Windows API Prozedur)

Deklaration

```
procedure GetWindowsDirectory(Buffer: PChar; Size: Word);
```

Beschreibung

Diese Funktion ermittelt den Pfadnamen des Windows-Verzeichnisses. Das Windows-Verzeichnis enthält z. B. Windows-Anwendungen, Initialisierungsdateien und Hilfedateien.

Parameter

Buffer	Zeigt auf den Puffer, der den null-terminierten String empfangen soll, der den Pfadnamen enthält.
Size	Gibt die maximale Größe des Puffers in Bytes an. Dieser Wert sollte auf mindestens 144 gesetzt werden.

Rückgabewert

Der Rückgabewert ist die Länge des Strings, der in Buffer kopiert wurde.

GetWindowTask (Windows API Funktion)

Deklaration

```
function GetWindowTask(Wnd: HWnd): THandle;
```

Beschreibung

Diese Funktion sucht nach dem Handle einer Task, die dem Parameter Wnd zugeordnet ist.

Parameter

Wnd Bezeichnet das Fenster, für das ein Task-Handle abgerufen wird.

Rückgabewert

Bezeichnet die Task, die einem bestimmten Fenster zugeordnet ist.

GetWindowText (Windows API Funktion)

Deklaration

```
functio GetWindowText(Wnd: HWnd; Str: PChar; MaxCount: Integer): Integer;
```

Beschreibung

Diese Funktion kopiert die Kopfzeile des angegebenen Fensters (wenn es eine hat) in den Puffer.

Parameter

Wnd	Bezeichnet das Fenster oder das Steuerelement, dessen Kopfzeile kopiert werden soll.
Str	Zeigt auf den Puffer, der den kopierten String empfangen soll.
MaxCount	Größe von Str.

Rückgabewert

Der Rückgabewert gibt die Länge des kopierten Strings an. Er ist 0, wenn das Fenster keine Kopfzeile hat oder wenn die Kopfzeile leer ist.

GetWindowTextLength (Windows API Funktion)

Deklaration

```
function GetWindowTextLength(Wnd: HWnd): Integer;
```

Beschreibung

Diese Funktion gibt die Länge der Kopfzeile des angegebenen Fensters zurück. Wenn der Parameter Wnd ein Steuerelement bezeichnet, gibt die Funktion die Länge des Textes innerhalb des Steuerelements zurück.

Parameter

Wnd	Bezeichnet das Fenster oder das Steuerelement.

Rückgabewert

Der Rückgabewert gibt die Textlänge an. Er ist 0, wenn kein Text vorhanden ist.

GetWindowWord (Windows API Funktion)

Deklaration

```
function GetWindowWord(Wnd: HWnd; Index: Integer): Word;
```

Beschreibung

Diese Funktion ruft Informationen über das von Wnd bezeichnete Fenster ab.

Parameter

Wnd	Bezeichnet das Fenster.
Index	Gibt den Byte-Offset des abzurufenden Wertes oder eine der **gww_xxx** - Konstanten.

Rückgabewert

Der Rückgabewert gibt Informationen über das bezeichnete Fenster an.

GetWinFlags (Windows API Funktion)

Deklaration

```
function GetWinFlags: Longint;
```

Beschreibung

Diese Funktion gibt einen 32-Bit-Wert zurück, der die Flags enthält, die die Speicherkonfiguration beschreiben, unter der Windows läuft.

Rückgabewert

Der Rückgabewert enthält Flags, die die aktuelle Speicherkonfiguration angeben. Diese **wf_xxx**-Flags können die folgenden Werte haben:

wf_CPU286
wf_CPU386
wf_PMode
wf_SmallFrame
wf_Win286
wf_Win386

GlobalAddAtom (Windows API Funktion)

Deklaration

```
function GlobalAddAtom(Str: PChar): Atom;
```

Beschreibung

Diese Funktion fügt die Zeichenkette, auf die der Parameter Str zeigt, zur Atomtabelle hinzu und erzeugt ein neues globales Atom, das den String eindeutig bezeichnet.

Parameter

Str Zeigt auf die Zeichenkette, die zur Atomtabelle hinzugefügt werden soll.

Rückgabewert

Bezeichnet das neu erzeugte Atom, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

GlobalAlloc (Windows API Funktion)

Deklaration

```
function GlobalAlloc(Flags: Word; Bytes: Longint): THandle;
```

Beschreibung

Diese Funktion reserviert die vom Parameter Bytes vorgegebene Zahl von Bytes auf dem globalen Heap. Der Speicher kann fest oder verschiebbar sein, abhängig von dem durch den Parameter Flags angegebenen Speichertyp.

Parameter

Flags	Gibt ein oder mehrere Flags an, oder eine der **gmem_Globale Speicherflags** -Konstanten.
Bytes	Gibt die Zahl von Bytes an, die belegt werden sollen.

Rückgabewert

Bezeichnet den belegten globalen Speicher, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch

GlobalSize

GlobalCompact (Windows API Funktion)

Deklaration

```
function GlobalCompact(MinFree: Longint): Longint;
```

Beschreibung

Diese Funktion erzeugt die Anzahl freier Bytes im globalen Speicher, die der Parameter MinFree bezeichnet, indem sie den globalen Heap des Systems verdichtet und gegebenenfalls Teile des Heaps verwirft.

Parameter

MinFree Gibt die gewünschte Anzahl von freien Bytes an.

Rückgabewert

Der Rückgabewert gibt die Anzahl von Bytes im größten Block freien Speichers an.

GlobalDeleteAtom (Windows API Funktion)

Deklaration

```
function GlobalDeleteAtom(AnAtom: Atom): Atom;
```

Beschreibung

Diese Funktion verringert den Referenzzähler eines globalen Atoms um eins. Wenn der Referenzzähler des Atoms 0 wird, löscht diese Funktion den zugeordneten String aus der Atomtabelle.

Parameter

AnAtom Bezeichnet das Atom und die Zeichenkette, die gelöscht werden sollen.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist gleich AnAtom, wenn die Funktion fehlschlägt und das Atom nicht gelöscht wurde.

GlobalDiscard (Windows API Funktion)

Deklaration

```
function GlobalDiscard(Mem: THandle): THandle;
```

Beschreibung

Diese Funktion verwirft einen globalen Speicherblock, falls dieser Block nicht gesperrt ist und gelöscht werden kann.

Parameter

Mem Bezeichnet den globalen Speicherblock, der verworfen werden soll.

Rückgabewert

Wenn die Funktion erfolgreich ausgeführt wurde, bezeichnet der Rückgabewert den verworfenen Block, andernfalls ist er 0.

GlobalDosAlloc (Windows API Funktion)

Deklaration

```
function GlobalDosAlloc(Bytes: LongInt): Longint;
```

Beschreibung

Diese Funktion belegt globalen Speicher, auf den von DOS im Real-Modus zugegriffen werden kann. Es wird garantiert, daß der Speicher im ersten MByte des linearen Adreßraums liegt.

Parameter

Bytes Bezeichnet die Anzahl der zu reservierenden Bytes.

Rückgabewert

Der Rückgabewert enthält einen Paragraphsegmentwert in seinem höherwertigen und einen Selektor in seinem niederwertigen Word. Wenn Windows keinen Speicherblock der geforderten Größe reservieren kann, wird 0 zurückgeliefert.

GlobalDosFree (Windows API Funktion)

Deklaration

```
function GlobalDosFree(Selector: Word): Word;
```

Beschreibung

Gibt einen globalen Speicherblock frei, der vorher durch einen Aufruf der Funktion **GlobalDosAlloc** reserviert wurde.

Parameter

Selector Gibt den Speicher an, der freigegeben werden soll.

Rückgabewert

Bei erfolgreicher Ausführung 0, ansonsten ist der Rückgabewert gleich Selector.

GlobalFindAtom (Windows API Funktion)

Deklaration

```
function GlobalFindAtom(Str: PChar): Atom;
```

Beschreibung

Diese Funktion sucht in der Atomtabelle nach der Zeichenkette, auf die der Parameter Str zeigt und ruft das globale Atom ab, das diesem String zugeordnet ist.

Parameter

Str	Zeigt auf die Zeichenkette, nach der gesucht werden soll. Der String muß null-termininert sein.

Rückgabewert

Bezeichnet das globale Atom, das dem gegebenen String zugeordnet ist. Er ist 0, wenn der String nicht in der Tabelle ist.

GlobalFlags (Windows API Funktion)

Deklaration

```
function GlobalFlags(Mem: THandle): Word;
```

Beschreibung

Diese Funktion gibt Informationen über den globalen Speicherblock zurück, den der Parameter Mem bezeichnet.

Parameter

Mem Bezeichnet den globalen Speicherblock.

Rückgabewert

Der Rückgabewert gibt ein Speicherbelegungsflag im höherwertigen Byte an und im niederwertigen Byte den Sperrenzähler des Blocks.

Siehe auch

gmem_xxx-Flags

GlobalFree (Windows API Funktion)

Deklaration

```
function GlobalFree(Mem: THandle): THandle;
```

Beschreibung

Diese Funktion gibt den globalen Speicherblock frei, der vom Parameter Mem bezeichnet wird und macht das Handle des Speicherblocks ungültig.

Parameter

Mem Bezeichnet den globalen Speicherblock der freigegeben werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er gleich Mem.

GlobalGetAtomName (Windows API Funktion)

Deklaration

```
function GlobalGetAtomName(AnAtom: Atom; Buffer: PChar; Size: Integer):
Word;
```

Beschreibung

Diese Funktion ruft eine Kopie der Zeichenkette ab, die dem Parameter AnAtom zugeordnet ist und legt sie in dem Puffer ab, auf den der Parameter Buffer zeigt.

Parameter

AnAtom	Bezeichnet die Zeichenkette, die abgerufen werden soll.
Buffer	Zeigt auf den Puffer, der die Zeichenkette empfangen soll.
Size	Gibt die maximale Größe des Puffers in Bytes an.

Rückgabewert

Der Rückgabewert gibt die tatsächliche Anzahl der in den Puffer kopierten Bytes an. Er ist 0, wenn das angegebene Atom ungültig ist.

GlobalHandle (Windows API Funktion)

Deklaration

```
function GlobalHandle(Mem: Word): Longint;
```

Beschreibung

Diese Funktion ruft das Handle des globalen Speicherobjekts ab, dessen Segmentadresse oder Selektor vom Parameter Mem bezeichnet wird.

Parameter

Mem	Bezeichnet die Segmentadresse oder den Selektor eines globalen Speicherobjekts.

Rückgabewert

Das niederwertige Word des Rückgabewerts gibt das Handle des globalen Speicherobjekts an. Das höherwertige Word des Rückgabewerts gibt die Segmentadresse oder den Selektor des Speicherobjekts an. Der Rückgabewert ist 0, wenn für das Speicherobjekt kein Handle existiert.

GlobalLock (Windows API Funktion)

Deklaration

```
function GlobalLock(Mem: THandle): Pointer;
```

Beschreibung

Diese Funktion ruft einen Zeiger auf den globalen Speicherblock ab, den der Parameter Mem bezeichnet.

Parameter

Mem — Bezeichnet den globalen Speicherblock, der gesperrt werden soll.

Rückgabewert

Der Rückgabewert zeigt auf das erste Speicherbyte im globalen Block, wenn die Funktion erfolgreich ausgeführt wurde. Wenn das Objekt verworfen wurde oder ein Fehler auftritt, dann ist der Rückgabewert **nil**.

GlobalLRUNewest (Windows API Funktion)

Deklaration

```
function GlobalLRUNewest(Mem: THandle): THandle;
```

Beschreibung

Diese Funktion verschiebt das globale Speicherobjekt, das von Mem bezeichnet wird, an die neueste, zuletzt benutzte (LRU) Position im Speicher. Dies reduziert die Wahrscheinlichkeit sehr, daß das Objekt bald verworfen wird, verhindert aber das Verwerfen nicht.

Parameter

Mem	Bezeichnet das globale Speicherobjekt, das verschoben werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn der Parameter Mem kein gültiges Handle angibt.

GlobalLRUOldest (Windows API Funktion)

Deklaration

```
function GlobalLRUOldest(Mem: THandle): THandle;
```

Beschreibung

Diese Routine verschiebt das globale Speicherobjekt, das von Mem bezeichnet wird, an die älteste ganz zuletzt benutzte (LRU) Position im Speicher und macht es damit zum nächsten Anwärter zum Verwerfen.

Parameter

Mem Bezeichnet das globale Speicherobjekt, das verschoben werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn der Parameter Mem kein gültiges Handle angibt.

GlobalNotify (Windows API Prozedur)

Deklaration

```
procedure GlobalNotify(NotifyProc: TFarProc);
```

Beschreibung

Diese Funktion installiert eine Benachrichtigungsprozedur für die aktuelle Task. Windows ruft die Benachrichtigungsprozedur immer dann auf, wenn ein globaler Speicherblock, der mit dem Flag gmem_Notify reserviert wurde, kurz davor steht, verworfen zu werden.

Parameter

NotifyProc Ist die Adresse der Prozedurinstanz der aktuellen Benachrichtigungsprozedur der Task.

GlobalReAlloc (Windows API Funktion)

Deklaration

```
function GlobalReAlloc(Mem: THandle; Bytes: Longint; Flags: Word):
THandle;
```

Beschreibung

Diese Funktion reserviert den globalen Speicherblock, der vom Parameter Mem bezeichnet wird, indem sie seine Größe auf die Zahl von Bytes verringert oder erhöht, die der Parameter Bytes vorgibt.

Parameter

Mem	Bezeichnet den Speicherblock, der neu reserviert werden soll.
Bytes	Bezeichnet die neue Größe des Speicherblocks.
Flags	Eineder folgenden **gmem_Global memory flags**: gmem_Discardable gmem_Notify gmem_Moveable gmem_NoCompact gmem_NoDiscard gmem_ZeroInit

Rückgabewert

Bezeichnet den neu reservierten globalen Speicher, wenn die Funktion erfolgreich ausgeführt wurde. Der Rückgabewert ist 0, wenn der Block nicht neu zugewiesen werden konnte.

GlobalSize (Windows API Funktion)

Deklaration

```
function GlobalSize(Mem: THandle): Longint;
```

Beschreibung

Diese Funktion ruft die aktuelle Größe des globalen Speicherblocks in Byte ab, den der Parameter Mem bezeichnet.

Parameter

Mem	Bezeichnet den globalen Speicherblock.

Rückgabewert

Der Rückgabewert gibt die tatsächliche Größe (in Bytes) des angegebenen Speicherblocks an. Er ist 0, wenn das gegebene Handle nicht gültig ist oder wenn das Objekt verworfen wurde.

GlobalUnlock (Windows API Funktion)

Deklaration

```
function GlobalUnlock(Mem: THandle): Bool;
```

Beschreibung

Diese Funktion hebt die durch **GlobalLock** vorgenommene Sperre des globalen Speicherblocks auf, den der Parameter Mem bezeichnet.
Der Block kann verschoben oder verworfen werden, wenn der Sperrenzähler auf 0 verringert wurde.

Parameter

Mem	Bezeichnet den globalen Speicherblock, der entsperrt werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn der Sperrenzähler des Blocks bis auf 0 verringert wurde. Andernfalls ist der Rückgabewert ungleich 0.

GlobalUnWire (Windows API Funktion)

Deklaration

```
function GlobalUnWire(Mem: THandle): Bool;
```

Beschreibung

Diese Funktion hebt die Sperre eines Speichersegment auf, das von der Funktion GlobalWire gesperrt wurde und verringert den Sperrenzähler um 1.

Parameter

Mem Bezeichnet das Segment, dessen Sperre aufgehoben werden soll.

Rückgabewert

Er ist ungleich 0, wenn die Sperre des Speichersegments aufgehoben wurde, das heißt, wenn der Sperrenzähler auf 0 verringert wurde. Andernfalls ist er 0.

GlobalWire (Windows API Funktion)

Deklaration

```
function GlobalWire(Mem: THandle): PChar;
```

Beschreibung

Diese Funktion verschiebt ein Segment in den unteren Speicherbereich und sperrt ihn.

Parameter

Mem — Bezeichnet das Segment, das verschoben und gesperrt werden soll.

Rückgabewert

Der Rückgabewert zeigt auf die neue Segmentposition. Er ist **nil**, wenn die Funktion fehlgeschlagen ist.

Siehe auch

GlobalUnWire

GrayString (Windows API Funktion)

Deklaration

```
function GrayString(DC: HDC; Brush: HBrush; OutputFunc: TFarProc; Data:
Longint; Count, X, Y, Width, Height: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet an der angegebenen Stelle grauen nicht aktivierbaren Text. Die Funktion GrayString zeichnet den Text, indem sie ihn in ein Speicher-Bitmap schreibt, das Bitmap grau färbt und es dann auf den Bildschirm kopiert.

Parameter

DC	Bezeichnet den Gerätekontext.
Brush	Bezeichnet den Pinsel für das Graufärben
OutputFunc	Ist die Adresse der Prozedurinstanz oder **nil** für **TextOut**
Data	Wenn der Parameter OutputFunc 0 ist, dann muß Data ein Zeiger auf den String sein, der ausgegeben werden soll.
Count	Gibt die Anzahl der auszugebenden Zeichen an. Wenn der Parameter Count 0 ist, dann berechnet GrayString die Länge des Strings
X, Y	Startposition des Rechtecks an, das den String einschließt.
Width	Gibt die Breite des Rechtecks in virtuellen Einheiten an, das den String einschließt. Wenn der Parameter Width 0 ist, dann berechnet GrayString die Breite des Bereichs, vorausgesetzt, daß Data ein Zeiger auf den String ist.
Height	Gibt die Höhe (in virtuellen Einheiten) des Rechtecks an, das den String einschließt.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der String gezeichnet wird. Ein Rückgabewert von 0 bedeutet, daß entweder die Funktion TextOut oder die von der Anwendung definierte Funktion 0 zurückgegeben hat oder daß nicht genügend Speicherplatz vorhanden war, um ein Speicherbitmap zur Darstellung des nichtaktivierbaren Zustands zu erstellen.

Siehe auch

Color_Graytext
GetSysColor
mm_Text
SetTextColor

HiByte (Windows API Funktion)

Deklaration

```
function HiByte(A: Word): Byte;
  inline(
    $5A/        { POP AX    }
    $8A/$C4/    { MOV AL,AH }
    $32/$E4);   { XOR AH,AH }
```

Beschreibung

Liefert das höherwertige Byte des Parameters A.

Parameter

A _____ Ein 16-Bit-Integer, der konventiert werden soll.

Rückgabewert

Das höherwertige Byte

HiWord (Windows API Funktion)

Deklaration

```
function HiWord(AnInteger: Longint): Word;
```

Beschreibung

Liefert das höherwertige Word des mit AnInteger gegebenen 32-Bit Wertes.

Parameter

AnInteger Der 32-bit Integerwer

Rückgabewert

Das höherwertige Word

HideCaret (Windows API Prozedur)

Deklaration

```
procedure HideCaret(Wnd: HWnd);
```

Beschreibung

Diese Funktion blendet das Caret aus, indem sie es vom Bildschirm löscht.

Parameter

Wnd	Bezeichnet das Fenster, das das Caret besitzt oder ist 0, um indirekt das Fenster in der aktuellen Task anzugeben, welches das Caret besitzt.

Siehe auch

ShowCaret

HiliteMenuItem (Windows API Funktion)

Deklaration

```
function HiliteMenuItem(Wnd: HWnd; Menu: HMenu; IDHilite, Hilite: Word):
Bool;
```

Beschreibung

Diese Funktion hebt eine Hauptmenüoption (Menüleiste) hervor oder setzt es auf den Normalzustand zurück.

Parameter

Wnd	Bezeichnet das Fenster,welches das Menü enthält.
Menu	Bezeichnet das Hauptmenü, das die hervorzuhebende Option enthält.
IDHilite	Gibt den Integerbezeichner der Menüoption oder deren Offset im Menü in Abhängigkeit vom Parameter wHilite an.
Hilite	Kombination der **mf_xxx-Flags**, mf_ByCommand oder mf_ByPosition mit mf_Hilite oder mf_Unhilite

Rückgabewert

Er ist ungleich 0, wenn das Element in den angegebenen Zustand versetzt wurde. Andernfalls ist er 0 und damit False.

InflateRect (Windows API Funktion)

Deklaration

```
procedure InflateRect(var Rect: TRect; X, Y: Integer);
```

Beschreibung

Diese Funktion vergrößert oder verkleinert Höhe und Breite des angegebenen Rechtecks. Die Funktion InflateRect fügt X Einheiten an die linke und die rechte Seite des Rechtecks sowie Y Einheiten an die Ober- und Unterseite an.

Parameter

Rect	Zeigt auf die **TRect**-Datenstruktur, die verändert werden soll.
X	Gibt den Betrag an, um den die Breite des Rechtecks vergrößert oder verkleinert werden soll.
Y	Gibt den Betrag an, um den die Höhe des Rechtecks vergrößert oder verkleinert werden soll.

InitAtomTable (Windows API Funktion)

Deklaration

```
function InitAtomTable(Size: Integer): Bool;
```

Beschreibung

Diese Funktion initialisiert eine Atom-Hash-Tabelle und setzt ihre Größe entsprechend der Angabe des Parameters Size (Vorgabe: 37).

Parameter

Size	Gibt die Größe der Atom-Hash-Tabelle (in Tabelleneinträgen) an. Dieser Wert sollte eine Primzahl sein.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

InSendMessage (Windows API Funktion)

Deklaration

```
function InSendMessage: Bool;
```

Beschreibung

Diese Funktion gibt an, ob die aktuelle Fensterfunktion eine Botschaft bearbeitet, die ihr durch Aufruf der Funktion **SendMessage** übergeben wurde.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Fensterfunktion eine Botschaft bearbeitet, die ihr durch SendMessage gesendet wurde. Andernfalls ist er 0.

InsertMenu (Windows API Funktion)

Deklaration

```
function InsertMenu(Menu:HMenu; Position, Flags, IDNewItem: Word; NewItem:
PChar): Bool;
```

Beschreibung

Diese Funktion fügt an der Position, die der Parameter Position vorgibt, eine neue Menüoption ein, deren Status imParameter Flags übergeben werden.

Parameter

Menu	Bezeichnet das Menü, das verändert werden soll.
Position	Gibt die Menüoption an, vor der die neue Menüoption eingefügt werden soll.
Flags	Eine der **mf_Menüflags**, mf_ByCommand oder mf_ByPosition , mit einer der folgenden Werte: mf_Bitmap mf_String mf_OwnerDraw mf_Separator mf_Popup mf_MenuBarBreak mf_MenuBreak mf_Checked mf_Unchecked
IDNewItem	Neue Menüoptions-ID
NewItem	Neuer Menüoptionsinhalt.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch

DrawMenuBar
wm_DrawItem
wm_MeasureItem

IntersectClipRect (Windows API Funktion)

Deklaration

```
function IntersectClipRect(DC: HDC; X1, Y1, X2, Y2: Integer): Integer;
```

Beschreibung

Diese Funktion erzeugt eine neue Clipping-Region, indem sie den Überschneidungsbereich der aktuellen Region und des angegebenenRechtecks bildet.

Parameter

DC	Bezeichnet den Gerätekontext.
X1, Y1	Koordinaten der oberen linken Ecke des Rechtecks
X2, Y2	Koordinaten der unteren rechten Ecke des Rechtecks

Rückgabewert

Der Rückgabewert gibt den Typ der neuen Clipping-Region an. Er kann einer der **Bereichsflags** sein.

IntersectRect (Windows API Funktion)

Deklaration

```
function IntersectRect(var DestRect, Src1Rect, Src2Rect: TRect): Integer;
```

Beschreibung

Diese Funktion erzeugt den Überschneidungsbereich von zwei existierenden Rechtecken.

Parameter

DestRect	Zeigt auf die **TRect**-Datenstruktur, die den Überschneidungsbereich empfangen soll.
Src1Rect	Zeigt auf eine TRect-Datenstruktur, die ein Quellrechteck enthält.
Src2Rect	Zeigt auf eine TRect-Datenstruktur, die ein Quellrechteck enthält.

Rückgabewert

Der Rückgabewert gibt den Überschneidungsbereich von zwei Rechtecken an. Er ist 0, wenn der Überschneidungsbereich leer ist, ansonsten ist er ungleich 0.

InvalidateRect (Windows API Prozedur)

Deklaration

```
procedure InvalidateRect(Wnd: HWnd; Rect: LPRect; Erase: Bool);
```

Beschreibung

Diese Funktion macht den Client-Bereich innerhalb des gegebenen Rechtecks ungültig, indem dieses Rechteck an die Aktualisierungsregion von Windows angefügt wird.

Parameter

Wnd	Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die das Rechteck (in Client-Koordinaten) enthält, das an die Aktualisierungsregion angefügt werden soll.
Erase	Gibt an, ob der Hintergrund innerhalb der Aktualisierungsregion von **BeginPaint** gelöscht werden soll.

Siehe auch

ValidateRect
ValidateRgn
wm_Paint

InvalidateRgn (Windows API Prozedur)

Deklaration

```
procedure InvalidateRgn(Wnd: HWnd; Rgn: HRgn; Erase: Bool);
```

Beschreibung

Diese Funktion macht den Client-Bereich innerhalb der angegebenen Region ungültig, indem sie ihn zur Aktualisierungsregion des gegebenen Fensters hinzufügt.

Parameter

Wnd	Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.
Rgn	Bezeichnet die Region, die an die Aktualisierungsregion angefügt werden soll. Es wird davon ausgegangen, daß die Region in Client-Koordinaten angegeben ist.
Erase	Gibt an, ob der Hintergrund innerhalb der Aktualisierungsregion von **BeginPaint** gelöscht werden soll.

Siehe auch

ValidateRect
ValidateRgn
wm_Paintwm_Paint
BeginPaint

InvertRect (Windows API Prozedur)

Deklaration

```
procedure InvertRect(DC: HDC; var Rect: TRect);
```

Beschreibung

Diese Funktion invertiert die Inhalte des gegebenen Rechtecks.

Parameter

DC	Bezeichnet den Gerätekontext.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die die virtuellen Koordinaten des zu invertierenden Rechtecks enthält.

InvertRgn (Windows API Funktion)

Deklaration

```
function InvertRgn(DC: HDC; Rgn: HRgn): Bool;
```

Beschreibung

Diese Funktion invertiert die Farben in der Region, die der Parameter Rgn bezeichnet.

Parameter

DC	Bezeichnet den Gerätekontext.
Rgn	Bezeichnet die Region, die ausgefüllt werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

IsCharAlpha (Windows API Funktion)

Deklaration

```
function IsCharAlpha(AChar: Char): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob ein Zeichen ein alphabetisches Zeichen ist. Diese Feststellung wird vom Sprachtreiber getroffen, der vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde.

Parameter

AChar — Gibt das Zeichen an, das überprüft werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Zeichen alphabetisch ist. Andernfalls ist er 0.

IsCharAlphaNumeric (Windows API Funktion)

Deklaration

```
function IsCharAlphaNumeric(AChar: Char): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob ein Zeichen ein alphabetisches oder numerisches Zeichen ist.

Parameter

AChar Gibt das Zeichen an, das überprüft werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Zeichen alphanumerisch ist. Andernfalls ist er 0.

IsCharLower (Windows API Funktion)

Deklaration

```
function IsCharLower(AChar: Char): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob das Zeichen in Kleinschreibung vorliegt. Diese Feststellung wird vom Sprachtreiber getroffen, der vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde.

Parameter

AChar — Gibt das Zeichen an, das überprüft werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Zeichen in Kleinschreibung vorliegt. Andernfalls ist er 0.

IsCharUpper (Windows API Funktion)

Deklaration

```
function IsCharUpper(AChar: Char): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob das Zeichen in Großschreibung vorliegt.

Parameter

AChar Gibt das Zeichen an, das überprüft werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Zeichen in Großschreibung vorliegt. Andernfalls ist er 0.

IsChild (Windows API Funktion)

Deklaration

```
function IsChild(Parent, Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion zeigt an, ob das vom Parameter Wnd angegebene Fenster ein untergeordnetes Fenster oder ein anderer direkter Abkömmling des Fensters ist, das der Parameter WndParent bezeichnet.

Parameter

WndParent	Bezeichnet ein Fenster.
Wnd	Bezeichnet das Fenster, das überprüft werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das vom Parameter Wnd bezeichnete Fenster ein untergeordnetes Fenster des Fensters ist, das vom Parameter WndParent bezeichnet ist. Andernfalls ist er 0.

IsClipboardFormatAvailable (Windows API Funktion)

Deklaration

```
function IsClipboardFormatAvailable(Format: Word): Bool;
```

Beschreibung

Diese Funktion gibt an, ob Daten eines bestimmten Typs in der Zwischenablage existieren.

Parameter

Format	Gibt ein registriertes Zwischenablageformat an, ist eine der **cf_xxx** -Konstanten

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn Daten, im bezeichneten Format vorliegen. Andernfalls ist er 0.

Siehe auch

SetClipboardData

IsDialogMessage (Windows API Funktion)

Deklaration

```
function IsDialogMessage(Dlg: HWnd; var Msg: TMsg): Bool;
```

Beschreibung

Diese Funktion ermittelt, ob die angegebene Botschaft für das nicht-modale Dialogfenster, das durch den Parameter Dlg bezeichnet ist, bestimmt ist und verarbeitet die Botschaft automatisch, wenn dies der Fall ist.

Parameter

Dlg	Bezeichnet das Dialogfenster.
Msg	Zeigt auf eine **TMsg**-Datenstruktur, die die Botschaft enthält, die geprüft werden soll.

Rückgabewert

Der Rückgabewert gibt an, ob die angegebene Botschaft (dazu sollten weder **TranslateMessage** noch **DispatchMessage** aufgerufen werden)verarbeitet wurde. Er ist ungleich 0, wenn die Botschaft verarbeitet worden ist. Andernfalls ist er 0.

IsDlgButtonChecked (Windows API Funktion)

Deklaration

```
function IsDlgButtonChecked(Dlg: HWnd; IDButton: Integer): Word;
```

Beschreibung

Diese Funktion stellt fest, ob ein Schalter eine Auswahlmarkierung hat.

Parameter

Dlg	Bezeichnet das Dialogfenster, das den Schalter enthält.
IDButton	Gibt den Integerbezeichner des Schalters an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der angegebene Schalter markiert ist. Andernfalls ist er 0. Bei dreistufigen Schaltern ist der Rückgabewert 2, wenn der Schalter nichtaktivierbar ist. Er ist 1, wenn der Schalter markiert ist, andernfalls ist er 0.

IsIconic (Windows API Funktion)

Deklaration

```
function IsIconic(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion gibt an, ob ein Fenster zum Symbol verkleinert ist.

Parameter

Wnd — Bezeichnet das Fenster.

Rückgabewert

Der Rückgabewert gibt an, ob das Fenster verkleinert ist. In diesem Fall ist er ungleich 0, andernfalls 0.

IsRectEmpty (Windows API Funktion)

Deklaration

```
function IsRectEmpty(var Rect: TRect): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob das bezeichnete Rechteck leer ist oder nicht. Ein Rechteck ist leer, wenn Höhe und/oder Breite 0 sind.

Parameter

Rect — Zeigt auf eine **TRect**-Datenstruktur, die das angegebene Rechteck enthält.

Rückgabewert

Der Rückgabewert gibt an, ob das gegebene Rechteck leer ist oder nicht. Er ist ungleich 0, wenn das Rechteck leer ist. Er ist 0, wenn das Rechteck nicht leer ist.

IsWindow (Windows API Funktion)

Deklaration

```
function IsWindow(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob das Fenster, das durch den Parameter Wnd bezeichnet ist, ein gültiges, existierendes Fenster ist.

Parameter

Wnd	Bezeichnet das Fenster.

Rückgabewert

Der Rückgabewert gibt an, ob das angegebene Fenster gültig ist. Er ist ungleich 0, wenn Wnd ein gültiges Fenster ist. Andernfalls ist er 0.

IsWindowEnabled (Windows API Funktion)

Deklaration

```
function IsWindowEnabled(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion gibt an, ob das angegebene Fenster zur Entgegennahme von Maus- und Tastatureingaben befähigt ist.

Parameter

Wnd	Bezeichnet das Fenster.

Rückgabewert

Der Rückgabewert gibt an, ob das angegebene Fenster aktiviert ist. Er ist ungleich 0, wenn das Fenster aktiviert ist. Andernfalls ist er 0.

IsWindowVisible (Windows API Funktion)

Deklaration

```
function IsWindowVisible(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion gibt jedesmal einen Wert ungleich 0 zurück, wenn die Funktion ein Fenster mit der Funktion **ShowWindow** sichtbar gemacht hat. (Auch, wenn das bezeichnete Fenster vollständig von einem anderen untergeordneten oder Pop-Up-Fenster verdeckt ist, ist der Rückgabewert ungleich 0).

Parameter

Wnd	Bezeichnet das Fenster.

Rückgabewert

Der Rückgabewert gibt an, ob ein gegebenes Fenster auf dem Bildschirm vorhanden ist. Andernfalls ist er 0.

IsZoomed (Windows API Funktion)

Deklaration

```
function IsZoomed(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob ein Fenster vergrößert ist oder nicht.

Parameter

Wnd — Bezeichnet das Fenster.

Rückgabewert

Der Rückgabewert gibt an, ob das gegebene Fenster vergrößert ist oder nicht. Er ist ungleich 0, wenn das Fenster vergrößert ist. Andernfalls ist er 0.

KillTimer (Windows API Funktion)

Deklaration

```
function KillTimer(Wnd: HWnd; IDEvent: Integer): Bool;
```

Beschreibung

Diese Funktion eliminiert das Timer-Ereignis, das durch die Parameter Wnd und nIDEvent bezeichnet wird. Alle anstehenden **wm_Timer**-Botschaften, die mit dem Timer verbunden sind, werden aus der Botschaftsschlange entfernt.

Parameter

Wnd	Bezeichnet das Fenster, das mit dem angegebenen Timer-Ereignis verbunden ist.
IDEvent	Gibt das Timer-Ereignis an, das eliminiert werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Ereignis eliminiert wurde. Er ist 0, wenn die Funktion das angegebene Timer-Ereignis nicht finden konnte.

Siehe auch

SetTimer

_lclose (Windows API Funktion)

Deklaration

```
function _lclose(FileHandle: Integer): Integer;
```

Beschreibung

Diese Funktion schließt die durch den Parameter File angegebene Datei.

Parameter

File	Gibt das MS-DOS Datei-Handle der Datei an, die geschlossen werden soll.

Rückgabewert

Der Rückgabewert zeigt an, ob die Funktion die Datei erfolgreich geschlossen hat. Er ist 0, wenn die Funktion die Datei geschlossen hat oder -1, wenn ein Fehler aufgetreten ist.

_lcreat (Windows API Funktion)

Deklaration

```
function _lcreat(PathName: PChar; Attribute: Integer): Integer;
```

Beschreibung

Diese Funktion öffnet eine Datei mit dem Namen, der durch den Parameter PathName angegeben wird.

Parameter

PathName	DOS-Dateiname der zu öffnenden Datei
Attribute	Gibt das Dateiattribut an. Der Parameter muß einen der folgenden Werte annehmen: 0 Lesen oder Schreiben 1 Schreibgeschützt; 2 Unsichtbar; 3 System;

Rückgabewert

Der Rückgabewert gibt ein MS-DOS-Datei-Handle an, wenn die Funktion erfolgreich ist. Andernfalls ist der Rückgabewert -1.

LimitEmsPages (Windows API Prozedur)

Deklaration

```
procedure LimitEmsPages(Kbytes: Longint);
```

Beschreibung

Diese Funktion begrenzt den Umfang des Expanded Memorys, den Windows einer Anwendung zuweisen wird. Er begrenzt nicht den Umfang des Expanded Memorys, den die Anwendung durch direkten Aufruf des INT 67H verwenden kann.

Parameter

Kbytes	Gibt die Anzahl KByte des Expanded Memory an, auf die die Anwendung zugreifen kann.

LineDDA (Windows API Prozedur)

Deklaration

```
procedure LineDDA(X1, Y1, X2, Y2: Integer; LineFunc: TFarProc; Data:
Pointer);
```

Beschreibung

Diese Funktion berechnet alle aufeinander folgenden Punkte einer Linie vom Anfangspunkt, der durch die Parameter X1 und Y1 festgelegt wird, bis zum Endpunkt mit den Koordinaten X2 und Y2. Der Endpunkt ist nicht als Teil der Linie eingeschlossen. Für jeden Punkt der Linie ruft die Funktion LineDDA die anwendungsunterstützte Funktion auf, auf die der Parameter LineFunc zeigt .und übergibt dieser Funktion die Koordinaten des aktuellen Punktes und den Parameter lData.

Parameter

X1, Y1	Gibt die virtuellen Koordinaten des ersten Punktes an.
X2, Y2	Gibt die virtuellen Koordinaten des Endpunktes an.
LineFunc	Ist die Adresse der anwendungsunterstützten Funktion.
Data	Zeigt auf die anwendungsunterstützten Daten.

LineTo (Windows API Funktion)

Deklaration

```
function LineTo(DC: HDC; X, Y: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet eine Linie von der aktuellen Position bis zum Punkt (aber nicht einschließlich), der durch die Parameter X und Y angegeben wird. Die Linie wird mit dem ausgewählten Stift gezeichnet.

Parameter

DC	Bezeichnet den Gerätekontext.
X ,Y	Gibt die virtuellen Koordinaten des Endpunkts der Linie an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Linie gezeichnet wurde. Andernfalls ist er 0.

_llseek (Windows API Funktion)

Deklaration

```
function _llseek(FileHandle: Integer; Offset: Longint; Origin: Integer):
Longint;
```

Beschreibung

Diese Funktion positioniert den Zeiger in einer vorher geöffneten Datei neu. Der Parameter Origin legt die Startposition in der Datei fest und Offset gibt an, wie weit (in Bytes) die Funktion den Zeiger verschieben soll.

Parameter

File	Gibt das MS-DOS-Datei-Handle der Datei an.
Offset	Gibt die Anzahl Bytes an, um die der Zeiger verschoben wird
Origin	Gibt die Startposition und Richtung des Zeigers an. Der Parameter muß einen der folgenden Werte annehmen: 0 Bewegt den Zeiger lOffset Bytes, ausgehend vom Anfang der Datei. 1 Bewegt den Zeiger lOffset Bytes, ausgehend von der aktuellen Position in der Datei. 2 Bewegt den Zeiger lOffset Bytes, ausgehend vom Ende der Datei.

Rückgabewert

Der Rückgabewert gibt den neuen Offset des Zeigers (in Bytes), ausgehend vom Anfang der Datei, an. Der Rückgabewert ist -1, wenn ein Fehler aufgetreten ist.

LoadAccelerators (Windows API Funktion)

Deklaration

```
function LoadAccelerators(Instance: THandle; TableName: PChar): THandle;
```

Beschreibung

Diese Funktion lädt die Akzeleratortabelle, die durch den Parameter TableName benannt wird, aus der ausführbaren Datei, die dem Modul, das durch den Parameter Instance angegeben wird, zugeordnet ist.

Parameter

Instance Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Akzeleratortabelle enthält.

TableName Zeigt auf einen null-terminierten String, die den Namen der Akzeleratortabelle enthält.

Rückgabewert

Bezeichnet die geladene Akzeleratortabelle, wenn die Funktion erfolgreich ist. Sonst ist er 0.

LoadBitmap (Windows API Funktion)

Deklaration

```
function LoadBitmap(Instance: THandle; BitmapName: PChar): HBitmap;
```

Beschreibung

Diese Funktion lädt die Bitmap-Ressource, die durch den Parameter BitmapName benannt wird, aus der ausführbaren Datei, die dem Modul zugeordnet ist, daß durch den Parameter Instance angegeben wird.

Parameter

Instance Bezeichnet die Instanz des Moduls, dessen ausführbare Datei das Bitmap enthält.

BitmapName Zeigt auf einen null-terminierten String, der den Namen des Bitmap enthält, eine Integer-ID eines Bitmaps oder enthält eine **obm_xxx** -Konstante, die ein vordefiniertes Bitmap kennzeichnet.

Rückgabewert

Bezeichnet das angegebene Bitmap. Er ist 0, wenn dieses Bitmap nicht extistiert.

LoadCursor (Windows API Funktion)

Deklaration

```
function LoadCursor(Instance: THandle; CursorName: PChar): HCursor;
```

Beschreibung

Diese Funktion lädt die Zeiger-Ressource, die durch den Parameter CursorName benannt wird.

Parameter

Instance — Bezeichnet die Instanz des Moduls, dessen ausführbare Datei den Zeiger enthält.

CursorName — Zeigt auf einen null-terminierten String, die den Namen der Zeiger-Ressource enthält. (**idc_ Standardcursor-IDs** -Konstante)

Rückgabewert

Bezeichnet den neugeladenen Zeiger, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

LockData (Windows API function)

Deklaration

```
function LockData(Dummy: Integer): THandle;
```

Beschreibung

Sperrt das aktuelle verschiebbare Datensegment im Speicher.

Parameter

Dummy	Wird nicht verwendet, wird auf 0

Return Value

Bezeichner des gesperrten Datensegments, andernfalls gleich 0.

Siehe auch

UnlockData

LoadIcon (Windows API Funktion)

Deklaration

```
function LoadIcon(Instance: THandle; IconName: PChar): HIcon;
```

Beschreibung

Diese Funktion lädt die Symbol-Ressource, die durch den Parameter IconName benannt wird.

Parameter

Instance	Bezeichnet die Instanz des Moduls, dessen ausführbare Datei das Symbol enthält.
IconName	Zeigt auf einen null-terminierten String, die den Namen der Symbol-Ressource enthält., oder auf eine der **idi_ Standardsymbol-IDs**

Rückgabewert

Bezeichner eine Symbol-Ressource, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

LoadLibrary (Windows API Funktion)

Deklaration

```
function LoadLibrary(LibFileName: PChar): THandle;
```

Beschreibung

Diese Funktion lädt das Bibliotheksmodul, daß in der angegebenen Datei enthalten ist und liefert ein Handle zur Instanz des geladenen Moduls.

Parameter

LibFileName — Zeigt auf einen null-terminierten String, die den Namen der Bibliotheksdatei enthält.

Rückgabewert

Bezeichnet die Instanz des geladenen Bibliothekmoduls, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls hat er einen Wert kleiner als 32, der den Fehler bezeichnet.

LoadMenu (Windows API Funktion)

Deklaration

```
function LoadMenu(Instance: THandle; MenuName: PChar): HMenu;
```

Beschreibung

Diese Funktion lädt die Menü-Ressource, die durch den Parameter MenuName benannt wird, aus der ausführbaren Datei, die dem Modul zugeordnet ist, das durch den Parameter Instance angegeben wird.

Parameter

Instance Bezeichnet die Instanz des Moduls, dessen ausführbare Datei das Menü enthält.

MenuName Zeigt auf einen null-terminierten String, die den Namen der Menü-Ressource enthält.

Rückgabewert

Bezeichnet eine Menü-Ressource, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

LoadMenuIndirect (Windows API Funktion)

Deklaration

```
function LoadMenuIndirect(var MenuTemplate): HMenu;
```

Beschreibung

Diese Funktion lädt das Menü in den Speicher, das durch den Parameter MenuTemplate benannt wird.

Parameter

MenuTemplate Zeigt auf eine Menüvorlage (ein Array einer oder mehrerer **TMenuItemTemplate**-Strukturen ist).

Rückgabewert

Bezeichnet das Menü, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

LoadModule (Windows API Funktion)

Deklaration

```
function LoadModule(ModuleName: PChar; ParameterBlock: Pointer): THandle;
```

Beschreibung

Diese Funktion lädt ein Windows-Programm und führt es aus oder erzeugt eine neue Instanz eines existierenden Windows-Programms.

Parameter

ModuleName Zeigt auf einen mit null-terminierten String, der den Dateinamen der Anwendung enthält, die gestartet werden soll. Wenn der String ModuleName keinen

ParameterBlock Zeigt auf eine Datenstruktur, die aus vier Feldern besteht, die einen Parameterblock definieren:

EnvSeg Gibt in einem Word die Segmentadresse der Umgebung an, unter der das Modul laufen soll; 0 zeigt an, daß die Windows- Umgebung kopiert werden soll.

CmdLine Zeigt auf einen null-terminierten String, der eine korrekt gebildete Kommandozeile enthält. Dieser String darf ein Länge von 120 Bytes nicht überschreiten.

CmdShow Zeigt auf eine Datenstruktur, die zwei Werte in Word-Länge enthält. Der erste Wert muß immer auf 2 gesetzt werden. Der zweite Wert wird dem Feld CmdShow der Funktion **ShowWindow** gleichgesetzt.

Reserved Ist reserviert und muß 0 sein.

Rückgabewert

Wie bei **LoadLibrary**

Siehe auch

WinExec

LoadResource (Windows API Funktion)

Deklaration

```
function LoadResource(Instance, ResInfo: THandle): THandle;
```

Beschreibung

Diese Funktion lädt eine Ressource, die durch den Parameter ResInfo bezeichnet wird. Die Funktion lädt die Ressource nur in den Speicher, wenn sie nicht schon vorher geladen wurde. Andernfalls liefert sie ein Handle zur bestehenden Ressource.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.
ResInfo	Bezeichnet die gewünschte Ressource. Voraussetzung ist, daß dieses Handle durch die Funktion **FindResource** erzeugt wurde.

Rückgabewert

Bezeichnet den globalen Speicherblock, der die Daten empfängt, die der Ressource zugeordnet sind. Er ist 0, wenn die Ressource nicht existiert.

Siehe auch

LockResource

LoadString (Windows API Funktion)

Deklaration

```
function LoadString(Instance: THandle; ID: Word; Buffer: PChar; BufferMax:
Integer): Integer;
```

Beschreibung

Diese Funktion lädt eine String-Ressource, die durch den Parameter ID bezeichnet wird. Die Funktion kopiert den String in den Puffer, auf den der Parameter Buffer zeigt und fügt ein abschließendes Nullzeichen an.

Parameter

Instance	Bezeichnet eine Instanz des Moduls, dessen ausführbare Datei die String-Ressource enthält.
ID	Gibt den Integerbezeichner des Strings an, der geladen werden soll.
Buffer	Zeigt auf den Puffer, der den String aufnimmt.
BufferMax	Gibt die maximale Anzahl der Zeichen an, die in den Puffer kopiert werden sollen. Der String wird abgeschnitten, wenn er länger als die angegebene Zeichenanzahl ist.

Rückgabewert

Der Rückgabewert gibt die tatsächlich in den Puffer kopierte Anzahl von Zeichen an. Er ist 0, wenn die String-Ressource nicht existiert.

LoByte (Windows API Funktion)

Deklaration

```
function LoByte(A: Word): Byte;
  inline(
    $5A/        { POP AX    }
    $32/$E4);   { XOR AH,AH }
```

Beschreibung

Liefert das niederwertige Byte eines 16-Bit-Integers.

Parameter

A	16-Bit-Integer

Rückgabewert

Das niederwertige Byte

LoWord (Windows API Funktion)

Beschreibung

```
function LoWord(AnInteger: Longint): Word;
```

Beschreibung

Liefert das niederwertige Word eines 32-Bit-Intergers.

Parameter

AnInteger 32-Bit-Integerwert

Rückgabewert

Das niederwertige Word

LocalAlloc (Windows API Funktion)

Deklaration

```
function LocalAlloc(Flags, Bytes: Word): THandle;
```

Beschreibung

Diese Funktion reserviert die Anzahl von Bytes auf dem lokalen Heap, die vom Parameter Bytes angegeben wird.

Parameter

Flags	Gibt an, wie der Speicher reserviert werden soll. Er kann einen oder mehrere der folgenden Werte annehmen: lmem_Discardable lmem_Fixed lmem_Modify lmem_Moveable lmem_Nocompact lmem_Nodiscard lmem_Zeroinit
Bytes	Gibt die gesamte Anzahl an Bytes an, die reserviert werden sollen.

Rückgabewert

Bezeichnet den neu reservierten lokalen Speicherblock, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

Siehe auch

LockData

LocalCompact (Windows API Funktion)

Deklaration

```
function LocalCompact(MinFree: Word): Word;
```

Beschreibung

Diese Funktion stellt die Anzahl an freien Bytes des Speichers zur Verfügung, die vom Parameter MinFree angegeben wird, indem sie den lokalen Heap des Moduls komprimiert, falls es nötig ist. Falls dadurch der angeforderte Speicher immer noch nicht zur Verfügung gestellt werden kann, verwirft die Funktion nach Möglichkeit solange verschiebbare und verwerfbare Speicherblöcke, die nicht gesperrt sind, bis der angeforderte Speicher zur Verfügung steht.

Parameter

MinFree	Gibt die Anzahl an gewünschten freien Bytes an. Wenn MinFree 0 ist, liefert die Funktion zwar einen Wert zurück, komprimiert aber den Speicher nicht.

Rückgabewert

Der Rückgabewert gibt die Anzahl der Bytes im größten freien Block des lokalen Speichers an.

LocalDiscard (Windows API function)

Deklaration

```
function LocalDiscard(Mem: THandle): THandle;
```

Beschreibung

Diese Funktion verwirft den lokalen Speicherblock, der durch den Parameter Mem bestimmt wird. Der lokale Speicherblock wird zwar aus dem Speicher entfernt, aber sein Handle bleibt gültig, d.h. sein Handle kann anschließend an die Funktion **LocalReAlloc** übergeben werden.

Parameter

Mem Handle des lokalen Speicherblocks

Rückgabewert

Bei erfolgreicher 0, andernfalls Mem

LocalFlags (Windows API Funktion)

Deklaration

```
function LocalFlags(Mem: THandle): Word;
```

Beschreibung

Diese Funktion liefert Informationen über den angegebenen lokalen Speicherblock zurück.

Parameter

Mem Bezeichnet den lokalen Speicherblock.

Rückgabewert

Der Rückgabewert enthält eines der folgenden Speicherreservierungs-Flags im oberen Byte: lmem_Discardable oder lmem_Discarded
Das untere Byte des Rückgabewertes enthält den Referenzzähler des Blocks.

Siehe auch

lmem_xxx Flags

LocalFree (Windows API Funktion)

Deklaration

```
function LocalFree(Mem: THandle): THandle;
```

Beschreibung

Diese Funktion gibt den lokalen Speicherblock, der durch den Parameter Mem bezeichnet wird, frei und macht das Handle des Speicherblocks ungültig.

Parameter

Mem	Bezeichnet den lokalen Speicherblock, der freigegeben werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er mit Mem identisch.

LocalHandle (Windows API Funktion)

Deklaration

```
function LocalHandle(Mem: Word): THandle;
```

Beschreibung

Diese Funktion liefert das Handle des lokalen Speicherobjektes, dessen Adresse durch den Parameter Mem bestimmt wird.

Parameter

Mem Gibt die Adresse eines lokalen Speicherobjektes an.

Rückgabewert

Bezeichnet das lokale Speicherobjekt.

LocalInit (Windows API Funktion)

Deklaration

```
function LocalInit(Segment, Start, End: Word): Bool;
```

Beschreibung

Diese Funktion initialisiert einen lokalen Heap in dem Segment, das durch den Parameter Segment angegeben wird und sperrt das Segment durch Aufruf der Funktion **GlobalLock**.

Parameter

Segment	Gibt die Segmentadresse des Segments an, das den lokalen Heap enthalten soll.
Start	Gibt die Anfangsadresse des lokalen Heap innerhalb des Segments an.
End	Gibt die Endadresse des lokalen Heap innerhalb des Segments an.

Rückgabewert

Der Rückgabewert gibt einen Booleschen Wert an, der ungleich 0 ist, wenn der Heap initialisiert wurde. Andernfalls ist er 0.

LocalLock (Windows API Funktion)

Deklaration

```
function LocalLock(Mem: THandle): Pointer;
```

Beschreibung

Diese Funktion sperrt den lokalen Speicherblock, der durch den Parameter Mem bestimmt wird. Der Block wird im Speicher an der gegebenen Adresse gesperrt und sein Referenzzähler wird um 1 erhöht. Gesperrter Speicher kannt nicht verschoben oder verworfen werden.

Parameter

Mem	Bezeichnet den lokalen Speicherblock, der gesperrt werden soll.

Rückgabewert

Der Rückgabewert zeigt auf das erste Byte im Speicher des lokalen Blocks, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er **nil**.

LocalReAlloc (Windows API Funktion)

Deklaration

```
function LocalReAlloc(Mem: THandle; Bytes, Flags: Word): THandle;
```

Beschreibung

Diese Funktion vergrößert oder verkleinert den lokalen Speicherblock, der durch den Parameter Mem bestimmt wird, auf die Anzahl von Bytes, die durch den Parameter Bytes vorgegeben ist, oder ändert die Attribute des angegebenen Speicherblocks.

Parameter

Mem — Bezeichnet den lokalen Speicherblock, dessen Größe verändert werden soll.
Bytes — Gibt die neue Größe des Speicherblocks an.
Flags — Gibt an, wie der lokale Speicherblock verändert werden soll. Er kann einen oder mehrere der folgenen annehmen **lmem_xxx-Flags**

- lmem_Discardable
- lmem_Modify
- lmem_Moveable
- lmem_NoCompact
- lmem_NoDiscard
- lmem_ZeroInit

Rückgabewert

Bezeichnet den geänderten lokalen Speicher, wenn die Funktion erfolgreich ist. Er ist 0, wenn die Größe des lokalen Speicherblocks nicht verändert werden kann.

Siehe auch

LockData

LocalShrink (Windows API Funktion)

Deklaration

```
function LocalShrink(Seg: THandle; Size: Word): Word;
```

Beschreibung

Diese Funktion verkleinert den angegebenen Heap auf die Größe, die vom Parameter Size übergeben wird. Die kleinste mögliche Größe des automatischen lokalen Heap ist in der Moduldefinitionsdatei der Anwendung festgelegt.

Parameter

Seg	Bezeichnet das Segment, das den lokalen Heap enthält.
Size	Gibt die Größe (in Bytes) des lokalen Heap nach der erwünschten Verkleinerung an.

Rückgabewert

Der Rückgabewert gibt die Größe des lokalen Heap nach der Verkleinerung an.

Siehe auch

GlobalSize

LocalSize (Windows API Funktion)

Deklaration

```
function LocalSize(Mem: THandle): Word;
```

Beschreibung

Diese Funktion liefert die gegenwärtige Größe (in Bytes) des lokalen Speicherblocks, der vom Parameter Mem bestimmt wird.

Parameter

Mem Bezeichnet den lokalen Speicherblock.

Rückgabewert

Der Rückgabewert gibt die Größe (in Bytes) des angegebenen Speicherblocks an. Er ist 0, wenn das gegebene Handle ungültig ist.

LocalUnlock (Windows API Funktion)

Deklaration

```
function LocalUnlock(Mem: THandle): Bool;
```

Beschreibung

Diese Funktion hebt die Sperre des lokalen Speicherblocks, der durch den Parameter Mem bestimmt wird, auf und verringert den Referenzzähler des Blocks um 1.

Parameter

Mem — Bezeichnet den lokalen Speicherblock, dessen Sperre aufgehoben werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn der Referenzzähler des Blocks auf 0 verringert wurde. Andernfalls ist der Rückgabewert ungleich 0.

LockResource (Windows API Funktion)

Deklaration

```
function LockResource(ResData: THandle): Pointer;
```

Beschreibung

Diese Funktion ermittelt die absolute Speicheradresse der geladenen Ressource, die durch den Parameter ResData bezeichnet wird. Die Ressource wird im Speicher gesperrt und die gegebene Adresse und ihr Referenzzähler werden um 1 erhöht. Die gesperrte Ressource kann nicht verschoben oder verworfen werden.

Parameter

ResData	Bezeichnet die gewünschte Ressource. Es wird vorausgesetzt, daß dieses Handle durch die Funktion **LoadResource** erzeugt wurde.

Rückgabewert

Der Rückgabewert zeigt auf das erste Byte der geladenen Ressource, falls sie gesperrt wurde. Andernfalls ist er 0.

Siehe auch

UnlockResource

LockSegment (Windows API Funktion)

Deklaration

```
function LockSegment(Segment: Word): THandle;
```

Beschreibung

Diese Funktion sperrt das Segment, dessen Segmentadresse durch den Parameter Segment bestimmt wird. Wenn Segment -1 ist, sperrt die Funktion LockSegment das aktuelle Datensegment.
Mit Ausnahme von nichtverwerfbaren Segmenten im Protected Mode (Standard- oder erweiterter 386-Modus) wird das Segment an der gegebenen Adresse im Speicher gesperrt, und sein Sperrenzähler wird um 1 erhöht.

Im Protected Mode erhöht LockSegment ausschließlich den Sperrenzähler von verwerfbaren und automatischen Datensegmenten.

Parameter

Segment	Gibt die Segmentadresse des Segments an, das gesperrt werden soll. Wenn Segment -1 ist, sperrt die Funktion LockSegment das aktuelle Datensegment.

Rückgabewert

Bezeichnet das Datensegment, wenn die Funktion erfolgreich ausgeführt wurde. Wenn das Objekt verworfen wurde oder ein Fehler aufgetreten ist, ist der Rückgabewert **nil**.

Siehe auch

UnlockSegment

_lopen (Windows API Funktion)

Deklaration

```
function _lopen(PathName: PChar; ReadWrite: Integer): Integer;
```

Beschreibung

Diese Funktion öffnet die Datei mit dem Namen, der durch den Parameter PathName bestimmt wird.

Parameter

PathName — Zeigt auf einen null-terminierten String, der den Namen der zu öffnenden Datei enthält.

ReadWrite — Bestimmt, ob die Funktion die Datei zum Lesezugriff, zum Schreibzugriff, oder zu beidem öffnen soll. Der Parameter muß eine der folgenden **of_xxx Konstanten** annehmen:

- of_Read
- of_ReadWrite
- of_Write

Rückgabewert

Der Rückgabewert gibt das MS-DOS-Datei-Handle aus, wenn die Funktion die Datei geöffnet hat. Andernfalls ist er -1.

LPtoDP (Windows API Funktion)

Deklaration

```
function LPtoDP(DC: HDC; var Points; Count: Integer): Bool;
```

Beschreibung

Diese Funktion wandelt virtuelle Punkte in Gerätepunkte um. Die Funktion LPtoDP überträgt die Koordinaten jedes Punktes, der durch den Parameter Points angegeben wird, vom virtuellen Koordinatensystem des GDI in ein Gerätekoordinatensystem. Die Umwandlung hängt vom aktuellen Abbildungsmodus ab.

Parameter

DC	Bezeichnet den Gerätekontext.
Points	Zeigt auf ein Array von Punkten. Jeder Punkt im Array ist eine **TPoint**-Datenstruktur.
Count	Gibt die Anzahl von Punkten im Array an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn alle Punkte umgewandelt sind. Andernfalls ist er 0.

_lread (Windows API Funktion)

Deklaration

```
function _lread(FileHandle: Integer; Buffer: PChar; Bytes: Integer): Word;
```

Beschreibung

Diese Funktion liest Daten aus der offenen Datei, die durch den Parameter File bestimmt wird.

Parameter

File	Gibt das MS-DOS-Datei-Handle der Datei an, die gelesen werden soll.
Buffer	Zeigt auf einen Puffer, der die Daten aufnehmen soll, die aus der Datei gelesen werden.
Bytes	Gibt die Anzahl der Bytes an, die aus der Datei gelesen werden sollen.

Rückgabewert

Der Rückgabewert liefert die Anzahl der Bytes, die die Funktion tatsächlich aus der Datei gelesen hat, oder -1, wenn die Funktion fehlschlägt.

lstrcat (Windows API Funktion)

Deklaration

```
function lstrcat(Str1, Str2: PChar): PChar;
```

Beschreibung

Diese Funktion hängt Str2 an den String an, der durch Str1 bestimmt wird, schließt den entstehenden String mit einem Nullzeichen ab, und liefert einen far-Zeiger auf den verbundenen String (Str1).
Alle Strings müssen kleiner als 64 KByte sein.

Parameter

Str1	Zeigt auf ein Byte-Array, das einen null-terminierten String enthält, an die die Funktion Str2 anhängen soll. Dieses Byte-Array, das den String enthält, muß groß genug sein, um beide Strings aufzunehmen.
Str2	Zeigt auf einen null-terminierten String, den die Funktion an Str1 anhängen soll.

Rückgabewert

Der Rückgabewert liefert einen Zeiger auf Str1. Er ist 0, wenn die Funktion nicht erfolgreich ausgeführt werden konnte.

lstrcmp (Windows API Funktion)

Deklaration

```
function lstrcmp(Str1, Str2: PChar): Integer;
```

Beschreibung

Diese Funktion vergleicht die beiden Strings, die durch Str1 und Str2 bezeichnet werden, lexikographisch. Der Vergleich basiert auf der aktuellen Sprache, die vom Anwender im Setup oder in der Systemsteuerung ausgewählt wurde. Der Vergleich unterscheidet zwischen Groß- und Kleinschreibung.

Parameter

Str1	Zeigt auf den ersten null-terminierten String, der verglichen werden soll.
Str2	Zeigt auf den zweiten null-terminierten String, der verglichen werden soll.

Rückgabewert

Der Rückgabewert zeigt an, ob Str1 kleiner, gleich oder größer als Str2 ist.

Wert	Bedeutung
< 0	Str1 ist kleiner als Str2.
= 0	Str1 ist mit Str2 identisch.
> 0	Str1 ist größer als Str2.

lstrcmpi (Windows API Funktion)

Deklaration

```
function lstrcmpi(Str1, Str2: PChar): Integer;
```

Beschreibung

Diese Funktion vergleicht die beiden Strings, die durch Str1 und Str2 bezeichnet werden, lexikographisch. Der Vergleich unterscheidet nicht zwischen Groß- und Kleinschreibung.

Parameter

Str1	Zeigt auf den ersten null-terminierten String, der verglichen werden soll.
Str2	Zeigt auf den zweiten null-terminierten String, der verglichen werden soll.

Rückgabewert

Der Rückgabewert zeigt an, ob Str1 kleiner, gleich oder größer als Str2 ist:

Wert	Bedeutung
< 0	Str1 ist kleiner als Str2.
= 0	Str1 ist mit Str2 identisch.
> 0	Str1 ist größer als Str2.

lstrcpy (Windows API Funktion)

Deklaration

```
function lstrcpy(Str1, Str2: PChar): PChar;
```

Beschreibung

Diese Funktion kopiert Str2 einschließlich des abschließenden Nullzeichens in den Bereich, der von Str1 bestimmt wird und gibt Str1 zurück.
Alle Strings müssen eine Größe von weniger als 64 KByte haben.

Parameter

Str1	Zeigt auf einen Puffer, der den Inhalt von Str2 aufnehmen soll. Der Puffer muß groß genug für Str2 sein.
Str2	Zeigt auf den null-terminierten String, der kopiert werden soll.

Rückgabewert

Der Rückgabewert gibt einen Zeiger auf Str1 an. Er ist 0, wenn die Funktion gescheitert ist.

lstrlen (Windows API Funktion)

Deklaration

```
function lstrlen(Str: PChar): Integer;
```

Beschreibung

Diese Funktion gibt die Länge von Str in Bytes ohne das abschließende Nullzeichen zurück.
Der String muß kleiner als 64 KByte sein.

Parameter

Str Zeigt auf einen null-terminierten String.

Rückgabewert

Der Rückgabewert gibt die Länge von Str an. Es gibt keinen Fehlerrückgabewert.

_lwrite (Windows API Funktion)

Deklaration

```
function _lwrite(FileHandle: Integer; Buffer: PChar; Bytes: Integer):
Word;
```

Beschreibung

Diese Funktion schreibt Daten aus einem Puffer in die Datei, die durch den Parameter File bestimmt wird.

Parameter

FileHandle	Gibt das DOS-Datei-Handle der Datei an, in die geschrieben werden soll.
Buffer	Zeigt auf einen Puffer, der die Daten enthält, die in die Datei geschrieben werden sollen.
Bytes	Gibt die Anzahl von Bytes an, die in die Datei geschrieben werden sollen.

Rückgabewert

Der Rückgabewert liefert die Anzahl von Bytes, die die Funktion tatsächlich in die Datei geschrieben hat oder -1, wenn die Funktion fehlschlägt.

MakeProcInstance (Windows API Funktion)

Deklaration

```
function MakeProcInstance(Proc: TFarProc; Instance: THandle): TFarProc;
```

Beschreibung

Diese Funktion erzeugt die Adresse einer Prozedurinstanz. Eine Prozedurinstanzadresse zeigt auf den Prologcode, der vor der Ausführung der Funktion ausgeführt wird. Der Prolog verknüpft das Datensegment der Instanz, die durch den Parameter Instance bezeichnet wird, mit der Funktion, auf die der Parameter Proc zeigt. Wenn die Funktion ausgeführt wird, hat sie Zugriff auf Variablen und Daten im Datensegment dieser Instanz.

Parameter

Proc	**TFarProc** Prozedurinstanzadresse .
Instance	Bezeichnet die Instanz, die dem gewünschten Datensegment zugeordnet ist.

Rückgabewert

Der Rückgabewert zeigt auf die Funktion, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

MapDialogRect (Windows API Prozedur)

Deklaration

```
procedure MapDialogRect(Dlg: HWnd; var Rect: TRect);
```

Beschreibung

Diese Funktion konvertiert die Dialogfenstereinheiten, die im Parameter Rect angegeben sind, in Bildschirmeinheiten.

Parameter

Dlg	Bezeichnet ein Dialogfenster.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die die zu konvertierenden Dialogfensterkoordinaten enthält.

Siehe auch

GetDialogBaseUnits

MapVirtualKey (Windows API Funktion)

Deklaration

```
function MapVirtualKey(Code, MapType: Word): Word;
```

Beschreibung

Diese Funktion nimmt einen virtuellen Tastencode oder einen Scan-Code für eine Taste entgegen und gibt den zugehörigen Scan-Code, virtuellen Tastencode oder ASCII-Wert zurück. Der Wert des Parameters MapType legt den Typ der Konversion fest, den diese Funktion ausführt.

Parameter

Code Gibt den virtuellen Tastencode oder Scan-Code für eine Taste an. Die Interpretation des Parameters wCode hängt vom Wert des Parameters MapType ab.

MapType Gibt den Typ der Konversion an, die ausgeführt werden soll. Der Parameter MapType kann einen der Werte annehmen:

0 Der Parameter Code bezeichnet einen virtuellen Tastencode, die Funktion gibt den zugehörigen Scan-Code zurück.

1 Der Parameter Code bezeichnet einen Scan-Code, die Funktion gibt den zugehörigen virtuellen Tastencode zurück.

2 Der Parameter Code bezeichnet einen virtuellen Tastencode, die Funktion gibt den zugehörigen ASCII-Wert zurück, der gilt, wenn die Shift-Taste nicht gedrückt ist.

Rückgabewert

Der Rückgabewert hängt vom Wert der Parameter Code und MapType ab.

MessageBeep (Windows API Prozedur)

Deklaration

```
procedure MessageBeep(BeepType: Word);
```

Beschreibung

Diese Funktion erzeugt einen Warnton am Systemlautsprecher.

Parameter

BeepType Wird nicht verwendet. Er sollte auf 0 gesetzt werden.

MessageBox (Windows API Funktion)

Deklaration

```
function MessageBox(Parent: HWnd; Txt, Caption: PChar; TextType: Word):
Integer;
```

Beschreibung

Diese Funktion erzeugt ein Fenster und stellt es mit einer von der Anwendung bereitgestellten Meldung und Kopfzeile und einer Kombination vordefinierter Symbole und Schalter dar.

Parameter

Parent	Bezeichnet das Fenster, dem das Meldungsfenster gehört.
Text	Zeigt auf einen null-terminierten String, die die anzuzeigende Meldung enthält
Caption	Zeigt auf einen null-terminierten String, die für die Überschrift des Dialogfensters benutzt wird. Wenn der Parameter Caption **nil** ist, wird die Standardüberschrift "Error" bzw. "Fehler" benutzt.
Type	Gibt den Inhalt des Dialogfensters an. Er kann jede Kombination der **mb_xxx -Konstanten** annehmen.

Rückgabewert

Der Rückgabewert ist 0, wenn nicht genug Speicherplatz vorhanden ist, um das Dialogfenster zu erzeugen. Andernfalls wird einer der **Menüoptionswerte** vom Dialogfenster zurückgegeben:

ModifyMenu (Windows API Funktion)

Deklaration

```
function ModifyMenu(Menu: HMenu; Position, Flags, IDNewItem: Word;
NewItem: PChar): Bool;
```

Beschreibung

Diese Funktion ändert eine vorhandene Menüoption an der Position, die durch den Parameter Position bestimmt wird. Die Anwendung gibt den neuen Status der Menüoption vor, indem sie die Werte im Parameter Flags einstellt. Wenn diese Funktion ein der Menüoption zugeordnetes Pop-Up-Menü ersetzt, dann zerstört sie das alte Pop-Up-Menü und gibt den Speicher frei, der vom Pop-Up-Menü benutzt wurde.

Parameter

Menu — Bezeichnet das Menü, das verändert werden soll.

Position — Bestimmt die zu verändernde Menüoption.

Flags — Kombination von mf_ByCommand oderr mf_ByPosition mit einer der folgenden **mf_xxx-Konstanten**:

- mf_Disabled
- mf_Enabled
- mf_Grayed
- mf_Bitmap
- mf_String
- mf_OwnerDraw
- mf_Separator
- mf_Popup
- mf_MenuBarBreak
- mf_MenuBreak
- mf_Checked
- mf_Unchecked

IDNewItem — Gibt entweder die Befehls-ID der modifizierten Menüoption an oder, wenn Flags auf mf_Popup gesetzt ist, das Menü-Handle des Pop-Up-Menüs.

NewItem — Gibt den Inhalt der geänderten Menüoption an. Wenn Flags auf mf_String gesetzt ist (Voreinstellung), dann ist NewItem ein Zeiger auf einen null-terminierten String. Wenn Flags auf mf_Bitmap gesetzt ist, dann enthält NewItem in seinem niederwertigen Word ein Bitmap-Handle (HBITMAP). Wenn Flags auf mf_OwnerDraw gesetzt ist, dann bestimmt NewItem einen von der Anwendung bereitgestellten 32-Bit- Wert, den die Anwendung benutzen kann, um zusätzliche Daten zu verwalten, die mit der Menüoption verbunden sind.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

MoveTo (Windows API Funktion)

Deklaration

```
function MoveTo(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion stellt den Punkt, der durch die Parameter X und Y bestimmt wird, als aktuelle Position ein.

Parameter

DC Bezeichnet den Gerätekontext.
X, YGibt die virtuellen Koordinaten der neuen Position an.

Rückgabewert

Der Rückgabewert gibt die x- und y-Koordinate der vorherigen Position an. Die y-Koordinate liegt im höherwertigen, die x-Koordinate im niederwertigen Word.

MoveWindow (Windows API Prozedur)

Deklaration

```
procedure MoveWindow(Wnd: HWnd; X, Y, Width, Height: Integer; Repaint: Bool);
```

Beschreibung

Diese Funktion bewirkt, daß eine Botschaft **wm_Size** an das vorgegebene Fenster gesendet wird. Die mit wm_Size übergebenen Parameter beziehen sich auf den Client-Bereich.

Parameter

Wnd	Bzeichnet ein untergeordnetes oder Pop-Up-Fenster.
X, YGibt	die neuen Koordinaten der oberen linken Ecke des Fensters an. Für Pop-Up-Fenster sind X und Y in Bildschirmkoordinaten gegeben (relativ zur oberen linken Ecke des Bildschirms). Für untergeordnete Fenster gelten sie in Client-Koordinaten (relativ zur oberen linken Ecke des Client-Bereichs des übergeordneten Fensters).
Width	Bestimmt die neue Breite des Fensters.
Height	Bestimmt die neue Höhe des Fensters.
Repaint	Bestimmt, ob das Fenster nach der Verschiebung neu gezeichnet werden soll. Wenn Repaint 0 ist, dann wird das Fenster nicht neu gezeichnet.

OemKeyScan (Windows API Funktion)

Deklaration

```
function OemKeyScan(OemChar: Word): Longint;
```

Beschreibung

Diese Funktion überträgt die OEM-ASCII-Codes 0 bis 0x0FF in die OEM Scan-Codes und Shift-Zustände.

Parameter

OemChar Gibt den ASCII-Wert des OEM-Zeichens (0 - $0FF) an.

Rückgabewert

Der Rückgabewert enthält in seinem niederwertigen Word den Scan-Code des OEM-Zeichens. Das höherwertige Word des Rückgabewerts enthält Flags, die den Shift-Zustand angeben:

2 CTRL-Taste ist gedrückt.
1 Eine SHIFT-Taste ist gedrückt.

Wenn das Zeichen nicht in der OEM-Zeichentabelle definiert ist, enthält sowohl das höherwertige als auch das niederwertige Word des Rückgabewerts -1.

OemToAnsi (Windows API Funktion)

Deklaration

```
function OemToAnsi(OemStr, AnsiStr: PChar): Bool;
```

Beschreibung

Diese Funktion übersetzt den String, auf den der Parameter OemStr zeigt, vom OEM-definierten Zeichensatz in den ANSI-Zeichensatz.

Parameter

OemStr	Zeigt auf einen null-terminierten String aus dem OEM-Zeichensatz.
AnsiStr	Zeigt auf die Position, an die der übersetzte String kopiert werden soll. Der Parameter AnsiStr kann derselbe wie OemStr sein, um den String an der aktuellen Position zu übersetzen.

Rückgabewert

Der Rückgabewert ist immer (False) -1.

OemToAnsiBuff (Windows API Prozedur)

Deklaration

```
procedure OemToAnsiBuff(OemStr, AnsiStr: PChar; Length: Integer);
```

Beschreibung

Diese Funktion übersetzt den String in dem Puffer, auf den der Parameter OemStr zeigt, vom OEM-Zeichensatz in den ANSI-Zeichensatz.

Parameter

OemStr	Zeigt auf einen Puffer, der ein oder mehrere Zeichen aus dem OEM-Zeichensatz enthält.
AnsiStr	Zeigt auf die Position, an die der String kopiert werden soll. Der Parameter AnsiStr kann mit OemStr identisch sein, um den String an der aktuellen Position zu übersetzen.
Length	Gibt die Zahl der Zeichen in dem Puffer an, der durch den Parameter OemStr bezeichnet wird.

OffsetClipRgn (Windows API Funktion)

Deklaration

```
function OffsetClipRgn(DC: HDC; X, Y: Integer): Integer;
```

Beschreibung

Diese Funktion verschiebt die Clipping-Region des gegebenen Gerätes um die angegebenen Offsetwerte. Diese Funktion verschiebt die Region um X Einheiten entlang der x-Achse und Y Einheiten entlang der y-Achse.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Bezeichnet die Anzahl der virtuellen Einheiten, um die nach links oder rechts verschoben werden soll.
Y	Bezeichnet die Zahl der virtuellen Einheiten, um die nach oben oder unten verschoben werden soll.

Rückgabewert

Bezeichnet den Typ der neuen Region. Er kann einen der folgenden Werte der **Bereichsflags** annehmen:

OffsetRect (Windows API Prozedur)

Deklaration

```
procedure OffsetRect(var Rect: TRect; X, Y: Integer);
```

Beschreibung*

Diese Funktion verschiebt das gegebene Rechteck um die angegebenen Offsetwerte.

Parameter

Rect	Zeigt auf eine **TRect**-Datenstruktur, die das zu verschiebende Rechteck enthält.
X	Gibt den Betrag der Verschiebung nach rechts oder links an.
Y	Gibt den Betrag der Verschiebung nach oben oder unten an. Zum Verschieben nach oben muß er negativ sein.

OffsetRgn (Windows API Funktion)

Deklaration

```
function OffsetRgn(Rgn: HRgn; X, Y: Integer): Integer;
```

Beschreibung

Diese Funktion verschiebt die angegebene Region um die gegebenen Offsetwerte. Die Funktion verschiebt um X Einheiten entlang der x-Achse, und um Y Einheiten entlang der y-Achse.

Parameter

Rgn	Bezeichnet die zu verschiebenden Region.
X	Bezeichnet die Anzahl der Einheiten, um die nach links oder rechts verschoben werden soll.
Y	Bezeichnet die Anzahl der Einheiten, um die nach oben oder unten verschoben werden soll.

Rückgabewert

Bezeichnet den Typ der neuen Region, ist eine **Bereichsflags**-Konstanten.

OffsetViewportOrg (Windows API Funktion)

Deklaration

```
function OffsetViewportOrg(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion verschiebt den Ursprung desGrafikfensters relativ zu den aktuellen Koordinaten. Der neue Ursprung ist die Summe des aktuellen Ursprungs und der Werte X und Y.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Gibt die Anzahl der gerätespezifischen Einheiten an, die zur X-Koordinate des aktuellen Ursprungs addiert werden sollen.
Y	Gibt die Anzahl der gerätespezifischen Einheiten an, die zur Y-Koordinate des aktuellen Ursprungs addiert werden sollen.

Rückgabewert

Der Rückgabewert bestimmt den vorherigen Ursprung des Fensters in gerätespezifischen Koordinaten. Die vorherige Y-Koordinate liegt im höherwertigen, die vorherige X-Koordinate im niederwertigen Word.

OffsetWindowOrg (Windows API Funktion)

Deklaration

```
function OffsetWindowOrg(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion verändert den Ursprung des Fensters relativ zu den aktuellen Werten. Der neue Ursprung ist die Summe des aktuellen Ursprungs und der Werte X und Y.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Gibt die Anzahl der virtuellen Einheiten an, die zur X-Koordinate des aktuellen Ursprungs addiert werden sollen.
Y	Gibt die Anzahl der virtuellen Einheiten an, die zur Y-Koordinate des aktuellen Ursprungs addiert werden sollen.

Rückgabewert

Der Rückgabewert bestimmt den vorherigen Fensterursprung in virtuellen Koordinaten.Die vorherige Y-Koordinate steht im höherwertigen, die vorherige X-Koordinate im niederwertigen Word.

OpenClipboard (Windows API Funktion)

Deklaration

```
function OpenClipboard(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion öffnet die Zwischenablage zur Untersuchung und hindert andere Anwendungen daran, den Inhalt der Zwischenablage zu ändern.

Parameter

Wnd	Bezeichnet das Fenster, das mit der offenen Zwischenablage verbunden werden soll.

Rückgabewert

Bezeichnet den Status der Zwischenablage. Er ist ungleich 0, wenn die Zwischenablage geöffnet ist. Wenn die Zwischenablage schon von einer anderen Anwendung geöffnet wurde, ist der Rückgabewert 0.

Siehe auch

CloseClipboard

OpenComm (Windows API Funktion)

Deklaration

```
function OpenComm(ComName: PChar; InQueue, OutQueue: Word): Integer;
```

Beschreibung

Diese Funktion öffnet eine Schnittstelle. Die Schnittstelle wird für eine Standardkonfiguration initialisiert und die Funktion OpenComm belegt Speicherplatz für Empfangs- und Sendeschlangen.

Parameter

ComName Weist auf einen String, der COMn oder LPTn enthält, wobei n ein Integer im Bereich von 1 bis zur Anzahl der verfügbaren Schnittstellen für den jeweiligen I/O-Anschluß liegt.

InQueue ibt die Größe der Empfangsschlange an.

OutQueue Gibt die Größe der Sendeschlange an.

Rückgabewert

Bezeichnet die geöffnete Schnittstelle. Wenn ein Fehler auftritt, hat der Rückgabewert einen negativen Fehlerwert (d.h. ist ein **ie_xxx-Flag**

Siehe auch

SetCommState

OpenFile (Windows API Funktion)

Deklaration

```
function OpenFile(FileName: PChar; var ReOpenBuff: TOFStruct; Style:
Word): Integer;
```

Beschreibung

Diese Funktion legt eine Datei an, öffnet eine Datei, öffnet sie wieder oder löscht eine Datei.

Parameter

FileName Zeigt auf einen null-terminierten String, die die zu öffnende Datei benennt.

ReOpenBuff Zeigt auf die Datenstruktur, die Informationen über die Datei empfängt, wenn diese sich auf die geöffnete Datei zu beziehen.

Style Bezeichnet die auszuführende Operation, ist eine der **of_xxx-Konstanten**.

Rückgabewert

Bezeichnet ein DOS-Datei-Handle, wenn die Funktion erfolgreich ist. Andernfalls ist er -1.

OpenIcon (Windows API Funktion)

Deklaration

```
function OpenIcon(Wnd: HWnd): Bool;
```

Beschreibung

Diese Funktion aktiviert ein zum Symbol verkleinertes Fenster und zeigt es an.

Parameter

Wnd Bezeichnet das Fenster.

Rückgabewert

Bezeichnet ist ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

OpenSound (Windows API Funktion)

Deklaration

```
function OpenSound: Integer;
```

Beschreibung

Diese Funktion greift auf das Klangausgabegerät zu und verhindert, daß es nachfolgend von anderen Anwendungen angesteuert wird.

Rückgabewert

Der Rückgabewert gibt die Anzahl der verfügbaren **s_xxx-Konstanten** an. Der Rückgabewert ist s_SerDVNA , wenn das Klangausgabegerät bereits verwendet wird und s_SerOFM, wenn nicht genügend Speicherplatz verfügbar ist.

OutputDebugString (Windows API Prozedur)

Deklaration

```
procedure OutputDebugString(OutputString: PChar);
```

Beschreibung

Diese Funktion übergibt eine Debuggerbotschaft OutputString an den Debugger, wenn er vorhanden ist oder an die Hilfsschnittstelle (AUX), wenn kein Debugger vorhanden ist.

Parameter

OutputString	Zeigt auf einen null-terminierten String

PaintRgn (Windows API Funktion)

Deklaration

```
function PaintRgn(DC: HDC; Rgn: HRgn): Bool;
```

Beschreibung

Diese Funktion füllt die vom Parameter Rgn bestimmte Region mit dem ausgewählten Pinsel aus.

Parameter

DC	Bezeichnet den Gerätekontext, der die Region enthält.
Rgn	Bezeichnet die Region, die ausgefüllt werden soll. Die Koordinaten für die angegebene Region sind in gerätespezifischen Einheiten angegeben.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

PatBlt (Windows API Funktion)

Deklaration

```
function PatBlt(DC: HDC; X, Y, Width, Height: Integer; Rop: Longint):
Bool;
```

Beschreibung

Diese Funktion erzeugt auf dem angegebenen Gerät ein Bitmuster. Das Muster ist eine Kombination des ausgewählten Pinsels mit dem Muster, das sich schon auf dem Gerät befindet. Der vom Parameter Rop angegebene Rasteroperationscode bestimmt, wie die Muster kombiniert werden sollen.

Parameter

DC	Bezeichnet den Gerätekontext.
X, Y	Gibt die virtuellen Koordinaten der oberen linken Ecke des Rechtecks an, das das Muster empfangen soll.
Width	Gibt die Breite des Rechtecks (in virtuellen Einheiten) an, das das Muster empfangen soll.
Height	Gibt die Höhe des Rechtecks (in virtuellen Einheiten) an.
Rop	Gibt den **Rasteroperationscode**: PatCopy PatInvert DSTInvert Blackness Whiteness

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Bitmuster gezeichnet wurde. Andernfalls ist er 0.

PeekMessage (Windows API Funktion)

Deklaration

```
function PeekMessage(var Msg: TMsg; Wnd: HWnd; MsgFilterMin, MsgFilterMax,
RemoveMsg: Word): Bool;
```

Beschreibung

Diese Funktion durchsucht die Anwendungwarteschlange nach einer Botschaft und legt die Botschaft (falls vorhanden) in der Datenstruktur ab, auf die der Parameter Msg zeigt. Wenn keine Botschaften vorliegen, kehrt die Funktion sofort zurück und übergibt Steuerung an Windows.

Parameter

Msg — Zeigt auf eine Datenstruktur **TMsg**, die die Botschaftsinformation aus der Windows-Anwendungsschlange enthält.

Wnd — Bezeichnet das Fenster, dessen Botschaften überprüft werden sollen, oder 0 für Windows-Anwendungen oder -1 für Botschaften, die von der Funktion **PostAppMessage** gesendet werden.

MsgFilterMin — Gibt den Wert der untersten Botschaftsposition an, die überprüft werden soll oder 0, wenn der Bereich nicht begrenzt werden soll.

MsgFilterMax — Gibt den Wert der höchsten Botschaftsposition an, die überprüft werden soll oder 0, wenn der Bereich nicht begrenzt werden soll.

RemoveMsg — Gibt eine Kombination der **pm_xxx-** Flags an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn eine Botschaft verfügbar ist. Andernfalls ist er 0.

Siehe auch

GetMessage
WaitMessage

Pie (Windows API Funktion)

Deklaration

```
function Pie(DC: HDC; X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet ein Tortendiagramm, indem sie einen Ellipsenbogen zeichnet, dessen Mittelpunkt und zwei Endpunkte durch Linien miteinander verbunden werden. Der Mittelpunkt des Bogens ist der Mittelpunkt des umgrenzenden Rechtecks, das die Parameter X1, Y1, X2 und Y2 vorgeben. Die Start- und Endpunkte des Bogens sind durch die Parameter X3, Y3, X4 und Y4 bezeichnet. Der Bogen wird mit dem ausgewählten Stift entgegen dem Uhrzeigersinn gezeichnet. Zusätzlich werden zwei Linien vom Endpunkt zum Mittelpunkt des Bogens gezeichnet.
Das Tortenbild wird mit dem ausgewählten Pinsel ausgefüllt.

Parameter

DC	Bezeichnet den Gerätekontext.
X1, Y1	Gibt die virtuellen Koordinaten der oberen linken Ecke des umgebenden Rechtecks an.
X2,, Y2	Gibt die virtuellenKoordinaten der unteren rechten Ecke des umgebenden Rechtecks an.
X3, Y3	Gibt die virtuellen Koordinaten des Startpunkts des Bogens an. Dieser Punkt muß nicht exakt auf dem Bogen liegen.
X4, Y4	Gibt die virtuellen Koordinaten des Endpunkts des Bogens an. Dieser Punkt muß nicht exakt auf dem Bogen liegen.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Tortendiagramm gezeichnet wurde. Andernfalls ist er 0.

PlayMetaFile (Windows API Funktion)

Deklaration

```
function PlayMetaFile(DC: HDC; MF: THandle): Bool;
```

Beschreibung

Diese Funktion spielt den Inhalt der bezeichneten Metadatei auf dem angegebenen Gerät ab. Die Metadatei kann beliebig oft wiedergegeben werden.

Parameter

DC Bezeichnet den Gerätekontext des Ausgabegerätes.
MF Bezeichnet die Metadatei.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

PlayMetaFileRecord (Windows API Prozedur)

Deklaration

```
procedure PlayMetaFileRecord(DC: HDC; var HandleTable: THandleTable; var
MetaRecord: TMetaRecord; Handles: Word);
```

Beschreibung

Diese Funktion spielt einen Metadatei-Record durch Ausführung desGDI-Funktionsaufrufes ab, der in dem Metadatei-Record enthalten ist.

Parameter

DC	Bezeichnet den Gerätekontext des Ausgabegeräts.
Handletable	Zeigt auf eine Objekt-Handle-Tabelle, die für die Metadatei-Wiedergabe benutzt werden soll.
MetaRecord	Zeigt auf die Metadatei, die wiedergegeben werden soll.
Handles	Gibt die Anzahl der Handles in der Handle-Tabelle an.

Siehe auch

EnumMetaFile

Polygon (Windows API Funktion)

Deklaration

function Polygon(DC: HDC; **var** Points; Count: Integer): Bool;

Beschreibung

Diese Funktion zeichnet ein Polygon, das aus zwei oder mehr Punkten (Eckpunkten) besteht, die durch Linien verbunden sind. Diese Polygone werden unter Benutzung des aktuellen Polygonfüllmodus ausgefüllt. Das Polygon wird automatisch geschlossen, falls notwendig, durch Zeichnen einer Linie vom letzten Eckpunkt zum ersten.

Parameter

DC	Bezeichnet den Gerätekontext.
Points	Zeigt auf ein Array von Punkten, die die Eckpunkte des Polygons bezeichnen. Jeder Punkt im Array ist eine **TPoint**-Datenstruktur.
Count	Bestimmt die Zahl von Eckpunkten, die im Array bezeichnet sind.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

Siehe auch

SetPolyFillMode

Polyline (Windows API Funktion)

Deklaration

```
function Polyline(DC: HDC; var Points; Count: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet mit dem gewählten Stift ine Reihe von Liniensegmenten, die die vom Parameter Points bezeichneten Punkte verbinden.

Parameter

DC	Bezeichnet den Gerätekontext.
Points	Zeigt auf ein Array von Punkten, die verbunden werden sollen. Jeder Punkt in dem Array ist eine **TPoint**-Datenstruktur.
Count	Gibt die Zahl der Punkte im Array an. Der Parameter Count muß mindestens 2 sein.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Liniensegmente gezeichnet wurden. Andernfalls ist er 0.

PolyPolygon (Windows API Funktion)

Deklaration

```
function PolyPolygon(DC: HDC; var Points; var PolyCounts; Count: Integer): Bool;
```

Beschreibung

Diese Funktion erzeugt eine Reihe von geschlossenen Polygonen. Die Polygone werden im aktuellen Polygonfüllmodus ausgefüllt. Die Polygone können, müssen sich aber nicht überlappen.

Parameter

DC	Bezeichnet den Gerätekontext.
Points	Zeigt auf ein Array von **TPoint**-Datenstrukturen, die die Eckpunkte der Polygone definiert.
PolyCounts	Zeigt auf ein Array von Integerwerten, von denen jeder die Anzahl von Punkten in einem der Polygone im Array Points bezeichnet.
Count	Gibt die Gesamtzahl von Integerwerten im Array PolyCounts an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Polygone gezeichnet wurden. Andernfalls ist er 0.

PostAppMessage (Windows API Funktion)

Deklaration

```
function PostAppMessage(Task: THandle; Msg, wParam: Word; lParam:
Longint): Bool;
```

Beschreibung

Diese Funktion übermittelt an eine Anwendung, die von einem Task-Handle bezeichnet wird, eine Botschaft und kehrt dann zurück, ohne darauf zu warten, daß die Anwendung die Botschaft verarbeitet. Der Parameter Wnd der zurückgegebenen MSG-Datenstruktur ist 0.

Parameter

Task	Bezeichnet die Task, die die Botschaft empfangen soll
Msg	Gibt den Typ der übermittelten Botschaft an.
wParam	Gibt zusätzliche Botschaftsinformationen an.
lParam	Gibt zusätzliche Botschaftsinformationen an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Botschaft übermittelt wurde. Andernfalls ist er 0.

Siehe auch

GetCurrentTask
GetMessage
PeekMessage

PostMessage (Windows API Funktion)

Deklaration

```
function PostMessage(Wnd: HWnd; Msg, wParam: Word; lParam: Longint): Bool;
```

Beschreibung

Diese Funktion plaziert eine Botschaft in der Anwendungsschlange eines Fensters und kehrt dann zurück, ohne darauf zu warten, daß das zugehörige Fenster die Botschaft bearbeitet.

Parameter

Wnd	Bezeichnet das Fenster, das die Botschaft empfangen soll. Wenn der Parameter Wnd 0xFFFF ist, dann wird die Botschaft an alle überlappten oder Pop-Up-Fenster im System gesendet. Die Botschaft wird nicht an untergeordnete Fenster gesendet.
Msg	Gibt den Typ der übermittelten Botschaft an.
wParam	Gibt zusätzliche Botschaftsinformationen an.
lParam	Gibt zusätzliche Botschaftsinformationen an.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Botschaft übermittelt wurde. Andernfalls ist er 0.

PostQuitMessage (Windows API Prozedur)

Deklaration

```
procedure PostQuitMessage(ExitCode: Integer);
```

Beschreibung

Diese Funktion sendet eine **wm_Quit** Botschaft und informiert Windows damit darüber, daß die Anwendung ihre Ausführung beenden möchte. Sie wird üblicherweise als Antwort auf eine Botschaft **wm_Destroy** benutzt.

Parameter

ExitCode	Gibt einen Exit-Code der Anwendung an. Er wird so wie der Parameter wParam in der Botschaft wm_Quit benutzt.

PtInRect (Windows API Funktion)

Deklaration

```
function PtInRect(var Rect: TRect; Point: TPoint): Bool;
```

Beschreibung

Diese Funktion gibt an, ob der angegebene Punkt innerhalb oder auf dem linken oder oberen Rand eines vorgegebenen Rechtecks liegt.

Parameter

Rect	Zeigt auf eine **TRect**-Datenstruktur, die das angegebene Rechteck enthält.
Point	Zeigt auf eine **TPoint**-Datenstruktur, die den angegebenen Punkt enthält.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der Punkt innerhalb des angegebenen Rechtecks liegt. Andernfalls ist er 0.

PtInRegion (Windows API Funktion)

Deklaration

```
function PtInRegion(Rgn: HRgn; X, Y: Integer): Bool;
```

Beschreibung

Diese Funktion gibt an, ob der von den Parametern X und Y bezeichnete Punkt in der angegebenen Region liegt.

Parameter

Rgn	Bezeichnet die Region, die überprüft werden soll.
X	Bestimmt die virtuelle x-Koordinate des Punktes.
Y	Bestimmt die virtuelle y-Koordinate des Punktes.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der Punkt in der Region liegt. Andernfalls ist er 0.

PtVisible (Windows API Funktion)

Deklaration

```
function PtVisible(DC: HDC; X, Y: Integer): Bool;
```

Beschreibung

Diese Funktion gibt an, ob der angebene Punkt innerhalb der Clipping-Region des vorgegebenen Gerätekontexts liegt.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Bestimmt die virtuelle x-Koordinate des Punktes.
Y	Bestimmt die virtuelle y-Koordinate des Punktes.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der Punkt innerhalb der Clipping-Region liegt. Andernfalls ist er 0.

ReadComm (Windows API Funktion)

Deklaration

```
function ReadComm(Cid: Integer; Buf: PChar, Size: Integer): Integer;
```

Beschreibung

Diese Funktion liest die Anzahl der Zeichen, die durch den Parameter Size bestimmt ist, aus der Schnittstelle, die durch den Parameter Cid bezeichnet ist und kopiert die Zeichen in den Puffer, auf den der Parameter Buf zeigt.

Parameter

Cid	Gibt die Schnittstelle an, von der gelesen werden soll.
Buf	Zeigt auf den Puffer, der die gelesenen Zeichen empfangen soll.
Size	Gibt die Zahl der zu lesenden Zeichen an.

Rückgabewert

Der Rückgabewert gibt die Zahl der tatsächlich gelesenen Zeichen an. Wenn der Rückgabewert 0 ist, sind keine Zeichen vorhanden.

Tritt ein Fehler auf, wird der Rückgabewert auf einen Wert kleiner 0 gesetzt, wobei der Absolutbetrag die Zahl der tatsächlich gelesenen Zeichen darstellt. Für parallele I/O-Anschlüsse ist der Rückgabewert immer 0.

Siehe auch

GetCommError
OpenComm

RealizePalette (Windows API Funktion)

Deklaration

```
function RealizePalette(DC: HDC): Word;
```

Beschreibung

Diese Funktion bildet die Systempaletteneinträge in der aktuell ausgewählten virtuellen Palette in einen Gerätekontext ab.

Parameter

DC Bezeichnet den Gerätekontext.

Rückgabewert

Der Rückgabewert bestimmt, wieviele Einträge seit der letzten Realisierungin der virtuellen Palette auf Einträge in der Systempalette abgebildet wurden.

Rectangle (Windows API Funktion)

Deklaration

```
function Rectangle(DC: HDC; X1, Y1, X2, Y2: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet ein Rechteck. Das Innere des Rechtecks wird mit dem gewählten Pinsel ausgefüllt; mit dem gewählten Stift wird eine Umrandung gezeichnet.

Parameter

DC	Bezeichnet den Gerätekontext.
X1, Y1	Bestimmt die virtuellen Koordinaten der oberen linken Ecke des Rechtecks.
X2, Y2	Bestimmt die virtuellen Koordinaten der unteren rechten Ecke des Rechtecks.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Rechteck gezeichnet wurde. Andernfalls ist er 0.

RectVisible (Windows API Funktion)

Deklaration

```
function RectVisible(DC: HDC; var Rect: TRect): Bool;
```

Beschreibung

Diese Funktion stellt fest, ob ein Teil des gegebenen Rechtecks innerhalb der Clipping-Region des angegebenen Bildschirmkontexts liegt.

Parameter

DC	Bezeichnet den Gerätekontext.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die die virtuellen Koordinaten des angegebenen Rechtecks enthält.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn ein Stück des angegebenen Rechtecks innerhalb der Clipping-Region liegt. Andernfalls ist er 0.

RegisterClass (Windows API Funktion)

Deklaration

```
function RegisterClass(var WndClass: TWndClass): Bool;
```

Beschreibung

Diese Funktion registriert eine Fensterklasse zur nachfolgenden Verwendung in Aufrufen der Funktion CreateWindow. Die Fensterklasse hat die Attribute, die durch die Datenstruktur definiert werden, auf die der Parameter WndClass zeigt. Wenn zwei Klassen mit demselben Namen registriert werden, dann schlägt der zweite Versuch fehl und die Information für diese Klasse wird ignoriert.

Parameter

WndClass Zeigt auf eine Datenstruktur **TWndClass**.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Klasse registriert wurde. Andernfalls ist er 0.

RegisterClipboardFormat (Windows API Funktion)

Deklaration

```
function RegisterClipboardFormat(FormatName: PChar): Word;
```

Beschreibung

Diese Funktion registriert ein neues Zwischenablageformat, auf dessen Bezeichnung der Parameter FormatName zeigt. Es erscheint in der Formatliste der Zwischenablage.

Parameter

FormatName — Zeigt auf einen null-termninierten String, die das neue Format benennt.

Rückgabewert

Der Rückgabewert bezeichnet das neu registrierte Format ($C000 bis $FFFF). Wenn der identische Formatname schon vorher registriert wurde (z.B. durch eine andere Anwendung), dann wird der Referenzzähler des Formats erhöht und es wird derselbe Wert zurückgegeben, als wäre das Format erstmals registriert worden. Der Rückgabewert ist 0, wenn das Format nicht registriert werden kann.

RegisterWindowMessage (Windows API Funktion)

Deklaration

```
function RegisterWindowMessage(Str: PChar): Word;
```

Beschreibung

Diese Funktion definiert eine neue Fensterbotschaft, die systemweit garantiert eindeutig ist.

Parameter

Str Zeigt auf den Botschaftsstring, der registriert werden soll.

Rückgabewert

Der Rückgabewert ist ein vorzeichenloser short-Integer im Bereich zwischen 0xC000 bis 0xFFFF, wenn die Botschaft erfolgreich registriert wurde. Andernfalls ist er 0.

ReleaseCapture (Windows API Prozedur)

Deklaration

```
procedure ReleaseCapture;
```

Beschreibung

Diese Funktion hebt die Fokussierung auf Mauseingaben auf und stellt die normale Eingabeverarbeitung wieder her.

Siehe auch

SetCapture

ReleaseDC (Windows API Funktion)

Deklaration

```
function ReleaseDC(Wnd: HWnd; DC: HDC): Integer;
```

Beschreibung

Diese Funktion hebt einen Gerätekontext auf und gibt ihn zur Benutzung durch andere Anwendungen frei. Sie gibt nur allgemeine und Fenster-Gerätekontexte frei. Sie hat keinen Einfluß auf Klassen- oder private Gerätekontexte.

Parameter

Wnd	Bezeichnet das Fenster, dessen Gerätekontext aufgehoben werden soll.
DC	Bezeichnet den Gerätekontext, deraufgehoben werden soll.

Rückgabewert

Der Rückgabewert gibt an, ob derGerätekontext aufgehoben wurde. In diesem ist er 1, andernfalls 0.

Siehe auch

GetDC
GetWindowDC

RemoveFontResource (Windows API Funktion)

Deklaration

```
function RemoveFontResourc(FileName: PChar): Bool;
```

Beschreibung

Diese Funktion löscht eine hinzugefügte Schriftressource aus der Datei, die durch den Parameter Filename benannt ist, oder aus der Windows-Schrifttabelle.

Parameter

Filename Zeigt auf einen null-terminierten String, der die Schriftressourcendatei benennt oder ein Handle zu einem geladenenen Modul enthält.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch

DeleteObject
wm_FontChange

RemoveMenu (Windows API Funktion)

Deklaration

```
function RemoveMenu(Menu: HMenu; Position, Flags: Word): Bool;
```

Beschreibung

Diese Funktion löscht einen Menüeintrag mit verbundenem Pop-Up-Menu aus dem Menü, das im Parameter Menu angegeben ist. Dabei zerstört sie aber das Handle für das Pop-Up-Menü nicht und erlaubt somit die Wiederverwendung des Menüs.

Parameter

Menu	Bezeichnet das Menü, das verändert werden soll.
Position	Gibt die zu löschende Menüoption an.
Flags	Gibt an, wie der Parameter Position interperpretiert wird. Er muß ein **mf_xxx-Flags** sein, d.h. mf_ByCommand oder mf_ByPosition sein.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

Siehe auch

DrawMenuBar
GetSubMenu

RemoveProp (Windows API Funktion)

Deklaration

```
function RemoveProp(Wnd: HWnd; Str: PChar): THandle;
```

Beschreibung

Diese Funktion löscht einen Eintrag aus der Eigenschaftsliste des angegebenen Fensters. Der durch den Parameter Str gegebene String bezeichnet den zu löschenden Eintrag.

Parameter

Wnd	Bezeichnet das Fenster, dessen Eigenschaftsliste geändert werden soll.
Str	Zeigt auf einen null-terminierten Zeichenkette oder auf ein Atom.

Rückgabewert

Bezeichnet den angegebenen String. Er ist 0, wenn der String in der angegebenen Eigenschaftsliste nicht gefunden werden kann.

Siehe auch

AddAtom

ReplyMessage (Windows API Prozedur)

Deklaration

```
procedure ReplyMessage(Reply: Longint);
```

Beschreibung

Diese Funktion wird benutzt, um eine mit der Funktion **SendMessage** übergebene Botschaft zu beantworten, ohne die Steuerung an die Funktion zurückzugeben, die SendMessage und **ReplyMessage** aufgerufen hatte.

Parameter

Reply	Bezeichnet das Ergebnis der Verarbeitung der Botschaft. Die möglichen Werte hängen von der tatsächlich gesendeten Botschaft ab.

ResizePalette (Windows API Funktion)

Deklaration

```
function ResizePalette(Palette: HPalette; NumEntries: Word): Bool;
```

Beschreibung

Diese Funktion ändert die Größe der virtuellen Palette, die durch den Parameter Palette bestimmt ist, in die Anzahl von Einträgen, die durch den Parameter NumEntries vorgegeben wird. Wenn die Anwendung ResizePalette aufruft, um sie zu vergrößern, dann werden die zusätzlichen Paletteneinträge auf schwarz gesetzt (die Rot-,Grün- und Blauwerte sind alle 0), ebenso werden die Flags für alle zusätzlichen Einträge auf 0 gesetzt.

Parameter

Palette Bezeichnet die Palette, die verändert werden soll.

NumEntries Bestimmt die Zahl der Einträge in der Palette, nachdem ihre Größe neu festgelegt wurde.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Größe der Palette erfolgreich geändert wurde. Andernfalls ist er 0.

RestoreDC (Windows API Funktion)

Deklaration

```
function RestoreDC(DC: HDC; SaveDC: Integer): Bool;
```

Beschreibung

Diese Funktion stellt den Gerätekontext, der durch den Parameter DC angegeben ist, entsprechend dem vorherigen Status wieder her, der mit dem Parameter SavedDC bezeichnet ist.
Der Kontext-Stack kann Statusinformationen für mehrere Gerätekontexte enthalten. Wenn der durch SavedDC angegebene Kontext nicht oben auf dem Stack abgelegt ist, dann löscht RestoreDC die Statusinformation zwischen dem durch den Parameter SavedDC bezeichneten Gerätekontext und dem oberen Ende des Stacks.

Parameter

DC	Bezeichnet den Gerätekontext.
SavedDC	Gibt den Gerätekontext an, der wiederhergestellt werden soll. Er kann ein Wert sein, der von einem vorangegangenen Aufruf der Funktion SaveDC zurückgegeben wurde. Wenn nSavedDC -1 ist, dann wird der zuletzt gesicherte Gerätekontext wiederhergestellt.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der angegebene Gerätekontext wiederhergestellt wurde. Andernfalls ist er 0.

RoundRect (Windows API Funktion)

Deklaration

```
function RoundRect(DC: HDC; X1, Y1, X2, Y2, X3, Y3: Integer): Bool;
```

Beschreibung

Diese Funktion zeichnet Rechtecke mit abgerundeten Ecken. Das Innere des Rechtecks wird mit dem gewählten Pinsel ausgefüllt; mit dem ausgewählten Stift wird eine Umrandung gezeichnet.

Parameter

DC — Bezeichnet den Gerätekontext.
X1, Y1 — Bestimmt die virtuellen Koordinaten der oberen linken Ecke des Rechtecks.
X2, Y2 — Bestimmt die virtuellen Koordinaten der unteren rechten Ecke des Rechtecks an.
X3, Y3 — Bestimmt die Breite der Ellipse, die benutzt wird, um die abgerundeten Ecken zu zeichnen.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Rechteck gezeichnet wurde. Andernfalls ist er 0.

SaveDC (Windows API Funktion)

Deklaration

```
function SaveDC(DC: HDC): Integer;
```

Beschreibung

Diese Funktion speichert den aktuellen Status desGerätekontexts, der durch den Parameter DC bezeichnet ist, indem sie Statusinformationen (zum Beispiel die Clipping-Region, ausgewählte Objekte und den Abbildungsmodus) auf einen Kontext-Stack kopiert

Parameter

DC	Bezeichnet den Gerätekontext, der gespeichert werden soll.

Rückgabewert

Der Rückgabewert gibt den gespeicherten Gerätekontext an. Er ist 0, wenn ein Fehler auftritt.

Siehe auch

RestoreDC

ScaleViewportExt (Windows API Funktion)

Deklaration

```
function ScaleViewportExt(DC: HDC; Xnum, Xdenom, Ynum, Ydenom: Integer):
Longint;
```

Beschreibung

Diese Funktion verändert die Abmessungen desGrafikfensters relativ zu den aktuellen Werten.

Parameter

DC	Bezeichnet den Gerätekontext
Xnum	Bestimmt den Wert, mit dem die aktuelle x-Abmessung multipliziert werden soll.
Xdenom	Bestimmt den Wert, durch den die aktuelle x-Abmessung dividiert werden soll.
Ynum	Bestimmt den Wert, mit dem die aktuelle y-Abmessung multipliziert werden soll.
Ydenom	Bestimmt den Wert, durch den die aktuelle y-Abmessung dividiert werden soll.

Rückgabewert

Der Rückgabewert gibt die vorherigen Grafikfensterabmessungen in gerätebezogenen Maßeinheiten an. Die vorherige y-Abmessung liegt im höherwertigen, die vorherige x-Abmessung im niederwertigen Word.

ScaleWindowExt (Windows API Funktion)

Deklaration

```
function ScaleWindowExt(DC: HDC; Xnum, Xdenom, Ynum, Ydenom: Integer):
Longint;
```

Beschreibung

Diese Funktion verändert die Abmessungen des Fensters relativ zu den aktuellen Werten.

Parameter

DC	Bezeichnet den Gerätekontext
Xnum	Bestimmt den Wert, mit dem die aktuelle x-Abmessung multipliziert werden soll.
Xdenom	Bestimmt den Wert, durch den die aktuelle x-Abmessung dividiert werden soll.
Ynum	Bestimmt den Wert, mit dem die aktuelle y-Abmessung multipliziert werden soll.
Ydenom	Bestimmt den Wert, durch den die aktuelle y-Abmessung dividiert werden soll.

Rückgabewert

Der Rückgabewert gibt die vorherigen Fensterabmessungen in virtuellen Einheiten an. Die vorherige y-Abmessung liegt im höherwertigen, die vorherige x-Abmessung im niederwertigen Word.

ScreenToClient (Windows API Prozedur)

Deklaration

```
procedure ScreenToClient(Wnd: HWnd; var Point);
```

Beschreibung

Diese Funktion konvertiert die Bildschirmkoordinaten eines gegebenen Punktes auf dem Bildschirm in Client-Koordinaten.

Parameter

Wnd	Bezeichnet das Fenster, dessen Client-Bereich für die Konvertierung benutzt wird.
Point	Zeigt auf eine **TPoint**-Datenstruktur, die die zu konvertierenden Bildschirmkoordinaten enthält.

ScrollDC (Windows API Funktion)

Deklaration

```
function ScrollDC(DC: HDC; dx, dy: Integer; var Scroll, Clip: TRect; UpdateRgn: HRgn;
UpdateRect: LPRect): Bool;
```

Beschreibung

Diese Funktion rollt ein Rechteck aus Bits in horizontaler und vertikaler Richtung. Der Parameter Scroll zeigt auf das Rechteck, das gerollt werden soll, der Parameter dx bezeichnet die Zahl der Einheiten, um die horizontal, der Parameter dy bezeichnet die Zahl der Einheiten, um die vertikal gerollt werden soll.

Parameter

DC	Bezeichnet den Gerätekontext
dx	Gibt die Zahl der horizontalen Bildlaufeinheiten an.
dy	Gibt die Zahl der vertikalen Bildlaufeinheiten an.
Scroll	Zeigt auf die **TRect**TRect-Datenstruktur mit den Koordinaten des zu rollenden Rechtecks.
Clip	Zeigt auf die **TRect**TRect-Datenstruktur mit den Koordinaten des Clipping-Rechtecks.
UpdateRgn	Bezeichnet den Bereich, der vom Bildlauf beeinflußt wird, falls **nil**, wird der neu zu zeichnende Bereich nicht berechnet.
UpdateRect	Zeigt auf die **TRect**TRect-Datenstruktur, die bei der Rückkehr die Koordinaten des Rechtecks enthält, das den Bildlaufbereich umfaßt; falls **nil**, wird der neu zu zeichnende Bereich nicht berechnet.

Rückgabewert

Dieser Wert ist ungleich 0, wenn der Bildlauf ausgeführt wurde. Andernfalls ist er 0.

ScrollWindow (Windows API Prozedur)

Deklaration

```
procedure ScrollWindow(Wnd: HWnd; XAmount, YAmount: Integer; Rect,
ClipRect: LPRect);
```

Beschreibung

Diese Funktion verschiebt ein Fenster, indem sie die Inhalte des Client-Bereichs des Fensters um die Anzahl von Einheiten entlang der x-Achse des Bildschirms verschiebt, die der Parameter XAmount vorgibt und um die Anzahl von Einheiten entlang der y-Achse, die der Parameter YAmount vorgibt. Der Bildlauf erfolgt nach rechts, wenn der Parameter XAmount positiv ist und nach links, wenn er negativ ist. Der Bildlauf erfolgt nach unten, wenn YAmount positiv ist und nach oben, wenn er negativ ist.

Parameter

Wnd	Bezeichnet das Fenster
XAmount	In X-Richtung zu rollende Geräteeinheiten
YAmount	In Y-Richtung zu rollende Geräteeinheiten
Rect	**TRect**-Datenstruktur mit dem zu rollenden Client-Bereich, falls **nil,** wird der gesamte Client-Bereich gerollt.
ClipRect	TRect-Datenstrukturmit dem zu rollenden Clipping-Rechtecks, falls **nil**, wird das gesamte Fenster gerollt.

Siehe auch

UpdateWindow
wm_Paint

SelectClipRgn (Windows API Funktion)

Deklaration

```
function SelectClipRgn(DC: HDC; Rgn: HRgn): Integer;
```

Beschreibung

Diese Funktion wählt die angegebene Region als die aktuelle Clipping-Region für den bezeichneten Gerätekontext. Es wird nur eine Kopie der ausgewählten Region benutzt.

Parameter

DC	Bezeichnet den Gerätekontext
Rgn	Bezeichnet die Region, die ausgewählt wrden soll.

Rückgabewert

Der Rückgabewert gibt den Typ der Region an und enthält eine **Bereichsflag** Konstanten.

SelectObject (Windows API Funktion)

Deklaration

```
function SelectObject(DC: HDC; hObject: THandle): THandle;
```

Beschreibung

Diese Funktion wählt das virtuelle Objekt, das der Parameter hObject bezeichnet, als ausgewähltes Objekt des angegebenen Gerätekontexts. Das neue Objekt ersetzt das vorherige Objekt gleichen Typs.

Parameter

DC Bzeichnet den Gerätekontext.
hObject Bezeichnet das Objekt, das ausgewählt werden soll (Bitmap, Pinsel, Schrift, Stift oder Bereich).

Rückgabewert

Bezeichnet das Objekt, das durch das Objekt ersetzt wurde, das der Parameter hObject vorgibt. Er ist 0, wenn ein Fehler auftritt.

Wenn der Parameter DC eine Metadatei bezeichnet, dann ist der Rückgabewert ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

Siehe auch

DeleteObject
SelectClipRgn
SelectPalette

SelectPalette (Windows API Funktion)

Deklaration

```
function SelectPalette(DC: HDC; Palette: HPalette; ForceBackground: Bool):
HPalette;
```

Beschreibung

Diese Funktion wählt die virtuelle Palette, die der Parameter Palette bezeichnet, als ausgewähltes Palettenobjekt desGerätekontextes aus, den der Parameter DC bezeichnet. Die neue Palette wird das von der GDI bei der Steuerung der imGerätekontext dargestellten Farben benutzte Palettenobjekt und ersetzt die vorherige Palette.

Parameter

DC Bezeichnet den Gerätekontext.

Palette Bezeichnet die virtuelle Palette, die ausgewählt werden soll. .

ForceBackground Gibt an, ob die virtuelle Palette eine Hintergrundpalette sein soll. Wenn ForceBackground ungleich 0 ist, ist die ausgewählte Palette immer eine Hintergrundpalette, ungeachtet dessen, ob das Fenster den Fokus hat. Wenn ForceBackground 0 ist, dann ist die virtuelle Palette eine Vordergrundpalette, wenn das Fenster den Fokus hat.

Rückgabewert

Bezeichnet die virtuelle Palette, die von der Palette ersetzt wurde, die der Parameter Palette vorgibt. Er ist 0, wenn ein Fehler vorliegt.

Siehe auch

CreatePalette

SendDlgItemMessage (Windows API Funktion)

Deklaration

```
function SendDlgItemMessage(Dlg: HWnd; IDDlgItem: Integer; Msg, wParam:
Word; lParam: Longint): Longint;
```

Beschreibung

Diese Funktion sendet eine Botschaft an ein Steuerelement, das der Parameter IDDlgItem bezeichnet, innerhalb des Dialogfensters, das der Parameter Dlg bezeichnet. Die Funktion SendDlgItemMessage kehrt nicht zurück, bis die Botschaft verarbeitet wurde.

Parameter

Dlg	Bezeichnet das Dialogfenster, in dem sich das Steuerelement befindet.
nIDDlgItem	Bestimmt den Integer-Bezeichner des Dialogelements, das die Botschaft empfangen soll.
Msg	Bestimmt den Botschaftswert.
wParam	Bestimmt zusätzliche Botschaftsinformationen.
lParam	Bestimmt zusätzliche Botschaftsinformationen.

Rückgabewert

Der Rückgabewert ist der Wert, der von der Fensterfunktion des Steuerelements zurückgegeben wurde, oder 0, wenn der Bezeichner des Steuerelements nicht gültig ist.

SendMessage (Windows API Funktion)

Deklaration

```
function SendMessage(Wnd: HWnd; Msg, wParam: Word; lParam: Longint):
Longint;
```

Beschreibung

Diese Funktion sendet eine Botschaft an ein oder mehrere Fenster. Die Funktion SendMessage kehrt nicht zurück, bis die Botschaft verarbeitet wurde.

Parameter

Wnd	Bezeichnet das Fenster, das die Botschaft empfangen soll. Wenn der Parameter Wnd 0xFFFF ist, dann wird die Botschaft an alle Pop-Up-Fenster im System übergeben. Die Botschaft wird nicht an untergeordnete Fenster gesendet.
Msg	Bestimmt die zu sendende Botschaft.
wParam	Bestimmt zusätzliche Botschaftsinformationen.
lParam	Bestimmt zusätzliche Botschaftsinformationen.

Rückgabewert

Der Rückgabewert ist der Wert, der von der Fensterfunktion, die die Botschaft empfangen hat, zurückgegeben wird; sein Wert hängt von der gesendeten Botschaft ab.

SetActiveWindow (Windows API Funktion)

Deklaration

```
function SetActiveWindow(Wnd: HWnd): HWnd;
```

Beschreibung

Diese Funktion macht ein Hauptfenster zum aktiven Fenster.

Parameter

Wnd Bezeichnet das zu aktivierende Hauptfenster.

Rückgabewert

Bezeichnet das Fenster, das vorher aktiv war.

SetBitmapBits (Windows API Funktion)

Deklaration

```
function SetBitmapBits(Bitmap: HBitmap; Count: Longint; Bits: Pointer):
Longint;
```

Beschreibung

Diese Funktion setzt die Bits eines Bitmaps auf die Bitwerte, die der Parameter Bits vorgibt.

Parameter

Bitmap	Bezeichnet das Bitmap, das gesetzt werden soll.
Count	Gibt die Zahl der Bytes an, auf die Bits zeigt.
Bits	Zeigt auf die Bitmap-Bits, die in einem Byte-Array gespeichert sind.

Rückgabewert

Der Rückgabewert gibt die Zahl der Bytes an, die beim Setzen der Bitmap-Bits benutzt werden. Er ist 0, wenn die Funktion fehlschlägt.

SetBitmapDimension (Windows API Funktion)

Deklaration

```
function SetBitmapDimension(ABitmap: HBitmap; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion weist einem Bitmap eine Breite und eine Höhe in Einheiten von 0,1 Millimeter zu.

Parameter

Bitmap	Bezeichnet das Bitmap.
X	Bestimmt die Breite des Bitmaps in Einheiten von 0,1 Millimeter.
Y	Bestimmt die Höhe des Bitmaps in Einheiten von 0,1 Millimeter.

Rückgabewert

Der Rückgabewert gibt die vorherigen Abmessungen des Bitmaps an. Die Höhe liegt im höherwertigen, die Breite im niederwertigen Word.

Siehe auch

GetBitmapDimension

SetBkColor (Windows API Funktion)

Deklaration

```
function SetBkColor(DC: HDC; Color: TColorRef): Longint;
```

Beschreibung

Diese Funktion setzt die aktuelle Hintergrundfarbe auf die Farbe, die der Parameter Color vorgibt, oder aber auf die nächste physikalische Farbe, wenn dasGerät den RGB-Farbwert nicht darstellen kann, der von Color vorgegeben wurde.

Parameter

DC	Bezeichnet den Gerätekontext.
Color	**TColorRef** der neuen Hintergrundfarbe.

Rückgabewert

Der Rückgabewert gibt die vorherige Hintergrundfarbe als einen RGB-Farbwert an. Wenn ein Fehler auftritt, dann ist der Rückgabewert 0x80000000.

Siehe auch

BitBlt
SetBkMode
StretchBlt

SetBkMode (Windows API Funktion)

Deklaration

```
function SetBkMode(DC: HDC; BkMode: Integer): Integer;
```

Beschreibung

Diese Funktion setzt den Hintergrundmodus, der für Text und Linien benutzt wird. Der Hintergrundmodus gibt an, ob die GDI vorhandene Hintergrundfarben auf der Bildschirmfläche löschen soll, bevor Text, Pinselschraffur oder irgendein Stift, der keine durchgezogene Linie erzeugt, dargestellt wird.

Parameter

DC	Bezeichnet den Gerätekontext.
BkMode	Gibt den **Hintergrundmodus** an. Er kann die Werte opaque oder Transparent annehmen:

Rückgabewert

Der Rückgabewert gibt den vorherigen Hintergrundmodus an, im Fehlerfall ist er 0.

SetBrushOrg (Windows API Funktion)

Deklaration

```
function SetBrushOrg(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion setzt den Ursprung des aktuell ausgewählten Pinsels in den gegebenen Gerätekontext.

Parameter

DC	Bezeichnet den Geräteekontext.
X, Y	Gibt die Koordinaten des neuen Ursprungs in gerätespezifischen Einheiten an. Der Wert muß im Bereich von 0 bis 7 liegen.

Rückgabewert

Der Rückgabewert bestimmt den Ursprung des Pinsels. Die vorherige x-Koordinate liegt im niederwertigen, die y-Koordinate im höherwertigen Word.

SetCapture (Windows API Funktion)

Deklaration

```
function SetCapture(Wnd: HWnd): HWnd;
```

Beschreibung

Diese Funktion bewirkt, daß alle nachfolgenden Mauseingaben an das Fenster gesendet werden, das der Parameter Wnd bezeichnet, ungeachtet der Zeigerposition.

Parameter

Wnd	Bezeichnet das Fenster, das Mauseingaben empfangen soll.

Rückgabewert

Der Rückgabewert gibt das Fenster an, das vorher den Mauszeiger hatte. Er ist 0, wenn kein solches Fenster existiert.

Siehe auch

ReleaseCapture

SetCaretBlinkTime (Windows API Prozedur)

Deklaration

```
procedure SetCaretBlinkTime(MSeconds: Word);
```

Beschreibung

Diese Funktion setzt die Blinkrate des Carets (die abgelaufene Zeit zwischen dem Aufblinken des Carets) auf die Zahl von Millisekunden, die der Parameter MSeconds bestimmt. Das Caret blinkt auf oder erlischt alle wMSeconds Millisekunden. Das bedeutet, daß ein kompletter Blinkvorgang (AN-AUS-AN) 2 * wMSeconds beansprucht.

Parameter

MSeconds	Gibt die neue Blinkrate in Millisekunden an.

SetCaretPos (Windows API Prozedur)

Deklaration

```
procedure SetCaretPos(X, Y: Integer);
```

Beschreibung

Diese Funktion bewegt das Caret zu der Position, die in virtuellen Koordinaten mit den Parametern X und Y angegeben ist.

Parameter

X, YDie neuen Koordinaten des Carets (in virtuellen Koordinaten).

SetClassLong (Windows API Funktion)

Deklaration

```
function SetClassLong(Wnd: HWnd; Index: Integer; NewLong: Longint):
Longint;
```

Beschreibung

Diese Funktion ersetzt den LongInt-Wert, der vom Parameter Index in der **TWndClass**-Datenstruktur des Fensters bezeichnet ist, das der Parameter Wnd bezeichnet.

Parameter

Wnd	Bezeichnet das Fenster.
Index	Bestimmt den Byte-Offset des Word, das verändert werden soll. Er kann einen der folgenden **gcl_xxx-** Werte annehmen oder einen positiven Byte-Offset enthalten

Rückgabewert

Der Rückgabewert zeigt den vorherigen Wert des angegebenen LongInt an.

SetClassWord (Windows API Funktion)

Deklaration

```
function SetClassWord(Wnd: HWnd; Index: Integer; NewWord: Word): Word;
```

Beschreibung

Diese Funktion ersetzt das vom Parameter Index bezeichnete Word in der **TWndClass** - Datenstruktur des Fensters, das vom Parameter Wnd bezeichnet wird.

Parameter

Wnd	Bezeichnet das Fenster.
Index	Gibt den Byte-Offset des Word an, das verändert soll. Er kann den Wert einer **gcw_xxx**-Konstanten annehmen.

Rückgabewert

Der Rückgabewert gibt den vorherigen Wert des bezeichneten Word an.

SetClipboardData (Windows API Funktion)

Deklaration

```
function SetClipboardData(Format: Word; Mem: THandle): THandle;
```

Beschreibung

Diese Funktion setzt ein Daten-Handle zur Zwischenablage für die Daten, die der Parameter Mem bezeichnet. Es wird vorausgesetzt, daß die Daten das Format haben, das der Parameter Format vorgibt. Nach dem Setzen eines Zwischenablagedaten-Handles gibt die Funktion SetClipboardData den von Mem bezeichneten Speicherblock frei.

Parameter

Format	Gibt ein Datenformat an. Es kann dabei um eines der vordefinierten **cf_xxx Datenformate** handeln.
Mem	Bezeichnet den globalen Speicherblock, der die Daten im angegebenen Format enthält. Der Parameter Mem kann 0 sein, wenn die Anwendung die Daten nicht formatieren und kein Handle auf sie bereitstellen, bis sie durch eine Botschaft **wm_RenderFormat** dazu aufgefordert wird.

Rückgabewert

Bezeichnet die Daten und wird von der Zwischenablage zugewiesen.

SetClipboardViewer (Windows API Funktion)

Deklaration

```
function SetClipboardViewer(Wnd: HWnd): HWnd;
```

Beschreibung

Diese Funktion fügt das Fenster, das durch den Parameter Wnd bestimmt wurde, an die Kette von Fenstern an, die (über die Botschaft **wm_DrawClipboard**) benachrichtigt werden, wann immer die Inhalte der Zwischenablage geändert werden.

Parameter

Wnd	Bezeichnet das Fenster, das Botschaften an die Zwischenablage-Viewer-Kette empfangen soll.

Rückgabewert

Bezeichnet das nächste Fenster in der Kette der Zwischenablage-Viewer.

Siehe auch

ChangeClipboardChain
wm_ChangeCBChain
wm_DrawClipboard
wm_Destroy

SetCommBreak (Windows API Funktion)

Deklaration

```
function SetCommBreak(Cid: Integer): Integer;
```

Beschreibung

Diese Funktion unterbricht die Zeichenübertragung und versetzt die Übertragungsleitung in einen Pausenzustand.

Parameter

Cid	Bezeichnet die Schnittstelle, die unterbrochen werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist negativ, wenn Cid kein gültigesGerät angibt.

Siehe auch

ClearCommBreak
OpenComm

SetCommEventMask (Windows API Funktion)

Deklaration

```
function SetCommEventMask(Cid: Integer; EvtMask: Word): PWord;
```

Beschreibung

Diese Funktion aktiviert die Ereignismaske der vom Parameter Cid angegebenen Schnittstelle und ruft sie ab. Die Bits des Parameters nEvtMask definieren, welche Ereignisse aktiviert werden sollen. Der Rückgabewert zeigt auf den aktuellen Status der Ereignismaske.

Parameter

Cid	Gibt die Schnittstelle an, die aktiviert werden soll. Die Funktion **OpenComm** gibt diesen Wert zurück.
EvtMask	Gibt an, welche Ereignisse aktiviert werden sollen. Er kann jede Kombination der **ev_xxx-Konstanten** Werte sein.

Rückgabewert

Der Rückgabewert zeigt auf eine Integer-Ereignismaske. Jedes Bit in der Ereignismaske gibt an, ob ein gegebenes Ereignis eingetreten ist, oder nicht. Ein Bit ist 1, wenn das Ereignis eingetreten ist.

Siehe auch

OpenComm

SetCommState (Windows API Funktion)

Deklaration

```
function SetCommState(var DCB: TDCB): Integer;
```

Beschreibung

Diese Funktion versetzt eine Schnittstelle in den Status, den der Gerätesteuerblock vorgibt, auf den der Parameter DCB zeigt. Das zu setzende Gerät muß durch das ID-Feld des Steuerblocks bezeichnet werden.

Diese Funktion initialisiert die gesamte Hardware und die Steuerleitungen neu, wie es von DCB definiert ist, aber sie leert weder Sende- noch Empfangsschlangen.

Parameter

DCB	Zeigt auf eine **TDCB**-Datenstruktur, die die gewünschten Schnittstelleneinstellungen für das Gerät enthält.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist negativ, wenn ein Fehler auftritt.

SetCursor (Windows API Funktion)

Deklaration

```
function SetCursor(Cursor: HCursor): HCursor;
```

Beschreibung

Diese Funktion setzt die Zeigerform in die Form, die der Parameter Cursor bestimmt.

Parameter

Cursor	Bezeichnet die Zeiger-Ressource. Die Ressource muß vorher mit der Funktion **LoadCursor** geladen worden sein.

Rückgabewert

Bezeichnet die Zeiger-Ressource, die die vorherige Zeigerform definiert. Er ist 0, wenn es keine vorherige Zeigerform gibt.

SetCursorPos (Windows API Prozedur)

Deklaration

```
procedure SetCursorPos(X, Y: Integer);
```

Beschreibung

Diese Funktion bewegt den Zeiger zu den Bildschirmkoordinaten, die durch die Parameter X und Y bestimmt werden. Wenn die neuen Koordinaten nicht innerhalb des Bildschirmrechtecks liegen, das durch die letzte Funktion **ClipCursor** gesetzt wurde, dann berichtigt Windows automatisch die Koordinaten, so daß der Zeiger innerhalb des Rechtecks bleibt.

Parameter

X Bestimmt die neue x-Koordinate des Zeigers (in Bildschirmkoordinaten).
Y Bestimmt die neue y-Koordinate des Zeigers (in Bildschirmkoordinaten).

SetDIBits (Windows API Funktion)

Deklaration

```
function SetDIBits(DC: HDC; Bitmap: THandle; StartScan, NumScans: Word;
Bits: Pointer; var BitsInfo: TBitmapInfo; Usage: Word): Integer;
```

Beschreibung

Diese Funktion setzt die Bits eines Bitmaps auf die Werte, die in einer geräteunabhängigen Bitmapspezifikation angegeben sind.

Parameter

DC	Bezeichnet den Gerätekontext.
Bitmap	Bezeichnet das Bitmap.
StartScan	Gibt die Nummer der ersten Scan-Zeile im Puffer Bits an.
NumScans	Gibt die Anzahl von Scan-Zeilen im Puffer Bits und die Anzahl von Zeilen an, die in dem vom Parameter Bitmap bezeichneten Bitmap gesetzt werden sollen.
Bits	Zeigt auf die geräteunabhängigen Bitmap-Bits, die als ein Array von Bytes gespeichert sind.
BitsInfo	Zeigt auf eine BITMAPINFO-Datenstruktur, die Information über das DIB enthält.
Usage	Gibt an, ob die Felder bmiColors[ ] des Parameters BitsInfo ausdrückliche RGB-Werte oder Indizes auf die gegenwärtig verwendete virtuelle Palette enthalten. Der Parameter Usage muß einen der folgenden Werte annehmen: DIB_Pal_Colors, wenn die Farbtabelle aus einem Array von 16-Bit-Indizes auf die aktuell verwendete virtuelle Palette besteht, oder DIB_RGB_Colors, wenn die Farbtabelle RGB-Werte enthält (siehe **DIB_ xxx Farbtabellenbezeichner**)

Rückgabewert

Der Rückgabewert gibt die Anzahl von Scan-Zeilen an, die erfolgreich kopiert wurden. Er ist 0, wenn die Funktion fehlschlägt.

SetDIBitsToDevice (Windows API Funktion)

Deklaration

```
function SetDIBitsToDevice(DC: HDC; DestX, DestY, Width, Height, SrcX,
SrcY, StartScan, NumScans: Word; Bits: Pointer; var BitsInfo: TBitmapInfo;
Usage: Word): Integer;
```

Beschreibung

Diese Funktion setzt Bits aus einem geräteunabhängigen Bitmap (DIB) direkt auf eine Bildschirmoberfläche.

Parameter

DC	Bezeichnet den Gerätekontext.
DestX	Bestimmt die x-Koordinate des Ursprungs des Zielrechtecks.
DestY	Bestimmt die y-Koordinate des Ursprungs des Zielrechtecks.
Width	Bestimmt die x-Abmessung des Rechtecks im DIB.
Height	Bestimmt die y-Abmessung des Rechtecks im DIB.
SrcX	Bestimmt die x-Koordinate der Quelle im DIB.
SrcY	Bestimmt die y-Koordinate der Quelle im DIB.
StartScan	Bestimmt die Scan-Zeilennummer des DIB, das in der ersten Scan-Zeile des Puffers Bits enthalten ist.
NumScans	Gibt die Anzahl von Scan-Zeilen des DIB an, die im Puffer Bits enthalten sind.
Bits	Zeigt auf die Bits des DIB, die in einem Array von Bytes abgelegt sind.
BitsInfo	Zeigt auf eine BITMAPINFO-Datenstruktur, die Information über das DIB enthält.
Usage	Gibt an, ob die Felder bmiColors[ ] des Parameters BitsInfo ausdrückliche RGB-Werte oder Indizes auf die gegenwärtig verwendete virtuelle Palette enthalten. Der Parameter Usage muß einen der folgenden Werte annehmen: DIB_Pal_Colors, wenn die Farbtabelle aus einem Array von 16-Bit-Indizes auf die aktuell verwendete virtuelle Palette besteht, oder DIB_RGB_Colors, wenn die Farbtabelle RGB-Werte enthält (siehe **DIB_xxx Farbtabellenbezeichner**)

Rückgabewert

Der Rückgabewert liefert die Zahl der gesetzten Scan-Zeilen.

SetDlgItemInt (Windows API Prozedur)

Deklaration

```
procedure SetDlgItemInt(Dlg: HWnd; IDDlgItem: Integer; Value: Word; Signed: Bool);
```

Beschreibung

Diese Funktion setzt den Text eines Steuerelements in dem angegebenen Dialogfenster auf den String, den der vom Parameter Value gegebene Integerwert repräsentiert.

Parameter

Dlg	Bezeichnet das Dialogfenster, das die Steuerung enthält.
IDDlgItem	Bestimmt das Steuerelement, das verändert werden soll.
Value	Bestimmt den Wert, der gesetzt werden soll.
Signed	Bestimmt, ob der Wert vorzeichenbehaftet ist oder nicht.

Siehe auch

wm_SetText

SetDlgItemText (Windows API Prozedur)

Deklaration

```
procedure SetDlgItemText(Dlg: HWnd; IDDlgItem: Integer; Str: PChar);
```

Beschreibung

Diese Funktion setzt die Überschrift oder den Text eines Steuerelements in das Dialogfenster, das durch den Parameter Dlg bezeichnet ist.

Parameter

Dlg	Bezeichnet das Dialogfenster, welches das Steuerelement enthält.
IDDlgItem	Bestimmt das Steuerelement, dessen Text gesetzt werden soll.
Str	Zeigt auf einen null-terminierten String, der in das Steuerelement kopiert werden soll.

Siehe auch

wm_SetText

SetDoubleClickTime (Windows API Prozedur)

Deklaration

```
procedure SetDoubleClickTime(Count: Word);
```

Beschreibung

Diese Funktion setzt das Doppelklickzeitintervall für die Maus. Ein Doppelklick ist eine Folge von zwei Klicks mit der Maustaste, wobei der zweite innerhalb einer festgelegten Zeitspanne nach dem ersten erfolgt.
Das Doppelklickzeitintervall ist die maximale Zahl von Millisekunden, die zwischen dem ersten und zweiten Klick eines Doppelklicks verstreichen dürfen.

Parameter

Count — Bestimmt die Zahl der Millisekunden, die zwischen den beiden Klicks liegen.

SetEnvironment (Windows API Funktion)

Deklaration

```
function SetEnvironment(PortName, Environ: PChar; Count: Word): Integer;
```

Beschreibung

Diese Funktion kopiert den Inhalt des Puffers, den der Parameter Environ bezeichnet, in die Umgebung, die dem Gerät zugeordnet ist, das an den vom Parameter PortName bezeichneten Anschluß angeschlossen ist.

Parameter

PortName	Zeigt auf einen null-terminierten String, der die Bezeichnung des gewünschten Anschlusses enthält.
Environ	Zeigt auf einen Puffer, der die neue Umgebung enthält.
Count	Bestimmt die Zahl der Bytes, die kopiert werden sollen.

Rückgabewert

Der Rückgabewert gibt die Anzahl der Bytes an, die tatsächlich in die Umgebung kopiert wurden. Er ist 0, wenn ein Fehler auftritt. Er ist -1, wenn die Umgebung gelöscht ist.

SetErrorMode (Windows API Funktion)

Deklaration

```
function SetErrorMode(Mode: Word): Bool;
```

Beschreibung

Diese Funktion steuert, ob Windows Fehler des DOS-Interrupts 24H übernimmt, oder ob dies der aufrufenden Anwendung überlassen wird.

Parameter

Mode	Gibt das Fehlermodus-Flag an. Wenn Bit 0 auf 0 gesetzt ist, stellt Windows ein Fehlermeldungsfenster dar, wenn INT 24H auftritt. Wenn Bit 0 auf 1 gesetzt ist, dann gibt Windows den Fehlerrückruf des INT 21H an die aufrufende Anwendung weiter und stellt kein Meldungsfenster dar.

Rückgabewert

Der Rückgabewert zeigt das vorherige Fehlermodus-Flag an.

SetFocus (Windows API Funktion)

Deklaration

```
function SetFocus(Wnd: HWnd): HWnd;
```

Beschreibung

Diese Funktion weist dem vom Parameter Wnd bezeichneten Fenster den Fokus zu und leitet damit alle nachfolgenden Tastatureingaben an das angegebene Fenster.

Parameter

Wnd	Bezeichnet das Fenster, das die Tastatureingabe empfangen soll.

Rückgabewert

Bezeichnet das Fenster, das vorher den Fokus hatte. Er ist 0, wenn es kein solches Fenster gibt.

SetHandleCount (Windows API Funktion)

Deklaration

```
function SetHandleCount(Number: Word): Word;
```

Beschreibung

Diese Funktion ändert die Anzahl der für eine Task verfügbaren Datei-Handle.

Parameter

Number — Bestimmt die Zahl der Datei-Handle an, die von der Anwendung benötigt werden. Das Maximum ist 255.

Rückgabewert

Der Rückgabewert gibt die Zahl der tatsächlich verfügbaren Datei-Handle für die Anwendung an. Er kann geringer sein als die Zahl, die vom Parameter Number vorgegeben ist.

SetKeyboardState (Windows API Prozedur)

Deklaration

```
procedure SetKeyboardState(var KeyState: Byte);
```

Beschreibung

Diese Funktion kopiert die 256 Bytes, auf die der Parameter KeyState weist, in die Tastaturstatus-Tabelle von Windows.

Parameter

KeyState Zeigt auf ein Array von 256 Bytes, das den Tastaturstatus enthält.

Siehe auch

GetKeyboardState

SetMapMode (Windows API Funktion)

Deklaration

```
function SetMapMode(DC: HDC; MapMode: Integer): Integer;
```

Beschreibung

Diese Funktion stellt den Abbildungsmodus des angegebenen Gerätekontextes ein. Der Abbildungsmodus definiert die Maßeinheit, die bei der Übertragung virtueller Einheiten in gerätebezogene Einheiten benutzt wird und bestimmt die Ausrichtung der x- und y-Achsen desGeräts. Die GDI benutzt den Abbildungsmodus, um virtuelle Koordinaten in passende gerätebezogene Koordinaten umzuwandeln.

Parameter

DC Bezeichnet den Gerätekontext.
MapMode Bestimmt den neuen Abbildungsmodus. Er kann eine **mm_xxx-Konstante Abbildungsmodi** sein.

Rückgabewert

Der Rückgabewert gibt den vorherigen Abbildungsmodus aus.

SetMapperFlags (Windows API Funktion)

Deklaration

```
function SetMapperFlags(DC: HDC; Flag: Longint): Longint;
```

Beschreibung

Diese Funktion wechselt den Algorithmus, den zur Schriftenabbilddung benutzt wird, wenn virtuelle Schriften auf physikalische Schriften abbildet werden..

Parameter

DC	Bezeichnet den Gerätekontext.
Flag	Bestimmt, ob versucht wird, das Höhen- und Seitenverhältnis an das Gerät anzugleichen. Wenn das erste Bit auf 1 gesetzt ist, dann werden nur Schriften ausgewählt, deren x- und y-Abmessungen exakt mit denen des angegebenen Geräts übereinstimmen.

Rückgabewert

Der Rückgabewert gibt den vorherigen Wert des Schriftenabbildungs-Flags an.

SetMenu (Windows API Funktion)

Deklaration

```
function SetMenu(Wnd: HWnd; Menu: HMenu): Bool;
```

Beschreibung

Diese Funktion ändert das Menü des angegebenen Fensters in das Menü, das vom Parameter Menu bezeichnet wird. Wenn Menu 0 ist, dann wird das aktuelle Menü des Fensters gelöscht. Die Funktion SetMenu bewirkt, daß das Fenster neu gezeichnet wird, um die Menüänderung widerzuspiegeln.

Parameter

Wnd	Bezeichnet das Fenster, dessen Menü geändert werden soll.
Menu	Bezeichnet das neue Menü.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Menü geändert wurde. Andernfalls ist er 0.

Siehe auch

DestroyMenu

SetMenuItemBitmaps (Windows API Funktion)

Deklaration

```
function SetMenuItemBitmaps(Menu: HMenu; Position, Flags: Word;
BitmapUnchecked, BitmapChecked: HBitmap): Bool;
```

Beschreibung

Diese Funktion ordnet die angegebenen Bitmaps einer Menüoption zu. Je nachdem, ob die Menüoption markiert ist oder nicht, stellt Windows das entsprechende Bitmap neben der Menüoption dar.

Parameter

Menu — Bezeichnet das Menü, das geändert werden soll.

Position — Gibt die Menüoption an, die verändert werden soll.

Flags — Eine der **mf_xxx-Konstanten**

BitmapUnchecked — Bezeichnet das Bitmap, das dargestellt werden soll, wenn die Menüoption nicht markiert ist.

BitmapChecked — Bezeichnet das Bitmap, das dargestellt werden soll, wenn die Menüoption markiert ist.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

SetMessageQueue (Windows API Funktion)

Deklaration

```
function SetMessageQueue(Msg: Integer): Bool;
```

Beschreibung

Diese Funktion erzeugt eine neue Botschaftsschlange von der mit Msg angegebenen Größe. werden. Die Funktion SetMessageQueue zerstört die alte Warteschlange zusammen mit allen Botschaften, die sie eventuell enthält.

Parameter

Msg	Gibt die maximale Anzahl von Botschaften an, die die neue Schlange enthalten kann.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion eine neue Schlange erstellt. Andernfalls ist er 0.

SetMetaFileBits (Windows API Funktion)

Deklaration

```
function SetMetaFileBits(Mem: THandle): THandle;
```

Beschreibung

Diese Funktion erstellt eine Speicher-Metadatei aus den Daten in dem Speicherblock, den der Parameter Mem bezeichnet.

Parameter

Mem Bezeichnet den globalen Speicherblock, der die Metadatei-Daten enthält. Es wird angenommen, daß die Daten vorher mit der Funktion **GetMetaFileBits** erzeugt wurden.

Rückgabewert

Bezeichnet eine Speicher-Metadatei, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist der Rückgabewert 0.

SetPaletteEntries (Windows API Funktion)

Deklaration

```
function SetPaletteEntries(Palette: HPalette; StartIndex, NumEntries:
Word;var PaletteEntries): Word;
```

Beschreibung

Diese Funktion setzt RGB-Farbwerte und Flags in einer Reihe von Einträgen in einer virtuellen Palette.

Parameter

Palette Bezeichnet die virtuelle Palette.

StartIndex Bestimmt den ersten Eintrag in der virtuellen Palette, der gesetzt werden soll.

NumEntries Bestimmt die Anzahl der Einträge in der virtuellen Palette, die gesetzt werden sollen.

PaletteEntries Zeigt auf das erste Element in einem Array von **TPaletteEntryT**-Datenstrukturen, das RGB-Werte und Flags enthält.

Rückgabewert

Der Rückgabewert ist die Zahl der Einträge, die in der virtuellen Palette gesetzt wurden. Er ist 0, wenn die Funktion fehlschlägt.

SetParent (Windows API Funktion)

Deklaration

```
function SetParent(WndChild, WndNewParent: HWnd): HWnd;
```

Beschreibung

Diese Funktion ändert das übergeordnete Fenster eines untergeordneten Fensters. Wenn das vom Parameter WndChild bezeichnete Fenster sichtbar ist, führt Windows eine entsprechende Neudarstellung durch.

Parameter

WndChild	Bezeichnet das untergeordnete Fenster.
WndNewParent	Bezeichnet das übergeordnete Fenster.

Rückgabewert

Bezeichnet das vorherige übergeordnete Fenster.

SetPixel (Windows API Funktion)

Deklaration

```
function SetPixel(DC: HDC; X, Y: Integer; Color: TColorRef): Longint;
```

Beschreibung

Diese Funktion setzt einen Pixel an dem Punkt, der durch die Parameter X und Y bezeichnet ist.

Parameter

DC Bezeichnet den Gerätekontext.
X, YBestimmt die virtuellen Koordinaten des zu setzenden Punktes.
Color **TColorRef**-Datenstruktur, die die Farbe bezeichnet, in der der Punkt gemalt wird.

Rückgabewert

Der Rückgabewert gibt den RGB-Farbwert für die Farbe an, in der der Punkt tasächlich gezeichnet wurde. Der Wert kann sich von dem durch den Parameter Color vorgegebenen unterscheiden, wenn ein Annäherungswert dieser Farbe benutzt wird. Falls die Funktion fehlschlägt (wenn der Punkt außerhalb der Clipping-Region liegt), dann ist der Rückgabewert -1.

SetPolyFillMode (Windows API Funktion)

Deklaration

```
function SetPolyFillMode(DC: HDC; PolyFillMode: Integer): Integer;
```

Beschreibung

Diese Funktion setzt den Polygonfüllmodus für GDI-Funktionen, die den Polygon-Algorithmus einsetzen, um innen liegende Punkte zu berechnen.

Parameter

DC — Bezeichnet den Gerätekontext.

PolyFillMode — Bestimmt den neuen Füllmodus und ist eine **Polyfüllmodus-Konstante**

Rückgabewert

Der Rückgabewert gibt den vorherigen Füllmodus an. Er ist 0, wenn ein Fehler auftritt.

SetProp (Windows API Funktion)

Deklaration

```
function SetProp(Wnd: HWnd; Str: PChar; Data: THandle): Bool;
```

Beschreibung

Diese Funktion fügt einen neuen Eintrag zur Eigenschaftsliste des angegebenen Fensters hinzu oder verändert einen darin vorhandenen Eintrag.

Parameter

Wnd	Bezeichnet das Fenster, dessen Eigenschaftsliste den neuen Eintrag empfangen soll.
Str	Zeigt auf einen null-terminierten String oder ein Atom, das einen String bezeichnet. Wenn ein Atom gegeben ist, muß es vorher mit der Funktion **AddAtom** erstellt worden sein.
Data	Bezeichnet ein Daten-Handle, das in die Eigenschaftsliste kopiert werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn Daten-Handle und String zur Eigenschaftsliste hinzugefügt wurden. Andernfalls ist er 0.

SetRect (Windows API Prozedur)

Deklaration

```
procedure SetRect(var Rect: TRect; X1, Y1, X2, Y2: Integer);
```

Beschreibung

Diese Funktion erzeugt ein neues Rechteck, indem sie die **TRect**-Datenstruktur, auf die der Parameter Rect weist, ausfüllt.

Parameter

Rect	Zeigt auf die **TRect**-Datenstruktur, die die neuen Rechteckkoordinaten aufnehmen soll.
X1, **Y1**	Bestimmt die Koordinaten der oberen linken Ecke.
X2, Y2	Bestimmt die Koordinaten der unteren rechten Ecke.

SetRectEmpty (Windows API Prozedur)

Deklaration

```
procedure SetRectEmpty(var Rect: TRect);
```

Beschreibung

Diese Funktion erzeugt ein leeres Rechteck (alle Koordinaten gleich 0).

Parameter

Rect	Zeigt auf eine **TRect**-Datenstruktur, diedas leere Rechteck aufnehmen soll.

SetRectRgn (Windows API Prozedur)

Deklaration

```
procedure SetRectRgn(Rgn: HRgn; X1, Y1, X2, Y2: Integer);
```

Beschreibung

Diese Funktion erzeugt eine rechteckige Region. ImGegensatz zu **CreateRectRgn**, ruft sie jedoch nicht den lokalen Speicher-Manager auf. Stattdessen benutzt sie den Speicherplatz, der für die mit dem Parameter Rgn zugeordnete Region reserviert ist. Die Punkte, die mit den Parametern X1, Y1, X2 und Y2 bezeichnet sind, bestimmen den minimalen Umfang des reservierten Speichers.

Parameter

Rgn	Bezeichnet die Region.
X1, Y1	Bestimmt die Koordinaten der oberen linken Ecke der rechteckigen Region.
X2, Y2	Bestimmt die Koordinaten der unteren rechten Ecke der rechteckigen Region.

Siehe auch

CreateRectRgn

SetResourceHandler (Windows API Funktion)

Deklaration

```
function SetResourceHandler(Instance: THandle; ResType: Pointer; LoadFunc: TFarProc): TFarProc;
```

Beschreibung

Diese Funktion bereitet eine Funktion zum Laden von Ressourcen vor. Sie wird intern von Windows benutzt, um berechnete Ressourcen zu implementieren. Der Parameter LoadFunc zeigt auf eine von der Anwendung bereitgestellte Callback-Funktion. Diese Funktion empfängt Informationen über die Ressource (von **FindResource**), die gesperrt werden soll und kann diese Information wie gewünscht weiterverarbeiten. Nachdem sie zurückkehrt, versucht **LockResource** die Ressource noch einmal zu sperren.

Parameter

Instance	Bezeichnet die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.
ResType	Zeigt auf einen Short-Integer, der einen Ressourcentyp bezeichnet.
LoadFunc	Ist die Adresse der Prozedurinstanz der von der Anwendung gestellten Callback- Funktion.

Rückgabewert

Der Rückgabewert zeigt auf eine von der Anwendung bereitgestellte Funktion.

SetROP2 (Windows API Funktion)

Deklaration

```
function SetROP2(DC: HDC; DrawMode: Integer): Integer;
```

Beschreibung

Diese Funktion setzt den aktuellen Zeichenmodus. Die GDI benutzt den Zeichenmodus, um Stifte und die Innenflächen von ausgefüllten Objekten mit den Farben zu kombinieren, die sich schon auf der Bildschirmfläche befinden.

Parameter

DC — Bezeichnet den Gerätekontext.

DrawMode — Bezeichnet den neuen Zeichenmodus, kann eine der **r2_xxx-Konstanten** sein.

Rückgabewert

Bezeichnet den vorherhigen Zeichenmodus.

SetScrollPos (Windows API Funktion)

Deklaration

```
function SetScrollPos(Wnd: HWnd; Bar, Pos: Integer; Redraw: Bool):
Integer;
```

Beschreibung

Diese Funktion setzt die aktuelle Position des Bildlaufleistensymbols auf die durch den Parameter Pos bestimmte und zeichnet die Bildlaufleiste neu (wenn dies vorgegeben ist), um die neue Position wiederzugeben.

Parameter

Wnd	Bezeichnet das Fenster, dessen Bildlaufleiste gesetzt werden soll.
Bar	Gibt die zu setzende Bildlaufleiste an, eine der **sb_-Konstanten**.
Pos	Bestimmt die neue Position. Sie muß innerhalb des Bildlaufbereichs liegen.
Redraw	Bestimmt, ob die Bildlaufleiste neu gezeichnet werden soll, um die neue Position widerzuspiegeln. Wenn der Parameter Redraw ungleich 0 ist, dann wird die Bildlaufleiste neu gezeichnet. Wenn er 0 ist, wird sie nicht neu gezeichnet.

Rückgabewert

Der Rückgabewert bestimmt die vorherige Position der Positionsmarke

SetScrollRange (Windows API Prozedur)

Deklaration

```
procedure SetScrollRange(Wnd: HWnd; Bar, MinPos, MaxPos: Integer; Redraw:
Bool);
```

Beschreibung

Diese Funktion setzt Minimal- und Maxima Positionswerte für die gegebene Bildlaufleiste. Sie kann außerdem dazu eingesetzt werden, Standardbildlaufleisten anzuzeigen oder zu verbergen, indem die Parameter für nMinPos und nMaxPos auf 0 gesetzt werden.

Parameter

Wnd	Bezeichnet ein Fenster oder eine Bildlaufleiste in Abhängigkeit vom Wert des Parameters Bar.
Bar	Bestimmt die zu setzende Bildlaufleiste, ist eine sb_xxx-Konstantesb_ScrollBarConstants.
MinPos	Legt die kleinste Position im Bildlauf fest.
MaxPos	Legt die größte Position im Bildlauf fest.
Redraw	Legt fest, ob die Bildlaufleiste neu gezeichnet werden soll, um die Änderung anzuzeigen. Wenn der Parameter Redraw ungleich 0 ist, wird die Bildlaufleiste neu gezeichnet, andernfalls nicht.

SetSoundNoise (Windows API Funktion)

Deklaration

```
function SetSoundNoise(Source, Duration: Integer): Integer;
```

Beschreibung

Diese Funktion setzt Quelle und Dauer eines Tons am Lautsprecher des Klangausgabegeräts.

Parameter

Source	Legt die Tonquelle fest, ist eine **s_xxx-Konstante**.
Duration	Legt die Dauer des Tons fest (in Clock- Ticks).

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Ist die Quelle ungültig, ist der Rückgabewert s_SerDSR.

Siehe auch

s_xxx-Konstanten

SetStretchBltMode (Windows API Funktion)

Deklaration

```
function SetStretchBltMode(DC: HDC; StretchMode: Integer): Integer;
```

Beschreibung

Diese Funktion setzt den Dehnungsmodus für die **StretchBlt** -Funktion fest. Der Dehnungsmodus legt fest, welche Scan-Zeilen und/oder -Spalten StretchBlt beim Kontrahieren eines Bitmaps entfernt.

Parameter

DC — Legt den Gerätekontext fest.

StretchMode — Legt den neuen Dehnungsmodus fest, ist eine der **StretchBlt Moduskonstanten**.

Rückgabewert

Der Rückgabewert gibt den vorherigen Dehnungsmodus an.

SetSwapAreaSize (Windows API Funktion)

Deklaration

```
function SetSwapAreaSize(Size: Word): Longint;
```

Beschreibung

Diese Funktion erhöht den Speicherumfang, den eine Anwendung für ihre Code-Segmente verwendet. Der maximal verfügbare Speicherumfang ist die Hälfte des freien Speichers, der verbleibt, nachdem Windows geladen ist.

Parameter

Size	Bestimmt die Anzahl der 16-Byte Paragraphen, die eine Anwendung zum Gebrauch als Code-Segment anfordert.

Rückgabewert

Das niederwertige Word des Rückgabewertes gibt die Anzahl der erhaltenen Paragraphen an, die als Speicher für das Code-Segment belegt werden (oder die gegenwärtigen Größe, wenn Size 0 ist); das höherwertige Word gibt die maximal verfügbare Größe an.

SetSysColors (Windows API Prozedur)

Deklaration

```
procedure SetSysColors(Changes: Integer; var SysColor: Integer; var
ColorValues: Longint);
```

Beschreibung

Diese Funktion setzt die Systemfarben für ein oder mehrere Bildschirmelemente.

Parameter

Changes Legt die Anzahl der Systemfarben fest, die geändert werden sollen.

SysColor Zeigt auf ein Array von Integerindexwerten, die die zu ändernden Elemente bestimmen. Die Indexwerte sind **color_xxx-Konstanten**.

ColorValues Zeigt auf ein Array von vorzeichenlosen LongInt-Zahlen, das die neuen RGB--Farbwerte für jedes Element enthält.

SetSysModalWindow (Windows API Funktion)

Deklaration

```
function SetSysModalWindow(Wnd: HWnd): HWnd;
```

Beschreibung

Diese Funktion macht das gewählte Fenster zu einem systemmodalen Fenster.

Parameter

Wnd Bezeichnet das Fenster, das systemmodal gemacht werden soll.

Rückgabewert

Der Rückgabewert gibt das Fenster an, das vorher das systemmodale Fenster war.

SetSystemPaletteUse (Windows API Funktion)

Deklaration

```
function SetSystemPaletteUse(DC: HDC; Usage: Word): Word;
```

Beschreibung

Diese Funktion erlaubt einer Anwendung, deren Fenster gerade fokussiert ist, die volle Systempalette zu verwenden.

Parameter

DC Bezeichnet den Gerätekontext.
Usage Bestimmt den neuen Gebrauch der Systempalette, ist eine **syspal_xxx-Konstante**.

Rückgabewert

Der Rückgabewert gibt den vorhergehenden Gebrauch der Systempalette an.

Siehe auch

GetSysColor
SetSysColors
UnrealizeObject
wm_SysColorChange

SetTextAlign (Windows API Funktion)

Deklaration

```
function SetTextAlign(DC: HDC; Flags: Word): Word;
```

Beschreibung

Diese Funktion setzt die Textausrichtungsflags für den gegebenen Gerätekontext. Die Funktionen **TextOut** und **ExtTextOut** verwenden diese Flags beim Positionieren eines Textstrings auf dem Bildschirm oder auf einemGerät. Die Flags legen das Verhältnis zwischen einem speziellen Punkt und einem Rechteck fest, das den Text umschließt.

Parameter

DC Legt das gewählte Gerät oder den Bildschirm für die Textausgabe fest.
Flags Eine Kombination der **ta_xxx-Konstanten** .

Rückgabewert

Der Rückgabewert gibt die vorherige Einstellung für die Textausrichtung an; das niederwertige Word enthält die horizontale Ausrichtung, das höherwertige die vertikale Ausrichtung.

SetTextCharacterExtra (Windows API Funktion)

Deklaration

```
function SetTextCharacterExtra(DC: HDC; CharExtra: Integer): Integer;
```

Beschreibung

Diese Funktion setzt den Wert für den Zeichenabstand. Die GDI fügt diesen Abstand an jedes Zeichen an, einschließlich der Unterbrechungszeichen, wenn im Gerätekontext eine Textzeile geschrieben wird.

Parameter

DC	Legt den Gerätekontext fest.
CharExtra	Legt die Größe des zusätzlichen Abstandes (in virtuellen Einheiten) fest, der an jedes Zeichen angefügt werden soll.

Rückgabewert

Der Rückgabewert gibt den Wert des vorherigen Zeichenabstands an.

SetTextColor (Windows API Funktion)

Deklaration

```
function SetTextColor(DC: HDC; Color: TColorRef): Longint;
```

Beschreibung

Diese Funktion setzt die Textfarbe auf die Farbe, die durch den Parameter Color bestimmt ist oder auf die nächstliegende physikalische Farbe, wenn dasGerät die durch Color festgelegte Farbe nicht darstellen kann. Die GDI verwendet die Textfarbe, um das Erscheinungsbild jedes Zeichens zu bestimmen, das durch die Funktionen **TextOut** und **ExtTextOut** geschrieben wird. Die GDI verwendet die Textfarbe auch bei der Konvertierung von Bitmaps von Farbe in Monochrom und umgekehrt.

Parameter

DC	Legt den Gerätekontext fest.
Color	Bestimmt die Textfarbe.

Rückgabewert

Der Rückgabewert gibt den RGB-Farbwert der vorherigen Textfarbe an.

Siehe auch

SetBkColor
SetBkMode

SetTextJustification (Windows API Funktion)

Deklaration

```
function SetTextJustification(DC: HDC; BreakExtra, BreakCount: Integer):
Integer;
```

Beschreibung

Diese Funktion bereitet die GDI darauf vor, eine Textzeile zu justieren und dabei die Justierungsparameter zu verwenden, die durch die Parameter BreakExtra und BreakCount festgelegt werden.

Parameter

DC — Bezeichnet den Gerätekontext.

BreakExtra — Legt den Gesamtbetrag des Abstandes (in virtuellen Einheiten) fest, um den die Textzeile ergänzt wird.

BreakCount — Bestimmt die Anzahl der Unterbrechungszeichen in der Zeile.

Rückgabewert

Der Rückgabewert ist 1, wenn die Funktion erfolgreich war, ansonsten ist er 0.

Siehe auch

GetTextExtent
GetTextMetrics
TextOut

SetTimer (Windows API Funktion)

Deklaration

```
function SetTimer(Wnd: HWnd; IDEvent: Integer; Elapse: Word; TimerFunc:
TFarProc): Word;
```

Beschreibung

Diese Funktion erzeugt ein System-Timer-Ereignis. Wenn ein Timer-Ereignis eintritt, gibt Windows eine**wm_Timer**-Botschaft an die anwendungsunterstützte Funktion aus, die durch den Parameter TimerFunc bestimmt ist.

Parameter

Wnd	Legt das Fenster fest, das dem Timer zugeordnet werden soll. Ist Wnd 0, wird dem Timer kein Fenster zugeordnet.
IDEvent	Legt einen von 0 verschiedenen Bezeichner für ein Timer-Ereignis fest, wenn der Parameter Wnd nicht 0 ist.
Elapse	Legt die abgelaufene Zeit (in Millisekunden) zwischen Timer-Ereignissen fest.
TimerFunc	Ist die Adresse der Prozedurinstanz der Funktion, die benachrichtigt werden soll, wenn das Timer-Ereignis stattfindet. Wenn TimerFunc **nil** ist, wird diewm_Timer-Botschaft in der Anwendungsschlange plaziert.

Rückgabewert

Der Rückgabewert legt den Integerbezeichner IDEvent für das neue Timer-Ereignis fest. Ist der Parameter Wnd 0, gibt eine Anwendung diesen Wert an die Funktion **KillTimer**, um das Timer-Ereignis zu beseitigen. Der Rückgabewert ist 0, wenn der Timer nicht eingerichtet wurde.

SetViewportExt (Windows API Funktion)

Deklaration

```
function SetViewportExt(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion setzt die x- und y-Abmessungen desGrafikfensters des festgelegten Gerätekontextes fest. DasGrafikfenster definiert zusammen mit demGerätekontextfenster, wie die GDI Punkte im virtuellen Koordinatensystem auf Punkte im Koordinatensystem des aktuellen Gerätes abbildet.
Die x- und y-Abmessungen desGrafikfensters bestimmen, in welchem Maß die GDI Einheiten im virtuellen Koordinatensystem strecken oder stauchen muß, um sie an die Einheiten imGerätekoordinatensystem anzupassen.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Legt die x-Abmessung desGrafikfensters fest (in Geräteeinheiten).
Y	Legt die y-Abmessung desGrafikfensters fest (in Geräteeinheiten).

Rückgabewert

Der Rückgabewert gibt die vorherigen Abmessungen desGrafikfensters an.
Die vorherige y-Abmessung steht im höherwertigen Word, die vorherige x-Abmessung niederwertigen. Tritt ein Fehler auf, ist der Rückgabewert 0.

SetViewportOrg (Windows API Funktion)

Deklaration

```
function SetViewportOrg(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion bestimmt den Ursprung desGrafikfensters für den festgelegten Gerätekontext. DasGrafikfenster legt, zusammen mit demGerätekontextfenster, fest, wie die GDI Punkte im virtuellen Koordinatensystem auf Punkte im Koordinatensystem des aktuellen Gerätes abbildet.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Legt die x-Koordinate (inGeräteeinheiten) des Ursprungs desGrafikfensters fest. Der Wert muß innerhalb des Bereichs des Gerätekoordinatensystems liegen.
Y	Bestimmt die y-Koordinate (inGeräteeinheiten) des Ursprungs desGrafikfensters. Der Wert muß innerhalb des Bereichs des Gerätekoordinatensystems liegen.

Rückgabewert

Der Rückgabewert gibt den vorherigen Ursprung desGrafikfensters (in Geräteeinheiten) an. Die y-Koordinate steht im höherwertigen, die x-Koordinate im niederwertigen Word.

SetVoiceAccent (Windows API Funktion)

Deklaration

```
function SetVoiceAccent(Voice, Tempo, Volume, Mode, Pitch: Integer):
Integer;
```

Beschreibung

Diese Funktion reiht einen festgelegten Ton (Tempo, Lautstärke, Modus und Tonhöhe) in die Klangwarteschlange ein, die durch den Parameter Voice festgelegt ist.

Parameter

Voice	Legt eine Klangwarteschlage fest. Die erste Schlange wird mit 1 numeriert.
Tempo	Legt die Anzahl von Viertelnoten fest, die pro Minute gespielt werden. Dies kann ein Wert zwischen 32 und 255 sein. Der Standardwert ist 120.
Volume	Legt den Lautstärkepegel fest. Er kann jeden Wert zwischen 0 (kleinste Lautstärke) und 255 (höchste) annehmen.
Mode	Legt fest, wie die Töne gespielt werden sollen, ist eine der **s_xxx-Konstantent**.
Pitch	Legt die Tonhöhe der zu spielenden Töne fest. Dies kann ein Wert zwischen 0 und 83 sein. Der Tonhöhenwert wird unter Verwendung von Modulo-84-Arithmetik zu jedem Tonwert addiert.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Wenn ein Fehler auftritt, ist der Rückgabewert einer der folgenden Werte:

s_SerDMD	Ungültiger Modus
s_SerDTP	Ungültiges Tempo
s_SerDVL	Ungültige Lautstärke
s_SerQFUL	Warteschlange voll

SetVoiceEnvelope (Windows API Funktion)

Deklaration

```
function SetVoiceEnvelope(Voice, Shape, RepeatCount: Integer): Integer;
```

Beschreibung

Diese Funktion reiht die Hüllkurve (Kurvenform und Anzahl der Wiederholungen) in die durch den Parameter Voice festgelegte Klangwarteschlage ein. Die neue Hüllkurve ersetzt die vorherige und bleibt bis zum nächsten Aufruf der Funktion SetVoiceEnvelope wirksam. Eine Hüllkurve wird nicht als Ton gezählt.

Parameter

Voice	Bestimmt die Klangwarteschlage, der die Hüllkurve zugeordnet wird.
Shape	Legt einen Index auf eine OEM-Kurvenformtabelle fest.
Repeat	Legt die Anzahl der Wiederholungen der Kurvenform während der Dauer einer Note fest.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Tritt ein Fehler auf, nimmt der Rückgabewert einen der folgenden Werte an:

s_SerQFUL
s_SerDSH

Siehe auch

s_xxx-Konstanten

SetVoiceNote (Windows API Funktion)

Deklaration

```
function SetVoiceNote(Voice, Value, Length, Cdots: Integer): Integer;
```

Beschreibung

Diese Funktion reiht einen Ton in die durch den Parameter Voice festgelegte Klangwarteschlage ein. .

Parameter

Voice	Legt die Klangwarteschlage fest, der der Ton zugeordnet wird.
Value	Legt eine von 84 möglichen Noten fest (sieben Oktaven). Ist Value 0, wird eine Pause angenommen.
Length	Legt den Kehrwert der Dauer des Tons fest. 1 setzt beispielsweise eine ganze Note fest, 2 eine halbe Note, 4 eine viertel Note, usw.
Cdots	Legt die Dauer des Tons in Punkten fest. Die Dauer ist gleich Length / (Cdots / 3/2).

Rückgabewert

Der Rückgabewert ist 0, wenn erfolgreich ist. Tritt ein Fehler auf, nimmt der Rückgabewert einen der folgenden Werte an:

s_SerDCC Ungültige Punktanzahl.
s_SerDLN Ungültige Tonlänge.
s_SerBDNT Ungültige Note
s_SerQFUL Warteschlange voll

Siehe auch

s_xxx-Konstanten

SetVoiceQueueSize (Windows API Funktion)

Deklaration

```
function SetVoiceQueueSize(Voice, Bytes: Integer): Integer;
```

Beschreibung

Diese Funktion reserviert die Anzahl von Bytes, die durch den Parameter Bytes festgelegt ist, für die durch den Parameter Voice festgelegte Klangwarteschlage. Wenn die Schlangengröße nicht bestimmt wird, ist der Standardwert 192 Bytes, was ungefähr 32 Tönen Platz bietet.

Parameter

Voice	Legt eine Klangwarteschlage fest.
Bytes	Legt die Anzahl von Bytes in der Klangwarteschlage fest.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Tritt ein Fehler auf, nimmt der Rückgabewert einen der folgenden Werte an:

s_SerMACT
s_SerOFM

Siehe auch

s_xxx-Konstanten

SetVoiceSound (Windows API Funktion)

Deklaration

```
function SetVoiceSound(Voice: Longint; Frequency: Longint; Duration:
Integer): Integer;
```

Beschreibung

Diese Funktion stellt Frequenz und Dauer in die Klangwarteschlage ein, die durch den Parameter Voice festgelegt ist.

Parameter

Voice	Legt eine Klangwarteschlage fest. Die erste Klangwarteschlage wird mit 1 numeriert.
Frequency	Legt die Frequenz fest. Das höherwertige Word enthält die Frequenz in Hertz, das niederwertige Word enthält die Teilfrequenz.
Duration	Legt die Dauer des Tons (in Clock-Ticks) fest.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Tritt ein Fehler auf, nimmt der Rückgabewert einen der folgenden Werte an:

s_SerDDR
s_SerDFQ
s_SerDVL
s_SerQFUL

Siehe auch

s_SoundConstants

SetVoiceThreshold (Windows API Funktion)

Deklaration

```
function SetVoiceThreshold(Voice, Notes: Integer): Integer;
```

Beschreibung

Diese Funktion setzt den Schwellenwert für die gegebene Klangwarteschlange. Wenn die Anzahl der in der Warteschlange verbleibenden Töne den Wert unterschreitet, der durch den Parameter Notes festgelegt ist, wird das Schwellenflag gesetzt.

Parameter

Voice	Gibt die zu setzende Klangwarteschlange an.
Notes	Bestimmt die Anzahl der Töne für den Schwellwert.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist 1, wenn die Anzahl der Töne in Notes außerhalb des gültigen Bereichs liegt.

Siehe auch

GetThresholdStatus

SetWindowExt (Windows API Funktion)

Deklaration

```
function SetWindowExt(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion setzt die x- und y-Abmessungen des Fensters, das mit dem vorgegebenen Gerätekontext verbunden ist. Das Fenster legt zusammen mit demGerätekontextfenster fest, wie die GDI Punkte im virtuellen Koordinatensystem auf Punkte imGerätekoordinatensystem abbildet. Die x- und y-Abmessungen des Fensters bestimmen, in welchem Maß die GDI Einheiten im virtuellen Koordinatensystem strecken oder stauchen muß, um die Einheiten demGerätekoordinatensystem anzupassen.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Legt die x-Abmessung (in virtuellen Einheiten) des Fensters fest.
Y	Legt die y-Abmessung (in virtuellen Einheiten) des Fensters fest.

Rückgabewert

Der Rückgabewert gibt die vorherige Ausdehnung des Fensters (in virtuellen Einheiten) an. Die y-Abmessung steht im höherwertigen, die x-Abmessung steht im niederwertigen Word; Tritt ein Fehler auf, ist der Rückgabewert 0.

SetWindowLong (Windows API Funktion)

Deklaration

```
function SetWindowLong(Wnd: HWnd; Index: Integer; NewLong: Longint):
Longint;
```

Beschreibung

Diese Funktion ändert ein Attribut des Fensters, das durch den Parameter Wnd bestimmt ist.

Parameter

Wnd	Bezeichnet das Fenster.
Index	Bestimmt den Byte-Offset des zu ändernden Attributes, ist eine **gwl_xxx-Konstante**.
NewLong	Legt den Austauschwert fest.

Rückgabewert

Der Rückgabewert gibt den vorherigen Wert der genannten long Integerzahl an.

SetWindowOrg (Windows API Funktion)

Deklaration

```
function SetWindowOrg(DC: HDC; X, Y: Integer): Longint;
```

Beschreibung

Diese Funktion legt den Ursprung des Fensters des angegebenen Gerätekontextes fest. Das Fenster legt zusammen mit demGrafikfenster desGerätekontextes fest, wie die GDI Punkte im virtuellen Koordinatensystem auf Punkte imGerätekoordinatensystem abbildet.

Parameter

DC Bezeichnet den Gerätekontext.
X Legt die virtuelle x-Koordinate des neuen Fensterursprungs fest.
Y Legt die virtuelle y-Koordinate des neuen Fensterursprungs fest.

Rückgabewert

Der Rückgabewert gibt den vorherigen Ursprung des Fensters an. Die y-Koordinate steht im höherwertigen, die x-Koordinate im niederwertigen Word.

SetWindowPos (Windows API Prozedur)

Deklaration

```
procedure SetWindowPos(Wnd, WndInsertAfter: HWnd; X, Y, cx, cy: Integer;
Flags: Word);
```

Beschreibung

Diese Funktion ändert Größe, Position und Reihenfolge von untergeordneten Fenstern, Pop-Up- und Hauptfenstern.

Parameter

Wnd	Bestimmt das Fenster, das positioniert wird.
WndInsertAfter	Bezeichnet das Fenster in der Liste des Fenster-Managers, das dem positionierten Fenster vorangeht.
X, Y	Legt die Koordinaten der oberen linken Ecke des Fensters fest.
cx	Legt die neue Breite des Fensters fest.
cy	Legt die neue Höhe des Fensters fest.
Flags	Gibt in einer **swp_xxx-Konstanten flags**die Größenverhältnisse und Positionierung des Fenster an.

SetWindowsHook (Windows API Funktion)

Deklaration

```
function SetWindowsHook(FilterType: Integer; FilterFunc: TFarProc):
TFarProc;
```

Beschreibung

Diese Funktion installiert eine Filterfunktion in der durch FilterType bezeichneten Kette von Filterfunktionen.

Parameter

FilterType — Legt die zu installierende Systemfilterverwaltung fest, ist eine wh_ xxx-Konstante .

FilterFunc — Prozedurinstanzadresse der Filterfunktion

Rückgabewert

Der Rückgabewert zeigt auf die Adresse der Prozedurinstanz des vorher installierten Filters. Er ist **nil**, wenn kein vorheriger Filter existiert.

Siehe auch

DefHookProc

SetWindowText (Windows API Prozedur)

Deklaration

```
procedure SetWindowText(Wnd: HWnd; Str: PChar);
```

Beschreibung

Diese Funktion setzt die Überschrift des gegebenen Fensters (falls eine existiert) auf den String, auf den der Parameter Str zeigt.

Parameter

Wnd — Bezeichnet das Fenster oder das Steuerelement, dessen Text geändert werden soll.

Str — Zeigt auf einen null-terminierten String.

SetWindowWord (Windows API Funktion)

Deklaration

```
function SetWindowWord(Wnd: HWnd; Index: Integer; NewWord: Word): Word;
```

Beschreibung

Diese Funktion ändert ein durch den Parameter Wnd festgelegtes Fensterattribut.

Parameter

Wnd	Bezeichnet das zu modifizierende Fenster.
Index	Bestimmt den Byte-Offset des zu ändernden Word, ist eine **gww_xxx-Konstante**.
NewWord	Legt den Austauschwert fest.

Rückgabewert

Der Rückgabewert gibt den vorherigen Wert des festgelegten Word an.

ShowCaret (Windows API Prozedur)

Deklaration

```
procedure ShowCaret(Wnd: HWnd);
```

Beschreibung

Diese Funktion zeigt das Caret auf dem Bildschirm an der aktuellen Position an. Sobald es sichtbar ist, beginnt das Caret automatisch zu blinken.

Parameter

Wnd	Bezeichnet das Fenster, zu dem das Caret gehört, oder ist 0, um indirekt das Besitzerfenster in der aktuellen Task anzugeben.

ShowCursor (Windows API Funktion)

Deklaration

```
function ShowCursor(Show: Bool): Integer;
```

Beschreibung

Diese Funktion zeigt oder versteckt den Zeiger. Die FunktionShowCursor setzt dazu einen internen Anzeigezähler, der bestimmt, ob der Zeiger gezeigt werden soll. Ist der Parameter Show ungleich 0, wird der Anzeigezählerstand um 1 erhöht. Der Zeiger wird nur angezeigt, wenn der Anzeigezählerstand größer als oder gleich 0 ist.

Parameter

Show	Bestimmt, ob der Anzeigezählerstand erhöht oder verringert wird. Der Anzeigezählerstand wird erhöht, wenn Show ungleich 0 ist. Andernfalls wird er verringert.

Rückgabewert

Der Rückgabewert gibt den neuen Anzeigezählerstand an.

ShowOwnedPopups (Windows API Prozedur)

Deklaration

```
procedure ShowOwnedPopups(Wnd: HWnd; Show: Bool);
```

Beschreibung

Diese Funktion zeigt oder versteckt alle Pop-Up-Fenster, die über den Parameter Wnd bezeichnet werden.

Parameter

Wnd	Bestimmt das Fenster, zu dem die Pop- Up-Fenster gehören, die gezeigt oder versteckt werden sollen.
Show	Legt fest, ob Pop-Up-Fenster versteckt werden oder nicht. Er ist ungleich 0, wenn alle versteckten Pop-Up-Fenster gezeigt werden sollen; er ist 0, wenn alle sichtbaren Pop-Up-Fenster versteckt werden sollen.

ShowScrollBar (Windows API Prozedur)

Deklaration

```
procedure ShowScrollBar(Wnd: HWnd; Bar: Word; Show: Bool);
```

Beschreibung

Diese Funktion zeigt oder versteckt eine Bildlaufleiste, abhängig vom Wert des Parameters Show. Ist Show ungleich 0, wird die Bildlaufleiste angezeigt; ist bShow 0, wird die Bildlaufleiste versteckt.

Parameter

Wnd	Bezeichnet ein Fenster, das in seinem Nicht-Client-Bereich eine Bildlaufleiste enthält, oder eine Bildlaufleiste, wenn der Parameter Bar auf sb_Ctl gesetzt ist.
Bar	Bestimmt, ob die Bildlaufleiste ein Steuerelement oder Teil des Nicht-Client-Bereichs eines Fensters ist, ist eine **sb_xxx-Konstante**
Show	Ist ungleich 0, wenn die Bildlaufleiste angezeigt werden soll, oder 0, wenn die Bildlaufleiste nicht angezeigt werden soll.

ShowWindow (Windows API Funktion)

Deklaration

```
function ShowWindow(Wnd: HWnd; CmdShow: Integer): Bool;
```

Beschreibung

Diese Funktion zeigt oder entfernt das gegebene Fenster, wie durch den Parameter CmdShow festgelegt.

Parameter

Wnd Bezeichnet das Fenster.
CmdShow Bestimmt, wie das Fenster gezeigt werden soll, ist eine **sw_xxx-Konstante** dargestellt sind.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Fenster vorher sichtbar war. Er ist gleich 0, wenn das Fenster vorher versteckt war.

SizeofResource (Windows API Funktion)

Deklaration

```
function SizeofResource(Instance, ResInfo: THandle): Word;
```

Beschreibung

Diese Funktion liefert die Größe der angegebenen Ressource in Bytes.

Parameter

Instance	Bestimmt die Instanz des Moduls, dessen ausführbare Datei die Ressource enthält.
ResInfo	Bestimmt die gewünschte Ressource. Von diesem Handle wird angenommen, daß es durch Anwendung der Funktion **FindResource** erzeugt wurde.

Rückgabewert

Der Rückgabewert gibt die Anzahl der Bytes in der Ressource an. Er ist 0, wenn die Ressource nicht gefunden werden kann.

Siehe auch

AccessResource

StartSound (Windows API Funktion)

Deklaration

```
function StartSound: Integer;
```

Beschreibung

Diese Funktion startet die Wiedergabe in jeder Klangwarteschlage.

Rückgabewert

Obwohl der Rückgabewert vom Typ Integer ist, sollte sein Inhalt ignoriert werden.

StopSound (Windows API Funktion)

Deklaration

```
function StopSound: Integer;
```

Beschreibung

Diese Funktion beendet die Wiedergabe aller Klangwarteschlagen und löscht dann die Inhalte der Schlangen. Der Sound-Treiber jeder Stimme wird abgeschaltet.

Rückgabewert

Obwohl der Rückgabewert vom Typ Integer ist, sollte sein Inhalt ignoriert werden.

StretchBlt (Windows API Funktion)

Deklaration

```
function StretchBlt(DestDC: HDC; X, Y, Width, Height: Integer; SrcDC: HDC;
XSrc, YSrc, SrcWidth, SrcHeight: Integer; Rop: Longint): Bool;
```

Beschreibung

Diese Funktion bewegt ein Bitmap von einem Quellrechteck in ein Zielrechteck und streckt oder staucht das Bitmap, wenn notwendig, um seine Abmessungen dem Zielrechteck anzupassen. Quell- und Zielrechteckt werden gemäß der Angabe Rop kombiniert.

Parameter

DestDC	Bestimmt den Gerätekontext, der das Bitmap erhält.
X	Legt die virtuelle x-Koordinate der oberen linken Ecke des Zielrechtecks fest.
Y	Legt die virtuelle y-Koordinate der oberen linken Ecke des Zielrechtecks fest.
Width	Legt die Breite (in virtuellen Einheiten) des Zielrechtecks fest.
Height	Legt die Höhe (in virtuellen Einheiten) des Zielrechtecks fest.
SrcDC	Bestimmt den Gerätekontext, der das Quell-Bitmap enthält.
XSrc	Legt die virtuelle x-Koordinate der oberen linken Ecke des Quellrechtecks fest.
YSrc	Legt die virtuelle y-Koordinate der oberen linken Ecke des Quellrechtecks fest.
SrcWidth	Bestimmt die Breite (in virtuellen Einheiten) des Quellrechtecks.
SrcHeight	Bestimmt die Höhe (in virtuellen Einheiten) des Quellrechtecks.
Rop	Legt die durchzuführende **Rasteroperation** fest.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Bitmap gezeichnet wurde. Andernfalls ist er 0.

Siehe auch

SetStretchBltMode

SwapMouseButton (Windows API Funktion)

Deklaration

```
function SwapMouseButton(Swap: Bool): Bool;
```

Beschreibung

Diese Funktion tauscht die Bedeutung von linker und rechter Maustaste aus. Ist der Parameter Swap ungleich 0, erzeugt die linke Taste Maussignale der rechten Taste und die rechte Taste Maussignale der linken Taste. .

Parameter

Swap	Bestimmt, ob die Bedeutung der Tasten umgekehrt oder die ursprüngliche Bedeutung der Tasten wiederhergestellt werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion die Bedeutung der Maustasten vertauscht hat. Andernfalls ist er 0.

SyncAllVoices (Windows API Funktion)

Deklaration

```
function SyncAllVoices: Integer;
```

Beschreibung

Diese Funktion reiht eine Synchronisationsmarkierung in jede Schlange ein.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Ist eine Klangwarteschlage voll, dann ist der Rückgabewert s_SerQFUL.

Siehe auch

s_xxx-Konstanten

TabbedTextOut (Windows API Funktion)

Deklaration

```
function TabbedTextOut(DC: HDC; X, Y: Integer; Str: PChar; Count,
TabPositions: Integer; var TabStopPositions; TabOrigin: Integer): Longint;
```

Beschreibung

Diese Funktion schreibt eine Zeichenkette unter Benutzung der aktuell ausgewählten Schrift in die im Feld TabStopPositions angegebenen Spalten auf den angegebenen Bildschirm und erweitert dabei Tabulatorzeichen.

Parameter

DC	Bezeichner des Gerätekontext
X, Y	Startposition des Strings
Str	Zu zeichnender String
Count	Größe (in Zeichen) von Str
TabPositions	Bestimmt die Anzahl der Tabulatorpositionen in dem Array, auf das TabStopPositions zeigt.
TabStopPositions	Zeigt auf ein Array von Integerzahlen, das die Tabulatorpositionen in Pixeln enthält. Die Tabulatorstops müssen in aufsteigender Reihenfolge sortiert werden; Rückwärtstabulatoren sind nicht zulässig.
TabOrigin	Bestimmt die virtuelle x-Koordinate der Startposition, von der aus die Tabulatoren erweitert werden.

Rückgabewert

Wird nicht verwendet

TextOut (Windows API Funktion)

Deklaration

```
function TextOut(DC: HDC; X, Y: Integer; Str: PChar; Count: Integer):
Bool;
```

Beschreibung

Diese Funktion schreibt eine Zeichenkette unter Verwendung der aktuell ausgewählten Schrift auf den angegebenen Bildschirm. Die Startposition des Strings ist durch die Parameter X und Y gegeben.

Parameter

DC	Bezeichnet den Gerätekontext.
X	Bestimmt die virtuelle x-Koordinate des Startpunkts des Strings.
Y	Bestimmt die virtuelle y-Koordinate des Startpunkts des Strings.
Str	Zeigt auf die Zeichenkette, die gezeichnet werden soll.
Count	Bestimmt die Anzahl der Zeichen im String.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der String gezeichnet wurde. Andernfalls ist er 0.

Siehe auch

SetTextAlign

Throw (Windows API Prozedur)

Deklaration

```
procedure Throw(var CatchBuf: TCatchBuf; ThrowBack: Integer);
```

Beschreibung

Diese Funktion stellt die Ausführungsumgebung mit den Werten wieder her, die in dem Puffer gespeichert sind, auf den der Parameter CatchBuf zeigt. Die Ausführung fährt mit der Funktion **Catch** fort, die die Umgebung kopiert hat, auf die CatchBuf zeigt.

Parameter

CatchBuf — Zeigt auf eine **TCatchBuf**-Datenstruktur, die die Ausführungsumgebung enthält.

ThrowBack — Gibt den Wert an, der an die Funktion Catch übergeben werden soll.

TrackPopupMenu (Windows API Funktion)

Deklaration

```
function TrackPopupMenu(Menu: HMenu; Flags: Word; x, y, cx: Integer; Wnd:
HWnd; var Rect: TRect): Bool;
```

Beschreibung

Diese Funktion stellt ein »gleitendes« Pop-Up-Menü an der angegebenen Position dar und stellt die Auswahl der Optionen aus dem Pop-Up-Menü fest. Ein gleitendes Pop-Up-Menü kann irgendwo auf dem Bildschirm erscheinen.

Parameter

Menu	Bezeichnet das Pop-Up-Menü, das dargestellt werden soll.
Flags	Nicht verwendet. Dieser Parameter muß auf 0 gesetzt werden.
x	Bestimmt die horizontale Position der linken Seite des Menüs auf dem Bildschirm in Bildschirmkoordinaten.
y	Bestimmt die vertikale Position der Oberkante des Menüs auf dem Bildschirm in Bildschirmkoordinaten.
Reserved	Ist reserviert und muß auf 0 gesetzt werden.
Wnd	Bezeichnet das Fenster, das das Pop-Up- Menu besitzt. Dieses Fenster empfängt alle **wm_Command**-Botschaften vom Menü.
Rect	TRect-Datenstruktur, die den Bereich definiert, in dem die Maus sichtbar bleibt, wenn der Benutzer die Maustaste losläßt.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

Siehe auch

CreatePopupMenu
GetSubMenu

TranslateAccelerator (Windows API Funktion)

Deklaration

```
function TranslateAccelerator(Wnd: HWnd; AccTable: THandle; var Msg:
TMsg): Integer;
```

Beschreibung

Diese Funktion bearbeitet Tastaturakzeleratoren für Menübefehle. Die Funktion TranslateAccelerator übersetzt die **wm_KeyUp** und **wm_KeyDown**)-Botschaften in **wm_Command** und **wm_SysCommand**-Botschaften, wenn in der Akzeleratortabelle der Anwendung ein Eintrag für die Taste vorhanden ist. Diese Botschaften werden direkt an das Fenster gesendet und nicht in die Anwendungsschlange gestellt.

Parameter

Wnd	Bezeichnet das Fenster, dessen Botschaften übersetzt werden sollen.
AccTable	Bezeichnet eine Akzeleratortabelle, die mittels der Funktion **LoadAccelerators** geladen wurde.
Msg	Zeigt auf eine Botschaft, die mit der Funktion **GetMessage** oder **PeekMessage** abgerufen werden kann. Die Botschaft muß eine **TMsg**-Datenstruktur sein.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Übersetzung stattfindet. Andernfalls ist er 0.

TranslateMDISysAccel (Windows API Funktion)

Deklaration

```
function TranslateMDISysAccel(Wnd: HWnd; var Msg: TMsg): Bool;
```

Beschreibung

Diese Funktion übersetzt Tastaturakzelerator für Systemmenübefehle untergeordneter Fenster von Multidokumentschnittstellen (MDI) in **wm_SysCommand**-Botschaften.

Parameter

WndClient	Bezeichnet das übergeordnete MDI- Client-Fenster.
Msg	Zeigt auf eine Botschaft, die mittels der Funktion **GetMessage** oder **PeekMessage** abgerufen werden kann. Die Botschaft muß eine **TMsg**-Datenstruktur sein und Botschaftsinformationen der Windows-Anwendungsschlange enthalten.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion eine Botschaft in einen Systembefehl übersetzt hat. Andernfalls ist er 0.

TranslateMessage (Windows API Funktion)

Deklaration

```
function TranslateMessage(var Msg: TMsg): Bool;
```

Beschreibung

Diese Funktion übersetzt **wm_KeyDown**/**wm_KeyUp** Kombinationen in **wm_Charoderr** **wm_DeadChar** und **wm_SysKeyDown**/**wm_SysKeyUp** Kombinationen in **wm_SysChar** oder **wm_SysDeadChar** Botschaften
Die Zeichenbotschaften werden in die Anwendungsschlange eingereiht

Parameter

Msg	Zeigt auf eine Datenstruktur **TMsg**, die mit der Funktion **GetMessage** oder **PeekMessage** abgerufen wurde.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Botschaft übersetzt wurde (das bedeutet, Zeichenbotschaften werden in die Anwendungsschlange gestellt). Andernfalls ist er 0.

TransmitCommChar (Windows API Funktion)

Deklaration

```
function TransmitCommChar(Cid: Integer; AChar: Char): Integer;
```

Beschreibung

Diese Funktion markiert das durch den Parameter AChar angegebene Zeichen zur sofortigen Übertragung, indem es ihn an den Anfang der Übertragungsschlange stellt.

Parameter

Cid	Bestimmt das Datenübertragungsgerät, welches das Zeichen empfangen soll.
AChar	Bestimmt das zu übertragende Zeichen.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist negativ, wenn das Zeichen nicht übertragen werden konnte, weil das vorherige Zeichen noch nicht übertragen wurde.

Siehe auch

OpenComm

UngetCommChar (Windows API Funktion)

Deklaration

```
function UngetCommChar(Cid: Integer; AChar: Char): Integer;
```

Beschreibung

Diese Funktion stellt das Zeichen, das durch den Parameter AChar bestimmt ist, in die Empfangsschlange zurück und macht es zu dem Zeichen, das bei einem darauffolgenden Lesevorgang als erstes aus der Schlange gelesen wird.

Parameter

Cid	Bestimmt das Datenübertragungsgerät, welches das Zeichen empfangen soll.
AChar	Bestimmt das Zeichen, das in die Empfangsschlange zurückgestellt werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Er ist negativ, wenn ein Fehler auftritt.

Siehe auch

OpenComm

UnhookWindowsHook (Windows API Funktion)

Deklaration

```
function UnhookWindowsHook(Hook: Integer; HookFunc: TFarProc): Bool;
```

Beschreibung

Diese Funktion entfernt die Windows-Filterverwaltungsfunktion, auf die der Parameter HookFunc zeigt, aus einer Kette von Filterverwaltungsfunktionen.

Parameter

Hook	Gibt den Typ der entfernten Filterverwaltungsfunktion an, ist eine **wh_xxx-Konstante**..
HookFunc	Ist die Adresse der Prozedurinstanz der Filterverwaltungsfunktion.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Filterverwaltungsfunktion erfolgreich entfernt wurde. Andernfalls ist er 0.

UnionRect (Windows API Funktion)

Deklaration

```
function UnionRect(var DestRect, Src1Rect, Src2Rect: LPRect): Integer;
```

Beschreibung

Diese Funktion erzeugt die Vereinigung zweier Rechtecke. Die Vereinigung ist das kleinste Rechteck, das beide Quellrechtecke einschließt.

Parameter

DestRect	Weist auf eine Datenstruktur **TRect**, die die neue Vereinigung aufnehmen soll.
Src1Rect	Weist auf eine Datenstruktur TRect, die ein Quellrechteck enthält.
Src2Rect	Weist auf eine Datenstruktur TRect, die ein Quellrechteck enthält.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Vereinigung nicht leer ist. Er ist 0, wenn die Vereinigung leer ist.

UnlockSegment (Windows API Funktion)

Deklaration

```
function UnlockSegment(Segment: Word): THandle;
```

Beschreibung

Diese Funktion gibt das Segment frei, dessen Segmentadresse durch den Parameter Segment angegeben ist.Parameter

Segment Gibt die Segmentadresse des freizugebenden Segments an. Wenn Segment -1 ist, gibt die Funktion UnlockSegment das aktuelle Datensegment frei.

Rückgabewert

Der Rückgabewert ist 0, wenn der Sperrenzähler des Segments auf 0 heruntergesetzt wurde. Andernfalls ist der Rückgabewert ungleich 0.

Siehe auch

LockSegment

UnrealizeObject (Windows API Funktion)

Deklaration

```
function UnrealizeObject(hObject: HBrush): Bool;
```

Beschreibung

Wenn der Parameter hObject einen Pinsel bezeichnet, weist diese Funktion die GDI an, den Ursprung des gegebenen Pinsels zurückzusetzen, wenn er das nächste Mal ausgewählt wird.

Wenn hObject eine virtuelle Palette bezeichnet, weist diese Funktion die GDI an, eine Palette zu realisieren, als wäre sie vorher noch nicht realisiert worden.

Parameter

hObject Bezeichnet das Objekt, das zurückgesetzt werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

UnregisterClass (Windows API Funktion)

Deklaration

```
function UnregisterClass(ClassName: PChar; Instance: THandle): Bool;
```

Beschreibung

Diese Funktion löscht die Fensterklasse, die durch ClassName bestimmt wird, aus der Fensterklassentabelle und gibt dabei den für die Klasse erforderlichen Speicher frei.

Parameter

ClassName — Zeigt auf einen null-terminierten String, der die Klassenbezeichnung enthält. Diese Klassenbezeichnung muß vorher registriert werden. Vordefinierte Klassen, sowie Dialogfenstersteuerelemente können nicht gelöscht werden.

Instance — Bezeichnet die Instanz des Moduls, das die Klasse erzeugt hat.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion die Fensterklasse erfolgreich aus der Fensterklassentabelle gelöscht hat. Er ist 0, wenn die Klasse nicht gefunden werden konnte oder wenn ein Fenster existiert, das mit dieser Klasse erzeugt wurde.

Siehe auch

RegisterClass

UpdateColors (Windows API Funktion)

Deklaration

```
function UpdateColors(DC: HDC): Integer;
```

Beschreibung

Diese Funktion aktualisiert den Client-Bereich desGerätekontextes, der durch den Parameter DC bezeichnet wird, indem die aktuellen Farben im Client-Bereich pixelweise an die Systempalette angeglichen werden.

Parameter

DC Bezeichnet den Gerätekontext

UpdateWindow (Windows API Prozedur)

Deklaration

```
procedure UpdateWindow(Wnd: HWnd);
```

Beschreibung

Diese Funktion aktualisiert den Client-Bereich des gegebenen Fensters, indem sie eine **wm_Paint**-Botschaft an das Fenster sendet, wenn der Aktualisierungsbereich für das Fenster nicht leer ist. Wenn die Aktualisierungsregion leer ist, wird keine Botschaft gesendet.

Parameter

Wnd	Bezeichnet das zu aktualisierende Fenster.

ValidateFreeSpaces (Windows API Funktion)

Deklaration

```
function ValidateFreeSpaces: Pointer;
```

Beschreibung

Überprüft alle freien Speichersegmente auf gültige Inhalte. Diese Funktion ist nur in der Debugging-Version von Windows verfügbar.

ValidateCodeSegments (Windows API Prozedur)

Deklaration

```
procedure ValidateCodeSegments;
```

Description

Gibt Debugging-Informationen auf ein Terminal aus, wenn Codesegmente durch zufällige Speicherüberschreibungen verändert wurden.

Die Prozedur ist nur in der Debugging Version von Windows verfügbar.

Die Prozedur wird deaktiviert, indem EnableSegmentChecksum im [kernel] Abschnitt von WIN.INI zu 0 gesetzt wird..

Wird im Standard- und 386-Erweiterten Modus von Windows nicht verwendet.

ValidateRect (Windows API Prozedur)

Deklaration

```
procedure ValidateRect(Wnd: HWnd; Rect: LPRect);
```

Beschreibung

Diese Funktion führt eine Gültigkeitsprüfung des Client-Bereichs innerhalb des gegebenen Rechtecks durch, indem sie das Rechteck aus der Aktualisierungsregion des angegebenen Fensters löscht.

Parameter

Wnd	Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die das Rechteck in Client-Koordinaten enthält, das aus der Aktualisierungsregion gelöscht werden soll.

Siehe auch

BeginPaint

ValidateRgn (Windows API Prozedur)

Deklaration

```
procedure ValidateRgn(Wnd: HWnd; Rgn: HRgn);
```

Beschreibung

Diese Funktion führt eine Gültigkeitsprüfung des Client-Bereichs innerhalb der gegebenen Region durch, indem sie die Region aus der Aktualisierungsregion des angegebenen Fensters löscht.
Wenn der Parameter Rgn 0 ist, dann wird das gesamte Fenster auf Gültigkeit überprüft.

Parameter

Wnd	Bezeichnet das Fenster, dessen Aktualisierungsregion verändert werden soll.
Rgn	Bezeichnet eine Region, die einen Bereich definiert, der aus der Aktualisierungsregion gelöscht werden soll.

VkKeyScan (Windows API Funktion)

Deklaration

```
function VkKeyScan(AChar: Word): Word;
```

Beschreibung

Diese Funktion übersetzt ein ANSI-Zeichen in den zugehörigen virtuellen Tastencode und Shift-Status für die aktuelle Tastatur.

Parameter

AChar — Gibt das Zeichen an, für das der zugehörige virtuelle Tastencode gefunden werden soll.

Rückgabewert

Der virtuelle Tastencode wird im niederwertigen Byte zurückgegeben; der Shift-Status im höherwertigen Byte. Möglich sind:

0 Kein Shift.
1 Zeichentaste ist zusammen mit der Shift- Taste gedrückt.
2 Zeichen ist ein Steuerzeichen.
6 Zeichen ist CTRL+ALT.
7 Zeichen ist SHIFT+CTRL+ALT.
3, 4, 5 Eine Shift-Tastenkombination, die nicht für Zeichen benutzt wird.

Wenn keine Taste gefunden wird, deren Übersetzung dem übergebenen ANSI-Code entspricht, dann wird sowohl im höherwertigen, als auch im niederwertigen Byte -1 zurückgegeben.

WaitMessage (Windows API Prozedur)

Deklaration

```
procedure WaitMessage;
```

Beschreibung

Diese Funktion gibt die Programmsteuerung an andere Anwendungen weiter, wenn eine Anwendung keine anderen Aufgaben auszuführen hat. Die Funktion WaitMessage unterbricht die Anwendung und kehrt solange nicht zurück, bis eine neue Meldung in die Anwendungsschlange gestellt wird.

WaitSoundState (Windows API Funktion)

Deklaration

```
function WaitSoundState(State: Integer): Integer;
```

Beschreibung

Diese Funktion wartet, bis der Klangtreiber den angegebenen Zustand erreicht hat.

Parameter

State Gibt den Zustand der Warteschlange des Klangtreibers an, ist eine **s_xxx-Konstante**.

Rückgabewert

Der Rückgabewert ist 0, wenn die Funktion erfolgreich ausgeführt wurde. Wenn der Zustand ungültig ist, ist der Rückgabewert s_SerDST .

Siehe auch

s_xxx-Konstanten

WindowFromPoint (Windows API Funktion)

Deklaration

```
function WindowFromPoint(Point: TPoint): HWnd;
```

Beschreibung

Diese Funktion ermittelt das Fenster, das den vorgegebenen Punkt enthält. Der Parameter Point muß die Bildschirmkoordinaten eines Punktes auf dem Bildschirm bestimmen.

Parameter

Point	Bestimmt eine **TPoint**-Datenstruktur, die den zu untersuchenden Punkt definiert.

Rückgabewert

Bezeichnet das Fenster, in dem der Punkt liegt. Er ist 0, wenn an dem angegebenen Punkt kein Fenster existiert.

WinExec (Windows API Funktion)

Deklaration

```
function WinExec(CmdLine: PChar; CmdShow: Word): Word;
```

Beschreibung

Diese Funktion bringt die durch den Parameter CmdLine bezeichnete Windows- oder Nicht-Windows-Anwendung zur Ausführung.

Parameter

CmdLine	Zeigt auf einen null-terminierten String, die die Befehlszeile für die auszuführende Anwendung enthält (Dateiname und optionale Parameter).
CmdShow	Beschreibt, wie ein Windows-Anwendungsfenster dargestellt werden soll (siehe **ShowWindow**).

Rückgabewert

Wenn die Funktion erfolgreich war, dann ist der Rückgabewert größer als 32. Andernfalls ist es ein Wert kleiner als 32, der den Fehler näher bestimmt. Die folgende Liste beschreibt die Fehlerwerte, die von dieser Funktion zurückgegeben werden:

0	Zu wenig freier Speicher.
2	Datei nicht gefunden.
3	Pfad nicht gefunden.
5	Versuch, eine Task dynamisch einzubinden.
6	Bibliothek erfordert getrennte Datensegmente für jede Task.
10	Falsche Windows-Version.
11	Ungültige .EXE-Datei (Keine Windows-.EXE- Datei oder Fehler im .EXE-Darstellungsformat).
12	OS/2-Anwendung.
13	DOS 4.0-Anwendung.
14	Unbekannter .EXE-Typ.
15	Versuch, im Protected Mode (Standardmodus oder erweiterter 386-Modus) eine für frühere Windows-Versionen erstellte .EXE-Datei zu laden

WinHelp (Windows API Funktion)

Deklaration

```
function WinHelp(Wnd: HWnd; HelpFile: PChar; Command: Word; Data:
Longint): Bool;
```

Beschreibung

Diese Funktion ruft die Windows-Hilfeanwendung auf und übergibt in Command optionale Daten, die die Art der von der Anwendung angeforderten Hilfe festlegen.

Parameter

Wnd	Bezeichnet das Fenster, das die Hilfe anfordert.
HelpFile	Zeigt auf einen null-terminierten String, der, wenn notwendig, den Verzeichnispfad und den Namen der Hilfedatei enthält, die von der Hilfeanwendung angezeigt werden soll.
Command	Gibt die Art der angeforderten Hilfe an, ist eine **help_xxx-Konstante**
Data	Bestimmt den Kontext oder das Schlüsselwort für die angeforderte Hilfe. Wenn Command auf help_Context gesetzt ist, dann ist Data ein vorzeichenloser 32-Bit-Integer, der die ID eines Kontextbezeichners enthält. Wenn Command gleich help_Key ist, dann zeigt Data auf einen null-terminierten String, der ein Schlüsselwort enthält, das Hilfethema bezeichnet.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

WriteComm (Windows API Funktion)

Deklaration

```
function WriteComm(Cid: Integer; Buf: PChar; Size: Integer): Integer;
```

Beschreibung

Diese Funktion schreibt die Anzahl der Zeichen, die im Parameter Size angegeben ist, zu der durch den Parameter Cid bezeichneten Schnittstelle; sie nimmt sie aus dem Datenpuffer, auf den der Parameter Buf weist.

Parameter

Cid	Gibt dasGerät an, das die Zeichen empfangen soll.
Buf	Weist auf den Datenpuffer, der die zu schreibenden Zeichen enthält.
Size	Bestimmt die Zahl der zu schreibenden Zeichen.

Rückgabewert

Der Rückgabewert gibt die Zahl der gerade geschriebenen Zeichen an. Wenn ein Fehler auftritt, wird der Rückgabewert auf einen Wert kleiner als 0 gesetzt, wobei der Absolutbetrag des Rückgabewerts die aktuelle Zahl der geschriebenen Zeichen angibt.

WritePrivateProfileString (Windows API Funktion)

Deklaration

```
function WritePrivateProfileString(ApplicationName, KeyName, Str,
FileName: PChar): Bool;
```

Beschreibung

Diese Funktion kopiert die durch den Parameter Str bezeichnete Zeichenkette in die angegebene Initialisierungsdatei. Sie sucht die Datei nach der Schlüsselanweisung ab, die durch den Parameter KeyName unter dem Programmkopf der Anwendung benannt ist, der wiederum durch den Parameter durch ApplicationName bestimmt ist.
Wenn keine Übereinstimmung besteht, fügt sie dem Benutzerprofil einen neuen Stringeintrag hinzu, der die Bezeichnung der Schlüsselanweisung und den Schlüsselwert enthält, der vom Parameter Str bestimmt wird. Wenn eine übereinstimmende Schlüsselanweisung vorhanden ist, ersetzt die Funktion den Wert dieser Schlüsselanweisung durch Str.

Parameter

ApplicationName	Zeigt auf den Programmkopf einer Anwendung in der Initialisierungsdatei.
KeyName	Zeigt auf eine Schlüsselbezeichnung, die unter dem Programmkopf der Anwendung in der Initialisierungsdatei erscheint, oder ist **nil**, wenn der gesamte Abschnitt mit Schlüsselanweisungen gelöscht werden soll.
Str	Zeigt auf den String, der den neuen Wert der Schlüsselanweisung enthält, oder ist **nil**, wenn die Schlüsselanweisung gelöscht werden soll.
FileName	Zeigt auf einen null-terminiertenString, der die Initialisierungsdatei bezeichnet.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ist. Andernfalls ist er 0.

WriteProfileString (Windows API Funktion)

Deklaration

```
function WriteProfileString(ApplicationName, KeyName, Str: PChar): Bool;
```

Beschreibung

Diese Funktion kopiert die Zeichenkette, auf die der Parameter Str zeigt, in die Windows-Initialisierungsdatei WIN.INI. Sie sucht WIN.INI nach der Schlüsselanweisung ab, die durch den Parameter KeyName unter dem Kopf des Anwendungsprogramms benannt wird. Das Anwendungsprogramm selbst wird durch den Parameter ApplicationName bezeichnet. Wenn keine Übereinstimmung festgestellt wird, dann fügt sie dem Benutzerprofil einen neuen Stringeintrag hinzu, der die Bezeichnung der Schlüsselanweisung und des Schlüsselwertes enthält und der durch den Parameter Str bestimmt wird. Ist keine entsprechende Schlüsselanweisung vorhanden, ersetzt die Funktion den Wert der Schlüsselanweisung durch Str.

Parameter

ApplicationName	Zeigt auf den Kopf eines Anwendungsprogramms in WIN.INI.
KeyName	Zeigt auf die Schlüsselbezeichnung, die unter dem Kopf eines Anwendungsprogrammes in WIN.INI erscheint, oder ist **nil**, wenn die Schlüsselanweisung gelöscht werden soll.
Str	Zeigt auf den String, der den neuen Wert der Schlüsselanweisung enthält, oder ist **nil**, wenn die Schlüsselanweisung gelöscht werden soll.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

wvsprintf (Windows API Funktion)

Deklaration

```
function wvsprintf(Output, Format, ArgList: PChar): Integer;
```

Beschreibung

Diese Funktion formatiert und speichert eine Reihe von Zeichen und Werten in einen Puffer. Jedes Argument wird gemäß der zugehörigen Formatvorgabe in der Formatangabe konvertiert und ausgegeben.

Parameter

Output	Zeigt auf einen null-terminierten String, der die formatierte Ausgabe empfangen soll.
Format	Zeigt auf einen null-terminierten String, der die Formatangabe enthält.
ArgList	Stellt ein oder mehrere optionale Argumente dar. Die Anzahl und der Typ der Argumentparameter hängt von den Formatangabensequenzen in Format ab.

Rückgabewert

Der Rückgabewert gibt die Zahl der in Output gespeicherten Zeichen an, wobei die abschließende 0 nicht gezählt wird. Wenn ein Fehler auftritt, gibt die Funktion einen Wert kleiner als die Länge von Format zurück.

Yield (Windows API Funktion)

Deklaration

```
function Yield: Bool;
```

Beschreibung

Diese Funktion unterbricht die aktuelle Task und startet beliebige wartende Tasks.

AllocDStoCSAlias (Windows API Funktion)

Deklaration

```
function AllocDStoCSAlias(Selector: Word): Word;
```

Beschreibung

Diese Funktion übernimmt den Selektor Selector eines Datensegments und gibt den Selektor eines Codesegments zurück.

Parameter

Selector Bestimmt den Selektor des Datensegments

Rückgabewert

Der Rückgabewert ist der dem Selektor des Datensegments entsprechende Selektor des Codesegments. Kann die Funktion keinen neuen Selektor zuweisen, ist der Rückgabewert 0.

AllocSelector (Windows API Funktion)

Deklaration

```
function AllocSelector(Selector: Word): Word;
```

Beschreibung

Diese Funktion weist einen neuen Selektor zu. Enthält der Parameter Selector einen gültigen Selektor, gibt AllocSelector einen neuen Selektor zurück, der eine genaue Kopie des vorgegebenen darstellt. Ist Selector **nil**, wird ein neuer, nichtinitialisierter Selektor zurückgegeben. Anwendungen sollten diese Funktion nur in Ausnahmefällen verwenden. Die Verwendung dieser Funktion verletzt alle empfehlenswerten Praktiken der Windows-Programmierung.

Parameter

Selector Bestimmt den zu kopierenden Selektor.

Rückgabewert

Der Rückgabewert ist entweder die Kopie eines bereits existierenden, oder ein neuer, nicht-initialisierter Selektor. Konnte die Funktion keinen neuen Selektor zuweisen, ist der Rückgabewert 0.

BeginDeferWindowPos (Windows API Funktion)

Deklaration

```
function BeginDeferWindowPos(NumWindows: Integer): THandle;
```

Beschreibung

Diese Funktion reserviert Speicher, der eine Multi-Fensterposition-Datenstruktur aufnehmen soll und gibt das Handle dieser Struktur zurück.

Parameter

NumWindows	Bestimmt die anfängliche Anzahl von Fenstern, deren Positionsdaten in der Datenstruktur abgelegt werden.

Rückgabewert

Bezeichnet die Multi-Fensterposition-Datenstruktur.

Siehe auch

DeferWindowPos
EndDeferWindowPos

CreatePolyPolygonRgn (Windows API Funktion)

Deklaration

```
function CreatePolyPolygonRgn(var Points; var PolyCounts; Counts,
PolyFillMode: Integer): HRgn;
```

Beschreibung

Diese Funktion erzeugt eine Region, die aus einer Reihe von geschlossenen Polygonen besteht. Die Polygone können sich überschneiden, müssen es aber nicht.

Parameter

Points	Zeigt auf ein Array von **TPoint**-Datenstrukturen, die die Scheitelpunkte der Polygone bestimmen.
PolyCounts	Zeigt auf ein Integer-Array. Jeder Integer bestimmt die Anzahl von Punkten eines der Polygone des Points-Arrays.
Count	Gibt die Gesamtanzahl von Integerzahlen im Array PolyCounts an.
PolyFillMode	Gibt den Füllmodus für die Region an.

Rückgabewert

Bezeichnet der Region, wenn die Funktion erfolgreich ausgeführt wurde. Andernfalls ist er 0.

DeferWindowPos (Windows API Funktion)

Deklaration

```
function DeferWindowPos(WinPosInfo: THandle; Wnd, WndInsertAfter: HWnd; X,
Y, cX, cY: Integer; Flags: Word): THandle;
```

Beschreibung

Diese Funktion aktualisiert die durch den Parameter WinPosInfo angegebene Multi-Fensterposition-Datenstruktur hinsichtlich des in Wnd gegebenen Fensters.

Parameter

WinPosInfo	Bezeichner einer Multi-Fensterposition-Datenstruktur, die Informationen überGröße und Position eines oder mehrerer Fenster enthält.
Wnd	Bezeichnet das Fenster, dessen Aktualisierungsdaten in der Struktur abgelegt werden.
WndInsertAfter	Bezeichnet das Fenster, das dem zu aktualisierenden vorhergeht.
X ,Y	Legt die Koordinaten der oberen linken Fensterecke fest.
cX	Legt die neue Breite des Fensters fest.
cY	Legt die neue Höhe des Fensters fest.
Flags	Enthält eine der **swp_xxx-Flags**.

Rückgabewert

Bezeichnet der aktualisierten Multi-Fensterposition-Datenstruktur.

Siehe auch

BeginDeferWindowPos
EndDeferWindowPos

EndDeferWindowPos (Windows API Prozedur)

Deklaration

```
procedure EndDeferWindowPos(WinPosInfo: THandle);
```

Beschreibung

Diese Funktion aktualisiert gleichzeitig Position undGröße eines oder mehrerer Fenster während einer einzelnen Bildwiederholrate.

Parameter

WinPosInfo	Bezeichnet eine Datenstruktur für Fensterpositionen mit Informationen überGröße und Position eines oder mehrerer Fenster.

Siehe auch

BeginDeferWindowPos
DeferWindowPos

ExitWindows (Windows API Funktion)

Deklaration

```
function ExitWindows(Reserved: DWord; ReturnCode: Word): Bool;
```

Beschreibung

Diese Funktion startet die Standardschließprozedur von Windows. Geben die Anwendungen keine Warnungen zurück, wird die Windows-Sitzung beendet und zu DOS zurückgekehrt.

Parameter

Reserved	Ist reserviert und sollte auf 0 gesetzt werden.
ReturnCode	Enthält den Wert, der an DOS (AL) übergeben wird.f

Rückgabewert

Der Rückgabewert ist 0, wenn eine oder mehrere Anwendungen die Beendigung ablehnen. Stimmen alle Anwendungen zu, kehrt die Funktion nicht zurück.

Siehe auch

wm_QueryEndSession
wm_EndSession

GetBValue (Windows API Funktion)

Deklaration

```
function GetBValue(RGBColor: Longint): Byte;
```

Beschreibung

Dieses Makro ermittelt den blauen Farbanteil eines RGB-Farbwerts.

Parameter

RGBColor — Gibt ein rotes, ein grünes und ein blaues Farbfeld an. Diese Felder bestimmen die Intensität der gegebenen Farbe.

Rückgabewert

Der Rückgabewert ist ein Byte und liegt im Bereich 0 - 255.

GetCodeInfo (Windows API Prozedur)

Deklaration

```
procedure GetCodeInfo(Proc: TFarProc; SegInfo: Pointer);
```

Beschreibung

Diese Funktion ermittelt Informationen über das Code-Segment, das die Funktion enthält, auf die der Parameter Proc zeigt.

Parameter

Proc	Die Adresse der Funktion im Segment, über das Informationen ermittelt werden sollen. Anstelle der Segment:Offset-Adresse kann der Wert auch die Form eines Modul-Handles und einer Segmentnummer haben.
SegInfo	Zeigt auf ein Array aus vier 32-Bit Werten, das mit den Informationen über das Code-Segment gefüllt wird.

GetCurrentPDB (Windows API Funktion)

Deklaration

```
function GetCurrentPDB: Word;
```

Beschreibung

Diese Funktion gibt die Paragraphenadresse oder den Selektor der aktuellen DOS Programmdatenbank zurück, diese wird auch Programmsegmentpräfix (PSP) genannt.

Rückgabewert

Der Rückgabewert ist die Paragraphenadresse oder der Selektor der aktuellen PDB.

GetDOSEnvironment (Windows API Funktion)

Deklaration

```
function GetDOSEnvironment: PChar;
```

Beschreibung

Diese Funktion gibt einen far-Zeiger auf den DOS-Umgebungs-String der laufenden Task zurück.

Rückgabewert

Umgebungsstring der laufenden Task.

GetDoubleClickTime (Windows API Funktion)

Deklaration

```
function GetDoubleClickTime: Word;
```

Beschreibung

Diese Funktion ermittelt den Zeitraum, innerhalb dessen die kurz aufeinanderfolgende, zweimalige Betätigung der Maustaste als Doppelklick interpretiert wird. Dabei muß die zweite Betätigung innerhalb einer festgelegten Zeit erfolgen.

Rückgabewert

Der Rückgabewert gibt die aktuelle Doppelklickzeit in Millisekunden an.

GetGValue (Windows API Funktion)

Deklaration

```
function GetGValue(RGBColor: Longint): Byte;
```

Beschreibung

Dieses Makro ermittelt den grünen Farbanteil eines RGB-Farbwerts.

Parameter

rgbColor	Gibt ein rotes, ein grünes und ein blaues Farbfeld an. Diese Felder bestimmen die Intensität der gegebenen Farbe.

Rückgabewert

Der Rückgabewert ist ein Byte, das den grünen Farbwert des Parameters rgbColor enthält.

GetRgnBox (Windows API Funktion)

Deklaration

```
function GetRgnBox(Rgn: HRgn; var Rect: TRect): Integer;
```

Beschreibung

Diese Funktion ruft die Koordinaten des umgrenzenden Rechtecks der Region ab, die der Parameter Rgn bezeichnet.

Parameter

Rgn	Bezeichnet die Region.
Rect	Zeigt auf eine **TRect**-Datenstruktur, die die Koordinaten des umgrenzenden Rechtecks aufnehmen soll.

Rückgabewert

Der Rückgabewert gibt den Typ der Region an, dh. ist eine der **Regionsflags:**

ComplexRegion
NullRegion
SimpleRegion

oder 0, wenn der Parameter Rgn keine gültige Region angibt.

GetRValue (Windows API Funktion)

Deklaration

```
function GetRValue(RGBColor: Longint): Byte;
```

Beschreibung

Dieses Makro entnimmt einem RGB-Farbwert den Wert für rot.

Parameter

RGBColor — Bezeichnet ein rotes, ein blaues und ein grünes Farbfeld, von denen jedes die Intensität der angegebenen Farbe vorgibt.

Rückgabewert

Ein Byte, das den roten Farbwert enthält.

GlobalFix (Windows API Prozedur)

Deklaration

```
procedure GlobalFix(Mem: THandle);
```

Beschreibung

Diese Funktion verhindert, daß der vom Parameter Mem bezeichnete globale Speicherblock im linearen Speicher verschoben wird.
Der Block wird im linearen Speicher an seiner aktuellen Adresse gesperrt und sein Sperrenzähler wird um 1 erhöht.

Parameter

Mem	Bezeichnet den globalen Speicherblock.

Siehe auch

GlobalUnfix

GlobalPageLock (Windows API Funktion)

Deklaration

```
function GlobalPageLock(Selector: THandle): Word;
```

Beschreibung

Diese Funktion erhöht den Seitensperrenzähler des Speichers, der dem angegebenen globalen Selektor zugeordnet ist. Seitensperroperationen können verschachtelt sein, aber jede Sperroperation muß durch eine zugehörige Operation aufgehoben werden.

Parameter

Selector Gibt den Selektor des Speichers an, der seitenweise gesperrt werden soll.

Rückgabewert

Der Rückgabewert gibt den Wert des Seitensperrenzähler an, nachdem die Funktion ihn erhöht hat. Wenn die Funktion fehlschlägt, ist der Rückgabewert 0.

Siehe auch

GlobalPageUnlock

GlobalPageUnlock (Windows API Funktion)

Deklaration

```
function GlobalPageUnlock(Selector: THandle): Word;
```

Beschreibung

Diese Funktion verringert den Seitensperrenzähler für den Speicherblock, den der Parameter Selector bezeichnet, und ermöglicht, daß der Speicherblock verschoben und auf Platte ausgelagert wird, wenn der Seitensperrenzähler 0 erreicht. Seitensperroperationen können verschachtelt sein, aber jede Seitensperre muß jedoch durch eine entsprechende Operation aufgehoben werden.

Parameter

Selector	Gibt den Selektor des Speichers an, dessen Seitensperrenzähler verringert werden soll.

Rückgabewert

Der Rückgabewert gibt den Seitensperrenzähler an, nachdem die Funktion ihn verringert hat. Wenn die Funktion fehlschlägt, ist der Rückgabewert 0.

Siehe auch

GlobalPageLock

GlobalUnfix (Windows API Funktion)

Deklaration

```
function GlobalUnfix(Mem: THandle): Bool;
```

Beschreibung

Diese Funktion hebt die durch **GlobalFix** vorgenommene Sperre eines globalen Speicherblocks auf, den der Parameter Mem bezeichnet.
GlobalUnfix verringert den Sperrenzähler des Blocks um 1. Der Block kann verschoben oder verworfen werden, wenn der Sperrenzähler bis auf 0 verringert worden ist.

Parameter

Mem Bezeichnet den globalen Speicherblock, der entsperrt werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn der Sperrenzähler des Blocks bis auf 0 verringert wurde. Andernfalls ist der Rückgabewert ungleich 0.

MakeLong (Windows API Funktion)

Deklaration

```
function MakeLong(Low, High: Word): Longint;
```

Beschreibung

Dieses Makro erzeugt einen vorzeichenlosen 32-Bit-Integer durch die Verkettung zweier Integer, die durch die Parameter Low und High bezeichnet sind.

Parameter

Low	Gibt das niederwertige Word des neuen LongInt an.
High	Gibt das höherwertige Word des neuen LongInt an.

Rückgabewert

Der Rückgabewert gibt einen vorzeichenlosen LongInt an.

MulDiv (Windows API Funktion)

Deklaration

```
function MulDiv(Number, Numerator, Denominator: Integer): Integer;
```

Beschreibung

Diese Funktion multipliziert zwei Integerwerte und dividiert das Ergebnis durch einen dritten. Der Rückgabwert ist das Endergebnis, gerundet auf den näherliegenden Integer.

Parameter

Number Gibt die Zahl an, die mit Numerator (Zähler) multipliziert werden soll.
Numerator Gibt die Zahl an, die mit nNumber multipliziert werden soll.
Denominator Gibt die Zahl an, durch die das Resultat der Multiplikation dividiert werden soll.

Rückgabewert

Der Rückgabewert ist das Ergebnis der Multiplikation und der Division. Der Rückgabewert ist 32767 oder -32767, wenn entweder ein Überlauf auftritt oder Denominator auf 0 gesetzt war.

NetBIOSCall (Windows API Prozedur)

Deklaration

```
procedure NetBIOSCall;
```

Beschreibung

Ruft den NETBIOS-Interrupt 5Ch auf. Eine Anwendung sollte diese Prozedur verwenden, anstatt einen NETBIOS-Interrupt 5CH direkt einzuleiten, um die Kompabilität zu künftigen Microsoft-Produkten sicherzustellen.

Diese Prozedur kann nur innerhalb eines Assemblerteils aufgerufen werden, da die CPU-Register vor dem Aufruf gesetzt werden müssen.

PaletteRGB (Windows API Funktion)

Deklaration

```
function PaletteRGB(Red,Green, Blue: Byte): Longint;
```

Beschreibung

Dieses Makro nimmt drei Werte entgegen, die die relativen Intensitäten von Rot,Grün und Blau darstellen und gibt einen Wert zurück, der im höherwertigen Byte aus einer 2 und in den drei niederwertigen Bytes aus einem RGB-Wert besteht. Dies wird palettenbezogener RGB-Bezeichner genannt. Bei der Benutzung einer Farbpalette kann eine Anwendung diesen Bezeichner anstelle eines expliziten RGB-Werts an Funktionen übergeben, die eine Farbe anfordern.

Für Ausgabegeräte, die virtuelle Paletten unterstützen, gleicht Windows einen palettenbezogenen RGB-Wert an den nächsten Farbwert in der virtuellen Palette desGerätekontexts an, als ob die Anwendung einen Index zu diesem Paletteneintrag angegeben hätte. Wenn ein Ausgabegerät keine Systempalette unterstützt, dann benutzt Windows den palettenbezogenen RGB-Wert, als wäre er ein vom Makro RGB zurückgegebener konventioneller RGB DWORD.

Parameter

Red	Bestimmt die Intensität des roten Farbfeldes.
Green	Bestimmt die Intensität des grünen Farbfeldes.
Blue	Bestimmt die Intensität des blauen Farbfeldes.

Rückgabewert

Der Rückgabewert gibt einen palettenbezogenen RGB-Wert an.

RectInRegion (Windows API Funktion)

Deklaration

```
function RectInRegion(Region: HRgn; var Rect: TRect): Bool;
```

Beschreibung

Diese Funktion bestimmt, ob ein Teil des Rechtecks, das durch den Parameter Rect vorgegeben ist, innerhalb derGrenzen einer Region liegt, die durch den Parameter hRegion bezeichnet ist.

Parameter

Region	Bezeichnet die Region.
Rect	Bezeichnet das Rechteck.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn ein Teil des angegebenen Rechtecks innerhalb derGrenzen der Region liegt. Andernfalls ist der Rückgabewert 0.

RGB (Windows API Funktion)

Deklaration

```
function RGB(Red,Green, Blue: Byte): Longint;
```

Beschreibung

Diese Funktion wählt aufgrund der gegebenen Parameter und der Farbwiedergabemöglichkeiten des Ausgabegeräts eine RGB-Farbe aus.

Parameter

Red	Bestimmt die Intensität des roten Farbfeldes.
Green	Bestimmt die Intensität des grünen Farbfeldes.
Blue	Bestimmt die Intensität des blauen Farbfeldes.

Rückgabewert

Bezeichnet die resultierende RGB-Farbe.

StretchDIBits (Windows API Funktion)

Deklaration

```
function StretchDIBits(DC: HDC; DestX, DestY, DestWidth, DestHeight, SrcX,
SrcY, SrcWidth, SrcHeight: Word; Bits: Pointer; var BitsInfo: TBitmapInfo;
Usage: Word; Rop: DWord): Integer;
```

Beschreibung

Diese Funktion bewegt ein geräteunbhängiges Bitmap (»device-independent bitmap«, DIB) aus einem Quellrechteck in ein Zielrechteck und streckt oder staucht das Bitmap, falls erforderlich, um es den Abmessungen des Zielrechtecks anzupassen.

Die durch den Parameter Rop festgelegte Rasteroperation bestimmt, wie das Quell-Bitmap und die gegebenen Bits des Zielgerätes kombiniert werden.

Parameter

DC	Legt den Zielgerätekontext für eine Bildschirmoberfläche oder ein Speicher-Bitmap fest.
DestX, DestY	Bestimmt die Koordinaten (in virtuellen Einheiten) des Ursprungs des Zielrechtecks.
DestWidth	Bestimmt die Breite (in virtuellen Einheiten) des Zielrechtecks.
DestHeight	Bestimmt die Höhe (in virtuellen Einheiten) des Zielrechtecks.
SrcX, SrcY	Bestimmt die Koordinaten (in Pixel) der Quelle im DIB.
SrcWidth	Bestimmt die Breite (in Pixel) des Quellrechtecks im DIB.
SrcHeight	Bestimmt die Höhe (in Pixel) des Quellrechtecks im DIB.
Bits	Zeigt auf die Bits des DIB, die in einem Array von Bytes abgelegt sind.
BitsInfo	Zeigt auf eine BITMAPINFO-Datenstruktur, die Information über das DIB enthält.
Usage	Gibt an, ob die Felder bmiColors[ ] des Parameters BitsInfo ausdrückliche RGB-Werte oder Indizes auf die gegenwärtig verwendete virtuelle Palette enthalten. Der Parameter Usage muß einen der folgenden Werte annehmen: DIB_Pal_Colors, wenn die Farbtabelle aus einem Array von 16-Bit-Indizes auf die aktuell verwendete virtuelle Palette besteht, oder DIB_RGB_Colors, wenn die Farbtabelle RGB-Werte enthält (siehe **DIB_xxx Farbtabellenbezeichner**)
Rop	Legt die durchzuführende **Rasteroperation** fest.

Rückgabewert

Der Rückgabewert ist die Anzahl der kopierten Scan-Zeilen.

SwapRecording (Windows API procedure)

Deklaration

```
procedure SwapRecording(Flag: Word);
```

Beschreibung

Wird mit dem Microsoft Windows Swap gearbeitet, startet oder beendet diese Prozedur die Analyse des Auslagerungsverhaltens..

Parameter

Flag	(0) beendet die Analyse, (1)zeichnet Auslagerungsaufrufe und -rückgaben auf, (2) wie 1, zusätzlich werden Aufrufe über Thrunks aufgezeichnet.

SwitchStackBack (Windows API Prozedur)

Deklaration

```
procedure SwitchStackBack;
```

Beschreibung

Diese Funktion gibt den Stack der aktuellen Task an das Datensegment der Task zurück, nachdem er vorher durch die Funktion SwitchTasksBack neu adressiert wurde.

Siehe auch

SwitchStackTo

SwitchStackTo (Windows API Prozedur)

Deklaration

```
procedure SwitchStackTo(StackSegment, StackPointer, StackTop: Word);
```

Beschreibung

Diese Funktion ändert den Stack der aktuellen Task auf das durch den Parameter StackSegment festgelegte Segment.
Dynamische Linkbibliotheken (DLL) haben keinen Stack; stattdessen verwendet eine DLL den Stack der Task, die die Bibliothek aufruft. Infolgedessen werden DLL-Funktionen fehlschlagen, die voraussetzen, daß die Inhalte des Code-Segment-Registers (CS) und des Stack-Segment-Registers identisch sind. Die Funktion SwitchStackTo adressiert den Stack der Task auf das Datensegment einer DLL um und läßt zu, daß die DLL diese Funktionen aufruft. SwitchStackTo kopiert die Argumente der Task auf dem Stack auf die neue Position des Stacks.

Parameter

StackSegment	Bestimmt das Datensegment, das den Stack enthalten soll.
StackPointer	Legt den Offset des Stack-Anfangs im Datensegment fest.
StackTop	Legt den Offset der Stack-Spitze relativ zum Stack-Anfang fest.

Siehe auch

SwitchStackBack

ToAscii (Windows API Funktion)

Deklaration

```
function ToAscii(VirtKey, ScanCode: Word; KeyState: PChar; CharBuff:
Pointer; Flags: Word): Integer;
```

Beschreibung

Diese Funktion übersetzt den virtuellen Tastencode, der durch den Parameter VirtKey bestimmt ist und den aktuellen Tastaturstatus, der durch den Parameter KeyState bestimmt ist in das oder die zugehörige(n) ANSI-Zeichen.

Parameter

VirtKey	Bestimmt den virtuellen Tastencode, der übersetzt werden soll.
ScanCode	Bestimmt den ursprünglichen Hardware- Scan-Code der Taste, der übersetzt werden soll. Das höchstwertige Bit dieses Wertes wird gesetzt, wenn die Taste nicht gedrückt ist.
KeyState	Zeigt auf ein Array von 256 Bytes, von denen jedes den Status einer Taste enthält. Wenn das höchstwertige Bit des Bytes gesetzt ist, dann ist die Taste nicht gedrückt.
Char	Zeigt auf einen 32-Bit-Puffer, der das oder die übersetzten ANSI-Zeichen empfängt.
Flags	Das Bit 0 dient als Flag für die Menüanzeige.

Rückgabewert

Der Rückgabewert gibt die Zahl der Zeichen an, die in den Puffer kopiert wurden, auf den der Parameter Char zeigt. Der Rückgabewert ist negativ, wenn die Taste eine tote Taste war. Andernfalls ist er einer der folgenden Werte:

2 Es wurden zwei Zeichen in den Puffer kopiert. Das sind gewöhnlich ein Akzent und das Zeichen einer tote Taste, falls die tote Taste nicht anders übersetzt werden kann.

1 Es wurde ein ANSI-Zeichen in den Puffer kopiert.

0 Die angegebene virtuelle Taste hat für den aktuellen Tastaturstatus keine Übersetzung.

UnlockData (Windows API Funktion)

Deklaration

```
function UnlockData(Dummy: Integer): THandle;
```

Beschreibung

Dieses Makro gibt das aktuelle Datensegment frei. Es ist für die Verwendung von Modulen vorgesehen, die verschiebbare Datensegmente haben.

Parameter

Dummy Wird nicht verwendet; kann auf 0 gesetzt werden.

UnlockResource (Windows API Funktion)

Deklaration

```
function UnlockResource(RezData: THandle): Bool;
```

Beschreibung

Diese Makro entsperrt die durch den Parameter ResData angegebene Ressource und verringert den Referenzzähler der Ressource um Eins.

Parameter

ResData Bezeichnet den globalen Speicherblock, der freigegeben werden soll.

Rückgabewert

Der Rückgabewert ist 0, wenn der Referenzzähler des Blocks bis auf 0 heruntergesetzt wurde. Andernfalls ist er ungleich 0.

Index der Compiler-Befehle

Compiler-Befehle
- **Schalterbefehle**
  - **$A (Ausrichtung der Variablen)**
  - **$B (Auswertung boolescher Ausdrücke)**
  - **$D (Debugger-Information)**
  - **$F (Erzwingen von FAR-Aufrufen)**
  - **$G (Codeerzeugung für 80286)**
  - **$I (Automatische Prüfung von Ein-/Ausgaben)**
  - **$L (Lokale Symbole)**
  - **$N (Numerische Operationen des 80x87)**
  - **$R (Bereichsüberprüfung)**
  - **$S (Stack-Prüfung)**
  - **$V (Überprüfung von Var-Strings)**
  - **$W (Windows Stack Frame)**
  - **$X (Erweiterte Syntax)**
- **Parameterbefehle**
  - **$Cxxx (Code Segment Attribute)**
  - **$Ixxx (Include-Datei)**
  - **$Lxxx (Einbinden von Object-Dateien)**
  - **$Mxxx (Speicherzuweisungsgröße)**
- **Befehle zur bedingten Compilierung**
  - **DEFINE**
  - **ELSE**
  - **ENDIF**
  - **IFDEF**
  - **IFNDEF**
  - **IFOPT**

Bedingte Compilierung

Befehle und IDE-Äquivalente

Index der Turbo Pascal Standard-Units

Eine Unit ist eine Zusammenstellung von Konstanten, Datentypen, Variablen, Prozeduren und Funktionen.
Turbo Pascal enthält folgende Standard-Units:

Strings
System
WinCrt
WinDOS
WinProcs
WinTypes

Diese Standard-Units unterstützen Ihre Turbo-Pascal-Programme, sie sind alle in TPW.TPL gespeichert.

Sprachumfang von Turbo Pascal

Anweisungen
Ausdrücke
Bezeichner
Funktionen
Gültigkeitsbereichsregeln
Kommentare
Konstantendeklarationen
Labels
Operatoren
Prozeduren
Reservierte Wörter
Spezielle Symbole
Typisierte Konstanten
Typdeklarationen
Units
Variablen
Zahlen
Zeichenketten

Turbo-Pascal-Typen

Wenn Sie ein Programm schreiben, arbeiten Sie mit Informationen, die im allgemeinen einem der fünf Grundtypen angehören: Integer-Werte, Realzahlen, Zeichen und Zeichenketten, boolesche Werte und Zeiger.

Integerwerte sind ganze Zahlen (z.B. 1, 5, -21 und 752).

Realzahlen haben Bruchteile (3.14159) und Exponenten (2.579x1024). Man nennt diese Zahlen auch Gleitkommazahlen.

Zeichen sind die Buchstaben des Alphabets, Symbole und die Zahlen 0-9. Sie können einzeln verwendet werden (`a, Z, !, 3`) oder kombiniert als Zeichenkette (`'This is only a test.'`).

Boolesche Ausdrücke können einen von zwei möglichen Werten annehmen: True oder False. Sie werden in konditionalen Ausdrücken verwendet.

Zeiger speichern die Adresse eines Speicherbereichs, in dem wiederum Daten gespeichert sind.

Siehe auch

Typdeklarationen

Typdeklarationen

Wenn Sie eine Variable deklarieren, müssen Sie ihren **Typ** festlegen.

Der Typ legt den Wertebereich der Variablen fest und bestimmt die Operationen, die mit ihr ausgeführt werden können. Eine Typdeklaration vereinbart also einen **Bezeichner, der seinerseits für einen bestimmten Typ steht.**

Ein Bezeichner auf der linken Seite einer Typdeklaration wird für den Block, in dem seine Definition stattfindet, als Typbezeichner vereinbart.

Der Gültigkeitsbereich eines Typbezeichners erstreckt sich nicht auf die Definition selbst (d.h. die Definition kann den zukünftig definierten Typ nicht selbst enthalten). Die einzige Ausnahme von dieser Regel stellen Zeigertypen dar.

Es gibt sechs Klassen von Typen:

1. **Einfache Typen** definieren angeordnete Wertemengen.
   - **Ordinale Types**
     - **Integer Typen**
     - **Boolsche Typen**
     - **Char-Typen**
     - **Aufzählbare Typen**
     - **Teilbereichstypen**
   - **Real-Typen**
2. **String-Typen** sind Zeichenfolgen mit dynamischer Länge, denen ein Speicherbereich konstanter Größe zubgewiesen wird.
3. **Strukturierte Typen** enthalten mehr als einen Wert.
   - **Array-Typen**
   - **Record-Typen**
   - **Objekttypen**
   - **Mengentypen**
   - **Dateitypen**
4. **Zeigertypen** definieren einen Satz von Werten, die auf dynamische Variablen eines festgelegten Typs (Grundtyps) zeigen.
5. **Prozedurtypen** ermöglichen, Prozeduren und Funktionen als Objekte zu behandeln.
6. **Objekttypen** sind Strukturen, die aus einer bestimmten Anzahl von Komponenten bestehen.

Prozedurtypen

Standard-Pascal betrachtet Prozeduren und Funktionen streng als Programmteile, die ausschließlich über Prozedur- oder Funktionsaufrufe ausgeführt werden können.

Dagegen ermöglicht es Turbo Pascal mit Hilfe der Prozedurtypen, Prozeduren und Funktionen wie Objekte zu behandeln, die **Variablen** zugewiesen und als **Parameter** übergeben werden können.

Die Deklaration eines Prozedurtyps gibt die Parameter und, bei einer Funktion, den Ergebnistyp an.

Die Syntax einer Prozedurtyp-Deklaration ist dieselbe wie die eines Prozedur- oder Funktionskopfs, nur daß dabei der Bezeichner hinter **procedure** oder **function** weggelassen wird.

Beispiele

```
type
  Proc = procedure;
  SwapProc = procedure(var X, Y: Integer);
  StrProc = procedure(S: string);
  MathFunc = function(X: Real): Real;
  DeviceFunc = function(var F: text): Integer;
  MaxFunc = function(A, B: Real; F: MathFunc): Real;
```

Die Namen der Parameter in einer Prozedurtyp-Deklaration haben keinerlei Bedeutung.

Funktionen, die einen Prozedurtyp zurückgeben, können in Turbo Pascal nicht deklariert werden.

Zulässige Typen für Funktionswerte sind **String-Typen**, **Real**, **Integer**, **Char**, **Boolean**, **Pointer**, oder selbstdefinierte **Aufzählungstypen**.

Siehe auch

Prozedurtyp-Konstanten

Strukturierte Typen

Ein strukturierter Typ enthält mehr als einen Wert. Strukturierte Typen sind

Array-Typen
Dateitypen
Objekttypen
Record-Typen
Mengentypen

Strukturierte Typen können wiederum strukturierte Typen als Komponenten enthalten. Sie können aus also mehreren (tatsächlich unbegrenzt vielen) Strukturebenen bestehen.

Turbo Pascal erlaubt strukturierte Typen mit einem maximalen Umfang von 65520 Bytes.

Das reservierte Wort **packed** in der Deklaration eines Strukturtyps weist den Compiler an, diese Daten in "gepackter Form" (d.h. mit einem möglichst geringen Verbrauch von Speicherplatz) zu speichern.

Siehe auch

Strukturierte typisierte Konstanten

Gleitkomma-Berechnungen per Software

Im **{$N-} - Modus** des Compilers (das ist die Voreinstellung) wird Code erzeugt, der alle Real-Berechnungen per Software durchführt, indem er Routinen der Laufzeitbibliothek aufruft.

Um Geschwindigkeit und Codegröße zu optimieren, steht in diesem Modus nur der Typ **real** zur Verfügung.

Jeder Versuch, Anweisungen zu kompilieren, die die Typen Single, Double, Extended oder Comp enthalten, führt zu einer Fehlermeldung.

80x87 Gleitkomma-Berechnungen

Im **{$N+}-Modus**, des Compilers wird Code erzeugt, der 80x87-Anweisungen verwendet, um Gleitkomma-Berechnungen durchzuführen. Alle fünf **Real-Typen** sind erlaubt.

Gültigkeitsbereich eines Komponentenbezeichners

Der Gültigkeitsbereich eines Komponenten-**Bezeichners** umfaßt den Bereich seines **Objekttyps** und den der Blöcke von **Prozeduren**, **Funktionen**, **Konstruktoren** und **Destruktoren, die Methoden** des Objekttyps und seiner Nachkommen implementieren.

Deshalb muß ein Komponentenbezeichner innerhalb eines Objekttyps, seiner Nachkommen und seiner Methoden eindeutig sein.

Der Gültigkeitsbereich eines Komponentenbezeichners, der im **private** - Abschnitt eines Objekttyps deklariert wird, ist beschränkt auf das Modul (das **Programm** oder die **Unit**), das die Deklaration dieses Objekttyps enthält.

Als **private** deklarierte Komponentenbezeichner verhalten sich innerhalb des Moduls, in dem der Objekttyp deklariert ist, wie normale öffentliche Komponentenbezeicher.

Außerhalb des Moduls sind private Komponentenbezeichner unbekannt und es kann nicht auf sie zugegriffen werden.

Wenn Sie verwandte Objekttypen im selben Modul unterbringen, dann können diese gegenseitig auf private Komponenten zugreifen, ohne daß diese in anderen Modulen bekannt sind.

Variablendeklarationen

Eine Variablendeklaration enthält eine Liste von **Bezeichnern**, die ihrerseits für neue Variablen (**var**) und ihren jeweiligen **Typ** stehen.

Beispiele

```
var
  X, Y, Z; Real;
  I, J, K: Integer;
  Digit: 0..9;
  C: Color;
  Done,Error: Boolean
  Operator: (Plus, Minus, Times);
  Hue1, Hue2: set of Color;
  Today: Date;
  Matrix: array[1..10, 1..10] of Real;
```

Dabei kann der Typ ein Typbezeichner sein, der zuvor deklariert wurde
- in einer Typdeklaration desselben Blocks, oder
- in einem übergeordneten Block, oder
- in einer **Unit**,

oder es kann auch ein neuer Typ vereinbart werden.

Wenn ein Bezeichner in der Bezeichnerliste eines Deklarationsteils steht, gilt er innerhalb des ganzen Blocks, für den er deklariert wurde.

Auf diese Variable kann dann im gesamten Block Bezug genommen werden, es sei denn, derselbe Bezeichner wird in einem untergeordneten Block erneut verwendet (Redeklaration).

Eine Redeklaration erzeugt eine neue Variable mit demselben Namen, ohne daß dabei der Wert der ursprünglichen Variablen beeinflußt wird.

Siehe auch

var (reserviertes Wort)
Globale und lokale Variablen

Globale und lokale Variablen

Globale Variablen werden außerhalb von Prozeduren und Funtionen deklariert und im Datensegment gespeichert.

Lokale Variablen werden innerhalb von Prozeduren und Funtionen deklariert und im Stacksegment gespeichert.

Typisierte Konstanten

Typisierte Konstanten sind mit vorinitialisierten Variablen vergleichbar (Variable, deren Wert beim Aufruf ihres Blocks gesetzt wird).

Anders als bei **untypisierten Konstanten** enthält die Deklaration sowohl den **Typ** als auch den Wert.

Typisierte Konstanten können genauso wie **Variablen** desselben Typs verwendet werden und dürfen auf der linken Seite einer **Zuweisung** erscheinen.

Achtung: Typisierte Konstanten werden genau **einmal** initialisiert, nämlich bei Start des Prgramms. Das gilt auch für lokale Deklarationen: die Initialisierung wird also **nicht** bei jedem Aufruf der entsprechenden Prozedur oder Funktion wiederholt.

Anstelle eines normalen konstanten Ausdrucks kann man auch einen **konstanten Adreß-Ausdruck** dazu verwenden, den Wert einer typisierten Konstanten zu defnieren.

Beispiele

```
(* Deklaration von typisierten Konstanten *)
type
  Point = record X, Y: real end;
const
  Minimum: Integer = 0;
  Maximum: Integer = 9999;
  Factorial: array[1..7] of Integer = (1, 2, 6, 24, 120, 720, 5040);
  HexDigits: set of Char = ['0'..'9', 'A'..'Z', 'a'..'z'];
  Origin: Point = (X: 0.0; Y: 0.0);
```

Siehe auch

Array-Konstanten
Objektkonstanten
Zeigerkonstanten
Prozedurkonstanten
Record-Konstanten
Mengenkonstanten
Einfach typisierte Konstanten
String-Konstanten
Strukturierte Konstanten

Array-Konstanten

Die Deklaration eines Arrays als typisierte Konstante enthält die Werte der einzelnen Elemente, die dem Arraynamen in Klammern folgen und durch Kommas voneinander getrennt sind.

Der Komponenten-Typ einer Array-Konstanten ist beliebig - nur **Dateitypen** sind nicht erlaubt.

Beispiele

```
type
  Status = (active, passive, waiting);
  StatusMap = array[Status] of string[7];
const
  StatStr: StatusMap = ('Active', 'Passive', 'Waiting');

{ Und das sind die Komponenten von StatStr:

  StatStr[active] = 'Active'
  StatStr[passive] = 'Passive'
  StatStr[waiting] = 'Waiting' }
```

Zeichen-Arrays

Gepackte String-Konstanten (Zeichen-Arrays) können sowohl als Zeichenkettten (Strings) als auch als einzelne Zeichen definiert werden. Ein Beispiel:

```
const
  Digits: array[0..9] of Char = ('0', '1', '2', '3', '4', '5',
    '6', '7', '8', '9');
```

Dasselbe kann man auch bequemer erreichen:

```
const
  Digits: array[0..9] of Char = '0123456789';
```

Zeichen-Arrays, die ab Null indiziert sind

Das sind Arraysbei denen das erste Element 0 und das letzte eine positive Integer-Zahl ist. Ein Beispiel:

```
array[0..X] of Char
```

Wenn Sie die **Compiler-Direktive {$X+}** verwenden (schaltet die erweiterte Syntax ein), dann können Sie einen ab Null indizierten Zeichenarray mit einer Zeichenkette initialisierten, die kürzer ist als die deklarierte Länge des Arrays. Ein Beispiel:

```
const
  FileName = array[0..79] of Char = 'TEST.PAS';
```

Wenn die Zeichenkette kürzer ist als die Länge des Arrays, so wird der Rest des Arrays mit NULL (#0) aufgefüllt, und er enthält eine **Null-terminierte Zeichenkette**

Mehrdimensionale Array-Konstanten

Mehrdimensionale Array-Konstanten werden definiert, indem man die Konstanten einer jeden Dimension in ein eigenes Klammerpaar einschließt und diese Paare durch Kommas trennt.

The innersten Konstanten entsprechen der Dimension rechts außen.

Die Deklaration

```
type
  Cube = array[0..1, 0..1, 0..1] of Integer;
const
  Maze: Cube = (((0, 1), (2, 3)), ((4, 5), (6, 7)));
```

erzeugt einen initialisierten Array Maze mit folgenden Werten:

```
Maze[0, 0, 0] = 0
Maze[0, 0, 1] = 1
Maze[0, 1, 0] = 2
Maze[0, 1, 1] = 3
Maze[1, 0, 0] = 4
Maze[1, 0, 1] = 5
Maze[1, 1, 0] = 6
Maze[1, 1, 1] = 7
```

Konstanten eines Objekttyps

Bei der Deklaration einer Konstanten eines **Objekttyps** wird die gleiche Syntax wie bei der Deklaration einer **Record-Konstanten** verwendet.

Für die Methoden wird kein Wert angegeben.

Beispiele:

```
const
  ZeroPoint: Point = (X: 0; Y: 0);
  ScreenRect: Rect = (A: (X: 0; Y: 0); B: (X: 80; Y: 25));
  CountField: NumField = (X: 5; Y: 20; Len: 4; Name: nil;
    Value: 0; Min: -999; Max: 999);
```

Konstanten eines **Objekttyps**, der virtuelle Methoden enthält, müssen nicht mit Hilfe eines **Konstruktoraufrufs** initialisiert werden - diese Initialisierung besorgt automatisch der Compiler.

Zeigerkonstanten

Deklarationen von Zeigern als typisierte Konstante gebrauchen typischerweise einen **konstanten Adreß-Ausdruck** zur Definition des Zeigerwertes.

Wenn Sie mit dem **Compiler-Befehl {$X+}** die erweiterte Syntax anschalten, können **typisierte Konstanten** des Typs **PChar** mit einer String-Konstanten initialisiert werden.

Beispiele

```
type
  Direction = (Left, Right, Up, Down);
  StringPtr = ^String;
  NodePtr = ^Node;
  Node = record
    Next: NodePtr;
    Symbol: StringPtr;
    Value: Direction;
  end;
const
  S1: string[4] = 'DOWN';
  S2: string[2] = 'UP';
  S3: string[5] = 'RIGHT';
  S4: string[4] = 'LEFT';
  N1: Node = (Next: nil; Symbol: @S1; Value: Down);
  N2: Node = (Next: @N1; Symbol: @S2; Value: Up);
  N3: Node = (Next: @N2; Symbol: @S3; Value: Right);
  N4: Node = (Next: @N3; Symbol: @S4; Value: Left);
  DirectionTable: NodePtr = @N4;
```

Prozedurkonstanten

Eine Konstante prozeduralen Typs muß solche **Prozeduren** oder **Funktionen** referenzieren, die zuweisungskompatibel zum **Typ** der Konstanten sind.

Beispiel

```
type
  ErrorProc = procedure(ErrorCode: Integer);

procedure DefaultError(ErrorCode: Integer); far;
begin
  WriteLn('Error ', ErrorCode, '.');
end;

const
  ErrorHandler: ErrorProc = DefaultError;
```

Siehe auch

Typisierte Konstanten

Record-Konstanten

Die Deklaration einer **Record**-Konstanten enthält Bezeichner und Wert eines jeden Feldes.

Die Felder müssen in derselben Reihenfolge wie bei der Definition des **Recordtyps** angegeben werden.

- Wenn der Record **Dateitypen** enthält, kann das entsprechende Feld in einer Konstanten nicht deklariert werden.
- Wenn der Record variante Teile enthält, ist bei einer typisierten Konstanten nur die Angabe der aktiven Varianten möglich.
  - Enthält der Record zusätzlich ein Selektorfeld, so muß dessen Wert entsprechend angegeben sein.

Beispiele

```
type
  Point = record
    X, Y: Real;
  end;
  Vector = array[0..1] of Point;
  Month = (Jan, Feb, Mar, Apr, May, Jun, Jly, Aug, Sep, Oct, Nov, Dec);
  Date = record
    D: 1..31;
    M: Month;
    Y: 1900..1999;
  end;
const
  Origin: Point = (X: 0.0; Y: 0.0);
  Line: Vector = ((X: -3.1; Y: 1.5), (X: 5.8; Y: 3.0));
  SomeDay: Date = (D: 2; M: Dec; Y: 1960);
```

Mengenkonstanten

Die Deklaration einer **Mengenkonstanten** enthält die Elemente der Menge, eingeschlossen in eckige Klammern und durch Kommas voneinander getrennt.

Beispiele

```
type
  Digits = set of 0..9;
  Letters = set of 'A'..'Z';
const
  EvenDigits: Digits = [0, 2, 4, 6, 8];
  Vowels: Letters = ['A', 'E', 'I', 'O', 'U', 'Y'];
  HexDigits: set of '0'..'z' = ['0'..'9', 'A'..'F', 'a'...f'];
```

Einfach typisierte Konstanten

Die Deklaration einer einfach typisierten Konstanten enthält lediglich den Typ und den Wert.

Der Wert einer typisierten Konstanten kann mittels eines **konstanten Adreß-Ausdrucks** angegeben werden.

Da eine typisierte Konstante in Wirklichkeit eine initialisierte Variable ist, kann sie nicht wie eine gewöhnliche Konstante verwendet werden.

Beispiele

```
const
  Maximum: Integer = 9999;
  Factor: Real = -0.1;
  Breakchar: Char = #3;
```

String-Konstanten

Die Deklaration einer typisierten Konstanten als **String** (Zeichenkette) enthält die maximale Länge des Strings und seinen Wert:

Beispiele

```
const
  Heading: string[7] = 'Section';
  NewLine: string[2] = #13#10;
  TrueStr: string[5] = 'Yes';
  FalseStr: string[5] = 'No';
```

Strukturierte Konstanten

Die Deklaration einer strukturierten Konstanten gibt den Wert einer jeden Komponente der Struktur an.

Turbo Pascal unterstützt folgende Typen strukturierter Konstanten:

Arrays, **Records**, **Mengen**, und **Zeiger**.

Nicht erlaubt sind

Konstanten von **Dateitypen**
Array- und **Record**-Konstanten, die **Dateitypen** als Komponenten enthalten.

Konstante Adreß-Ausdrücke

Ein konstanter Adreß-Ausdruck enthält Adresse, Segment oder Offset einer globalen Variablen, einer **typisierten Konstanten**, einer **Prozedur** oder einer **Funktion.**

Konstante Adreß-Ausdrücke können nicht auf lokale oder dynamische Variablen Bezug nehmen, da deren Adresse zum Zeitpunkt der Kompilierung nicht bekannt ist.

Zuweisungen

Zuweisungen können zwei Dinge bewirken:

1. Sie ersetzen den momentanen Wert einer **Variablen** durch einen neuen Wert, der durch einen Ausdruck gegeben ist

2. Sie geben einen Ausdruck an, dessen Wert von einer **Funktion** zurückgegeben wird.

Der **Zuweisungsoperator** ist **:=.**

Beispiele

```
X := Y + Z;
Done := (I >= 1) and (I < 100);
Hue1 := [Blue, Succ(C)];
I := Sqr(J) - I * K;
```

Gültigkeitsbereiche

Jeder Bezeichner und jedes Label in einer Deklaration vereinbart genau ein Objekt oder Sprungziel.

Bezeichner oder Labels dürfen nur innerhalb ihres Gültigkeitsbereichs benutzt werden.

Dieser Gültigkeitsbereich reicht von der Deklaration bis zum Ende des dazugehörigen Blocks, einschließlich aller untergeordneten Blöcke.

Ausnahmen

1) Neudeklaration in einem untergeordneten Block:
Wenn Außen ein Block ist, der den Block Innen einschließt und in beiden derselbe Bezeicher x deklariert wird, dann können Innen und Außen nur auf das jeweilige x zugreifen, das sie selbst deklariert haben

2) Der Ort der Deklaration in einem Block:
Bezeichner und Labels müssen deklariert sein, bevor sie verwendet werden können.

Ausnahme: Der Grundtyp eines **Zeigers** darf ein noch nicht deklarierter Bezeichner sein.

Allerdings muß dieser Bezeichner noch in derselben Typ-Deklaration definiert werden.

3) Neudeklaration im selben Block:
Grundsätzlich dürfen Bezeichner und Labels auf der obersten Ebene eines Blocks nur einmal deklariert werden.

Ausnahmen: Innerhalb eines untergeordneten Blocks oder innerhalb der Feldliste eines **Records** dürfen sie erneut deklariert werden.

Feldbezeichner werden innerhalb des Recordtyps deklariert. Sie haben nur zusammen mit einem Verweis auf eine Variable dieses Recordtyps Bedeutung.

Man kann also einen Bezeichner eines Feldes im selben Block erneut deklarieren, jedoch nicht auf derselben Ebene desselben Recordtyps.

4) Bezeichner einer Objektkomponente:
Der **Gültigkeitsbereich des Bezeichners einer Objektkomponente** umfasst den Bereich des **Objekttyps**.

Der Gültigkeitsbereich von Interface-Bezeichnern

Programme oder Units, die **Uses** - Anweisungen enthalten, haben Zugriff auf die Bezeichner, die im **Interface**-Teil der dort genannten Units deklariert sind.

Spezielle Symbole

Spezielle Symbole und **reservierte Wörter** sind Zeichen von fester Bedeutung.

Spezielle Symbole sind folgende Zeichen:

```
+      -      *      /      =      <      >
[      ]      ,      (      )      :      ;
^      .      @      {      }      $      #
```

Und auch folgende Zeichenpaare:

```
<=     >=     :=     ..
(*     *)     (.     .)
```

Spezielle Symbole sind auch **Operatoren.**

- Eckige Klammern `[ ]` sind gleichbedeutend mit der Kombination von runder Klammer und Punkt `(. .)`.

Zahlen

Für **Konstanten** vom Typ Integer und Real findet die gewöhnliche dezimale Notation Verwendung.

Hexadezimale Integer-Konstanten haben ein Dollarzeichen ($) als Präfix.

Real-Typen können in wissenschaftlicher Notation dargestellt werden:
`7E-2` entspricht 7×10^{-2}
`12.25e+6` oder `12.25e6` bedeuten beide $12.25 \times 10^{+6}$.

Dezimalzahlen oder Zahlen mit Exponenten stehen für Real-Konstanten.

Ganzzahlige Dezimalzahlen stehen für Integer-Konstanten, wenn sie im Bereich von -2 147 483 648 bis 2 147 483 647 liegen.

Hexadezimalzahlen

Hexadezimalzahlen stehen für Integer-Konstanten; sie müssen im Bereich von $00000000 bis $FFFFFFFF liegen. Diese Notation enthält implizit ein Vorzeichen.

Zeichenketten (Strings)

Eine Stringkonstante ist eine Folge von Zeichen aus dem erweiterten ASCII-Zeichensatz. Sie ist von Apostrophen eingeschlossen und muß in einer Zeile des Quelltextes stehen.

Ein String kann auch leer sein, d.h. zwischen den Apostrophen müssen nicht unbedingt weitere Zeichen stehen.
Ein Apostroph innerhalb eines Strings läßt sich durch zwei aufeinanderfolgende Apostrophe darstellen.

Das Längenattribut einer Stringkonstanten enthält die Anzahl der Zeichen zwischen den Apostrophen.

Steuerzeichen

In Erweiterung des Sprachstandards erlaubt Turbo Pascal die Aufnahme von Steuer- und Graphikzeichen in Strings.

Das Zeichen `#`, gefolgt von einer vorzeichenlosen Integer-Konstanten aus dem Bereich von 0 bis 255, steht für den entsprechenden ASCII-Code.

Zwischen dem `#` und der Integerkonstanten dürfen keine weiteren Zeichen stehen.

Mehrere aufeinanderfolgende Steuerzeichen dürfen ebenfalls nicht durch weitere Zeichen getrrennt werden.

- Ein String der Länge 0 (der leere String) ist nur zu **Stringtypen** kompatibel.
- Ein String der Länge 1 ist zu jedem **Char**- und Stringtyp kompatibel.
- Ein String der Länge N (mit N >= 2) ist zu folgenden Typen kompatibel:
  - jedem Stringtyp
  - gepackten Arrays von N Zeichen
  - tdem Typ **PChar**, wenn die erweiterte Syntax mit der **Compilerdirective** `{$X+}`.angeschaltet ist.

Kommentare

Die folgenden Ausdrücke sind Kommentare und werden vom Compiler ignoriert:

```
{ dieser Text steht in geschweiften Klammern }
(* das ist eine Klammer mit Stern *)
```

Ein Kommentar, bei dem ein Dollarzeichen (**$**) unmittelbar nach dem öffnenden Kommentarzeichen **{** oder **(*** steht, ist ein **Compiler-Befehl**. Dem Dollarzeichen folgt ein Compiler-Befehl.

Index der MS-Windows API von Turbo Pascal

Konstanten

- alphabetischer Index
- funktionaler (kategorischer) Index
  - Aktionsschalter-Konstanten
  - Fähigkeits-Konstanten
  - Klassen-Konstanten
  - Farb-Konstanten
  - Kombinationsfenster-Konstanten
  - Kommunikations-Konstanten
  - Geräte-Konstanten
  - Dialogfenster-Konstanten
  - Editorkontroll-Konstanten
  - Fehler-Konstanten
  - Feld-Offset-Konstanten
  - Flag-Konstanten
  - Schrift-Konstanten
  - Listenfenster-Konstanten
  - Botschafts-Konstanten
  - Modus-Konstanten
  - MS-Windows-Konstanten
  - Benachrichtungscode--Konstanten
  - Raster-Konstanten
  - Bildlaufleisten-Konstanten
  - Stil-Konstanten
  - Fenster-Konstanten
  - Diverse Konstanten

Funktionen

- Fensterverwaltung
- GDI-Routinen
- Systemschnittstellen--Routines

Botschaften

- Aktionsschalter-Botschaften
- Kombinationsfenster-Botschaften
- DDE-Botschaften
- Dialogfenster-Botschaften
- Editor-Botschaften
- Listenfenster-Botschaften
- Fenster-Botschaften

Units

- WinProcs.TPU
- WinTypes.TPU

Dialogfenster, Windows API

Konstanten
Meldungen
Prozeduren und Funktionen

Index der Windows-API-Botschaften

Dieser Index enthält eine funktionale (kategorische) Liste der API-Botschaften von Turbo Pascal für Windows.

Diese Botschaften kann man in verschiedenen Kategorien zusammenfassen:

Aktionsschalter-Botschaften
Kombinationsfenster-Botschaften
DDE-Botschaften
Dialogfenster-Botschaften
Editor-Botschaften
Listenfenster-Botschaften
Fenster-Botschaften

Aktionsschalter-Botschaften (Windows API)

bm_GetCheck
bm_GetState
bm_SetCheck
bm_SetState
bm_SetStyle

Kombinationsfenster-Botschaften (Windows API)

cb_AddString
cb_DeleteString
cb_Dir
cb_FindString
cb_GetCount
cb_GetCurSel
cb_GetEditSel
cb_GetItemData
cb_GetLBText
cb_GetLBTextLen
cb_InsertString
cb_LimitText
cb_ResetContent
cb_SelectString
cb_SetCurSel
cb_SetEditSel
cb_SetItemData
cb_ShowDropDown

DDE-Botschaften (Windows API)

wm_dde_Ack
wm_dde_Advise
wm_dde_Data
wm_dde_Execute
wm_dde_Initiate
wm_dde_Poke
wm_dde_Request
wm_dde_Terminate
wm_dde_Unadvise

Dialogfenster-Botschaften (Windows API)

dm_GetDefID
dm_SetDefID

Editor-Botschaften (Windows API)

em_CanUndo
em_EmptyUndoBuffer
em_FmtLines
em_GetHandle
em_GetLine
em_GetLineCount
em_GetModify
em_GetRect
em_GetSel
em_LimitText
em_LineFromChar
em_LineIndex
em_LineLength
em_LineScroll
em_ReplaceSel
em_SetHandle
em_SetModify
em_SetPasswordChar
em_SetRect
em_SetRectNP
em_SetSel
em_SetTabStops
em_SetWordBreak
em_Undo

Listenfenster-Botschaften (Windows API)

lb_AddString
lb_DeleteString
lb_Dir
lb_FindString
lb_GetCount
lb_GetCurSel
lb_GetHorizontalExtent
lb_GetItemData
lb_GetItemRect
lb_GetSel
lb_GetSelCount
lb_GetSelItems
lb_GetText
lb_GetTextLen
lb_GetTopIndex
lb_InsertString
lb_ResetContent
lb_SelectString
lb_SelItemRange
lb_SetColumnWidth
lb_SetCurSel
lb_SetHorizontalExtent
lb_SetItemData
lb_SetSel
lb_SetTabStops
lb_SetTopIndex

Fenster-Botschaften (Windows API)

wm_Activate
wm_ActivateApp
wm_AskCBFormatName
wm_CancelMode
wm_ChangeCBChain
wm_Char
wm_ChartoItem
wm_ChildActivate
wm_Clear
wm_Close
wm_Command
wm_Compacting
wm_CompareItem
wm_Copy
wm_Create
wm_CtlColor
wm_Cut
wm_DeadChar
wm_DeleteItem
wm_Destroy
wm_DestroyClipboard
wm_DevModeChange
wm_DrawClipboard
wm_DrawItem
wm_Enable
wm_EndSession
wm_EnterIdle
wm_EraseBkgnd
wm_FontChange
wm_GetDlgCode
wm_GetFont
wm_GetMinMaxInfo
wm_GetText
wm_GetTextLength
wm_HScroll
wm_HScrollClipboard
wm_IconEraseBkgnd
wm_InitDialog
wm_InitMenu
wm_InitMenuPopup
wm_KeyDown
wm_KeyUp
wm_KillFocus
wm_LButtonDblClk
wm_LButtonDown
wm_LButtonUp
wm_MButtonDblClk
wm_MButtonDown
wm_MButtonUp
wm_MDIActivate
wm_MDICascade
wm_MDICreate
wm_MDIDestroy

wm_MDIGetActive
wm_MDIIconArrange
wm_MDIMaximize
wm_MDINext
wm_MDIRestore
wm_MDISetMenu
wm_MDITile
wm_MeasureItem
wm_MenuChar
wm_MenuSelect
wm_MouseActivate
wm_MouseMove
wm_Move
wm_NCActivate
wm_NCCalSize
wm_NCCreate
wm_NCDestroy
wm_NCHitTest
wm_NCLButtonDbClk
wm_NCLButtonDown
wm_NCLButtonUp
wm_NCMButtonDblClk
wm_NCMButtonDown
wm_NCMButtonUp
wm_NCMouseMove
wm_NCPaint
wm_NCRButtonDblClk
wm_NCRButtonDown
wm_NCRButtonUp
wm_NextDlgCtl
wm_Paint
wm_PaintClipboard
wm_PaintIcon
wm_PaletteChanged
wm_ParentNotify
wm_Paste
wm_QueryDragIcon
wm_QueryEndSession
wm_QueryNewPalette
wm_QueryOpen
wm_Quit
wm_RButtonDblClk
wm_RButtonDown
wm_RButtonUp
wm_RenderAllFormats
wm_RenderFormat
wm_SetCursor
wm_SetFocus
wm_SetFont
wm_SetRedraw
wm_SetText
wm_ShowWindow
wm_Size
wm_SizeClipboard
wm_SpoolerStatus

wm_SysChar
wm_SysColorChange
wm_SysCommand
wm_SysDeadChar
wm_SysKeyDown
wm_TimeChange
wm_Timer
wm_Undo
wm_VKeyToItem
wm_VScroll
wm_VScrollClipboard
wm_wininichange

Funktionaler Index von Windows API-Konstanten

Dieser Index ist eine funktionale (kategorische) Liste der API-Konstanten von Turbo Pascal für Windows.

Diese Konstanten sind in strikt alphabetischer Reihenfolge im **Alphabetischen Index der Windows-API-Konstanten** aufgeführt**.**

Aktionsschalter-Konstanten
Bildlaufleisten-Konstanten
Bitmap-Konstanten
DDE-Konstanten
DDL-Konstanten
Dialogfenster-Konstanten
Editorkontroll-Konstanten
Eigenschafts-Konstanten
Farb-Konstanten
Fehler-Konstanten
Feld-Offset-Konstanten
Fenster-Konstanten
Flag-Konstanten
Font-Konstanten
Geräte-Konstanten
Klassen-Konstanten
Kombinationsfenster-Konstanten
Kommunikations-Konstanten
Listenfenster-Konstanten
Meldungs-Konstanten
Metadatei-Konstanten
Modus-Konstanten
Maus-Konstanten
MS-Windows-Konstanten
Notifikationscode-Konstanten
Objekttyp-Konstanten
Owner-Zeichen-Konstanten
Raster-Konstanten
Stil-Konstanten
Diverse Konstanten

Aktionsschalter-Konstanten (Windows API)

bn_xxx
bs_xxx

Bitmap-Konstanten

bi_xxx
cbm_Init
cchDeviceName
dib_xxx
meta_xxx
rc_xxx
Ternäre Rasteroperationen

Eigenschafts-Konstanten (Windows API)

cp_xxx
cc_xxx
dib_xxx
dc_xxx
lc_xxx
pc_xxx
rc_xxx
tc_xxx

Klassen-Konstanten (Windows API)

gcl_xxx (long)
gcw_xxx (word)
cs_xxx

Farb-Konstanten (Windows API)

dib_xxx
ctlcolor_xxx
pc_xxx
color_xxx

Kombinationsfenster-Konstanten (Windows API)

cbn_xxx
cb_xxx
cbs_xxx

Kommunikations-Konstanten (Windows API)

Comm-Konfiguration
dcb_xxx
com_xxx
ce_xxx
ev_xxx
Escape-Comm-Konstanten
ie_xxx

DDL-Konstanten

wep_xxx

DDE-Konstanten

dde_xxx
dde_xxx

Geräte-Konstanten (Windows API)

cch_xxx
dib_xxx
dmbin_xxx
dmcolor_xxx
dmdup_xxx
dm_xxx
dmorient_xxx
dmpaper_xxx
dmres_xxx
dm_xxx
dt_xxx

Dialogfenster-Konstanten (Windows API)

dlgc_xxx
ds_xxx
idxxx
Dialogklassenkonstanten

Editorkontroll-Konstanten (Windows API)

en_xxx
es_xxx

Fehler-Konstanten (Windows API)

ce_xxx
ie_xxx
sp_xxx

Feld-Offset-Konstanten (Windows API)

gcl_xxx
gcw_xxx (word)
gwl_xxx (long)
gww_xxx (word)

Flag-Konstanten (Windows API)

rgn_xxx
Füllmuster-Stil-Flags
out_xxx
gmem_xxx
lmem_xxx
mb_xxx
mf_xxx
pc_xxx
Bereichs-Flags
swp_xxx
syspal_xxx
wf_xxx

Font-Konstanten (Windows API)

clip_xxx
ff_xxx
Font-Zeichensatz-Flags
Font-Masken
Font-Ausgabequalitäts-Flags
Font-Pitch-Flags
fw_xxx
lf_xxx
out_xxx

ID-Konstanten (Windows API)

dib_xxx
idxxx
idc_xxx
idi_xxx

Listenfenster-Konstanten (Windows API)

lbn_xxx
lb_xxx
lbs_xxx

Meldungs-Konstanten (Windows API)

msgf_xxx
msgf_xxx
pm_xxx

Metadatei-Konstanten

Metadatei-Code

Modus-Konstanten (Windows API)

Hintergrund-Modus
mm_xxx
PolyFill-Modi
StretchBlt-Modi

Maus-Konstanten

ma_xxx

MS-Windows-Konstanten (Windows API)

wf_xxx
wh_xxx

Notifikationscode-Konstanten (Windows API)

bn_xxx
cbn_xxx
en_xxx
lbn_xxx

Objekttyp-Konstanten

obj_xxx

Owner-Zeichen-Konstanten

oda_xxx
ods_xxx
odt_xxx

Raster-Konstanten (Windows API)

r2_xxx
rc_xxx
Ternäre Rasteroperationen

Bildlaufleisten-Konstanten (Windows API)

sb_xxx
sb_xxx
sbs_xxx

Stil-Konstanten (Windows API)

bs_xxx
bs_xxx
cs_xxx
cbs_xxx
ds_xxx
es_xxx
ws_ex_xxx
hs_xxx
lbs_xxx
ps_xxx
sbs_xxx
ss_xxx
ws_xxx

Fenster-Konstanten (Windows API)

ws_ex_xxx
gw_xxx
show_xxx
sw_xxx
sw_xxx
gwl_xxx
gww_xxx **(word)**
ws_xxx

Diverse Konstanten (Windows API)

cf_xxx
cw_xxx
drive_xxx
Drucker-Escape-Sequenzen
dt_xxx
eto_xxx
help_xxx
htxxx
LPTx Konstanten
mk_xxx
obm_xxx
proc_xxx
of_xxx
pr_xxx
rt_xxx
s_xxx
sc_xxx
sizexxx
sm_xxx
sp_xxx
Stock logische Objekte
ta_xxx
tf_xxx
vk_xxx

Alphabetischer Index der Windows API-Konstanten

Dieser Index ist eine alphabetische Liste der API-Konstanten von Turbo Pascal für Windows.

Nach Kategorie oder Funktion geordnet sind diese Konstanten im **Funktionalen Index der Windows-API-Konstanten** aufgeführt..

AbortDoc
Alternate
ANSI_CharSet
ANSI_Fixed_Font
ANSI_Var_Font
AspectX
AspectXY
AspectY
BandInfo
Begin_Path
bi_RGB
bi_RLE8
bi_RLE4
BitsPixel
Black_Brush
Black_Pen
Blackness
BlackOnWhite
bn_Clicked
bn_DoubleClicked
bs_3State
bs_Auto3State
bs_AutoCheckBox
bs_AutoRadioButton
bs_CheckBox
bs_DefPushButton
bs_DIBPattern
bs_GroupBox
bs_Hatched
bs_Hollow
bs_LeftText
bs_Null
bs_OwnerDraw
bs_Pattern
bs_PushButton
bs_RadioButton
bs_Solid
cb_Err
cb_Okay
cbm_Init
cbn_DblClk
cbn_DropDown
cbn_EditChange
cbn_EditUpdate
cbn_ErrSpace
cbn_KillFocus
cbn_SelChange

cbn_SetFocus
cbs_AutoHScroll
cbs_DropDown
cbs_DropDownList
cbs_HasStrings
cbs_NoIntegralHeight
cbs_OEMConvert
cbs_OwnerDrawFixed
cbs_OwnerDrawVariable
cbs_Simple
cbs_Sort
cc_Chord
cc_Circles
cc_Ellipses
cc_Interiors
cc_None
cc_Pie
cc_Styled
cc_Wide
cc_WideStyled
cchDeviceName
ce_Break
ce_DNS
ce_DSRTO
ce_Frame
ce_IOE
ce_Mode
ce_OOP
ce_Overrun
ce_PTO
ce_RLSDTO
ce_RxOver
ce_RxParity
ce_STSTO
ce_TxFull
cf_Bitmap
cf_DIB
cf_DIF
cf_DSPBitmap
cf_DSPMetaFilePict
cf_DSPText
cf_MetaFilePict
cf_OEMText
cf_OwnerDisplay
cf_Palette
cf_PrivateFirst
cf_PrivateLast
cf_SYLK
cf_Text
cf_TIFF
clip_Character_Precis
clip_Default_Precis
clip_Stroke_Precis
ClipCaps
Clip_To_Path

ClrDTR
ClrRTS
color_ActiveBorder
color_ActiveCaption
color_AppWorkSpace
color_Background
color_BtnFace
color_BtnShadow
color_BtnText
color_CaptionText
color_GrayText
color_Highlight
color_HighlightText
color_InactiveBorder
color_InactiveCaption
color_Menu
color_MenuText
color_ScrollBar
color_Window
color_WindowFrame
color_WindowText
ColorOnColor
ColorRes
com_CtsHold
com_DsrHold
com_Eof
com_RlsdHold
com_Txim
com_XoffHold
com_XoffSent
ComplexRegion
cp_None
cp_Rectangle
cs_ByteAlignClient
cs_ByteAlignWindow
cs_ClassDC
cs_DblClks
cs_GlobalClass
cs_HRedraw
cs_NoClose
cs_OwnDC
cs_ParentDC
cs_SaveBits
cs_VRedraw
ctlcolor_Btn
ctlcolor_Dlg
ctlcolor_Edit
ctlcolor_ListBox
ctlcolor_MsgBox
ctlcolor_ScrollBar
ctlcolor_Static
CurveCaps
cw_UseDefault
dc_BinNames
dc_Bins

dc_Driver
dc_Duplex
dc_Extra
dc_Fields
dc_MaxExtent
dc_MinExtent
dc_Papers
dc_Papersize
dc_Size
dc_Version
dcb_Binary
dcb_RtsDisable
dcb_Parity
dcb_OutxCtsFlow
dcb_OutxDsrFlow
dcb_DtrDisable
dcb_OutX
dcb_InX
dcb_PeChar
dcb_Null
dcb_ChEvt
dcb_Dtrflow
dcb_Rtsflow
dde_Ack
dde_AckReq
dde_AppReturnCode
dde_Busy
dde_DeferUpdt
dde_Response
dde_Release
Default_Palette
Default_Pitch
Default_Quality
DeviceData
Device_Default_Font
Device_FontType
DlgWindowExtra
dib_Pal_Colors
dib_RGB_Colors
DkGray_Brush
dlgc_DefPushButton
dlgc_HasSetSel
dlgc_RadioButton
dlgc_UndefPushButton
dlgc_WantAllKeys
dlgc_WantArrows
dlgc_WantChars
dlgc_WantMessage
dlgc_WantTab
dm_Color
dm_Copies
dm_Copy
dm_DefaultSource
dm_Duplex
dm_Modify

dm_Orientation
dm_PaperLength
dm_PaperSize
dm_PaperWidth
dm_PrintQuality
dm_Prompt
dm_Scale
dm_Update
dmbin_Auto
dmbin_Cassette
dmbin_Envelope
dmbin_EnvManual
dmbin_LargeCapacity
dmbin_LargeFmt
dmbin_Lower
dmbin_Manual
dmbin_Middle
dmbin_OnlyOne
dmbin_SmallFmt
dmbin_Tractor
dmbin_Upper
dmcolor_Color
dmcolor_Monochrome
dmdup_Horizontal
dmdup_Simplex
dmdup_Vertical
dmorient_Landscape
dmorient_Portrait
dmpaper_10X14
dmpaper_11X17
dmpaper_A3
dmpaper_A4
dmpaper_A4Small
dmpaper_A5
dmpaper_B4
dmpaper_B5
dmpaper_CSheet
dmpaper_DSheet
dmpaper_Env_10
dmpaper_Env_11
dmpaper_Env_12
dmpaper_Env_14
dmpaper_Env_9
dmpaper_ESheet
dmpaper_Executive
dmpaper_Folio
dmpaper_Ledger
dmpaper_Legal
dmpaper_Letter
dmpaper_LetterSmall
dmpaper_Note
dmpaper_Quarto
dmpaper_Statement
dmpaper_Tabloid
dmres_Draft

dmres_Low
dmres_Medium
dmres_High
DraftMode
Draft_Quality
DrawPatternRect
drive_Fixed
drive_Remote
drive_Removeable
DriverVersion
ds_AbsAlign
ds_LocalEdit
ds_ModalFrame
ds_NoIdleMsg
ds_SetFont
ds_SysModal
DSTInvert
dt_Bottom
dt_CalcRect
dt_Center
dt_CharStream
dt_DispFile
dt_ExpandTabs
dt_ExternalLeading
dt_Internal
dt_Left
dt_MetaFile
dt_NoClip
dt_NoPrefix
dt_Plotter
dt_RasCamera
dt_RasDisplay
dt_RasPrinter
dt_Right
dt_SingleLine
dt_TabStop
dt_Top
dt_VCenter
dt_WordBreak
Ebenen
en_Change
en_ErrSpace
en_HScroll
en_KillFocus
en_MaxText
en_SetFocus
en_Update
en_VScroll
EnableDuplex
EnablePairKerning
EnableRelativeWidths
End_Path
EndDoc
EnumPaperBins
EnumPaperMetrics

EPSPrinting
Error
es_AutoHScroll
es_AutoVScroll
es_Center
es_Left
es_LowerCase
es_MultiLine
es_NoHideSel
es_OEMConvert
es_Password
es_Right
es_UpperCase
eto_Clipped
eto_Opaque
ev_Break
ev_CTS
ev_DSR
ev_Err
ev_PErr
ev_Ring
ev_RLSD
ev_RxChar
ev_RxFlag
ev_TxEmpty
EvenParity
Ext_Device_Caps
ff_Decorative
ff_DontCare
ff_Modern
ff_Roman
ff_Script
ff_Swiss
Fixed_Pitch
FloodFillBorder
FloodFillSurface
FlushOutput
fw_Black
fw_Bold
fw_DemiBold
fw_DontCare
fw_ExtraBold
fw_ExtraLight
fw_Heavy
fw_Light
fw_Medium
fw_Normal
fw_Regular
fw_SemiBold
fw_Thin
fw_UltraBold
fw_UltraLight
gcl_MenuName
gcl_WndProc
gcw_CBClsExtra

gcw_CBWndExtra
gcw_HBrBackground
gcw_HCursor
gcw_HIcon
gcw_HModule
gcw_Style
GDIExtTextOut
GDIStretchBlt
GetColorTable
GetExtendedTextMetrics
GetExtentTable
GetPairKernTable
GetPhysPageSize
GetPrintingOffset
GetScalingFactor
GetSetPaperBins
GetSetPaperMetrics
GetSetPrintOrient
GetTechnolgy
GetTrackKernTable
GetVectorBrushSize
GetVectorPenSize
gmem_DDEShare
gmem_Discardable
gmem_Discarded
gmem_Fixed
gmem_LockCount
gmem_Lower
gmem_Modify
gmem_Moveable
gmem_NoCompact
gmem_NoDiscard
gmem_Not_Banked
gmem_Notify
gmem_Share
gmem_ZeroInit
Gray_Brush
gw_Child
gw_HWndFirst
gw_HWndLast
gw_HWndNext
gw_HWndPrev
gw_Owner
gwl_ExStyle
gwl_Style
gwl_WndProc
gww_HInstance
gww_HWndParent
gww_ID
help_Context
help_HelpOnHelp
help_Index
help_Key
help_MultiKey
help_Quit

help_SetIndex
hide_Window
Hollow_Brush
HorzRes
HorzSize
hs_BDiagonal
hs_Cross
hs_DiagCross
hs_FDiagonal
hs_Horizontal
hs_Vertical
htBottom
htBottomLeft
htBottomRight
htCaption
htClient
htError
htGrowBox
htHScroll
htLeft
htMenu
htNowhere
htReduce
htRight
htSize
htSysMenu
htTop
htTopLeft
htTopRight
htTransparent
htVScroll
htZoom
idAbort
idc_Arrow
idc_Cross
idc_IBeam
idc_Icon
idc_Size
idc_SizeNESW
idc_SizeNS
idc_SizeNWSE
idc_SizeWE
idc_UpArrow
idc_Wait
idCancel
idi_Application
idi_Asterisk
idi_Exclamation
idi_Hand
idi_Question
idIgnore
idNo
idOk
idRetry
idYes

ie_BadID
ie_BaudRate
ie_ByteSize
ie_Default
ie_Hardware
ie_Memory
ie_NOpen
ie_Open
lb_Err
lb_ErrSpace
lb_Okay
lbn_DblClk
lbn_ErrSpace
lbn_KillFocus
lbn_SelChange
lbn_SetFocus
lbs_ExtendedSel
lbs_HasStrings
lbs_MultiColumn
lbs_MultipleSel
lbs_NoIntegralHeight
lbs_NoRedraw
lbs_Notify
lbs_OwnerDrawFixed
lbs_OwnerDrawFixed
lbs_OwnerDrawVariable
lbs_OwnerDrawVariable
lbs_Sort
lbs_Standard
lbs_UseTabStops
lbs_WantKeyboardInput
lc_Interiors
lc_Marker
lc_None
lc_PolyLine
lc_PolyMarker
lc_Styled
lc_Wide
lc_WideStyled
lf_FaceSize
LineCaps
lmem_Discardable
lmem_Discarded
lmem_Fixed
lmem_LockCount
lmem_Modify
lmem_Moveable
lmem_NoCompact
lmem_NoDiscard
lmem_ZeroInit
LogPixelsX
LogPixelsY
LPTx
LtGray_Brush
ma_Activate

ma_ActivateAndEat
ma_NoActivate
MarkParity
mb_AbortRetryIgnore
mb_ApplModal
mb_DefButton1
mb_DefButton2
mb_DefButton3
mb_IconAsterisk
mb_IconExclamation
mb_IconHand
mb_IconInformation
mb_IconQuestion
mb_IconStop
mb_Ok
mb_OkCancel
mb_RetryCancel
mb_SystemModal
mb_TaskModal
mb_YesNo
mb_YesNoCancel
MergeCopy
MergePaint
meta_AnimatePalette
meta_Arc
meta_BitBlt
meta_Chord
meta_CreateBitmap
meta_CreateBitmapIndirect
meta_CreateBrush
meta_CreateBrushIndirect
meta_CreateFontIndirect
meta_CreatePalette
meta_CreatePatternBrush
meta_CreatePenIndirect
meta_CreateRegion
meta_DeleteObject
meta_DIBBitBlt
meta_DIBCreatePatternBrush
meta_DIBStretchBlt
meta_DrawText
meta_Ellipse
meta_Escape
meta_ExcludeClipRect
meta_ExtTextOut
meta_FillRegion
meta_FloodFill
meta_FrameRegion
meta_IntersectClipRect
meta_InvertRegion
meta_LineTo
meta_MoveTo
meta_OffsetClipRgn
meta_OffsetViewportOrg
meta_OffsetWindowOrg

meta_PaintRegion
meta_PatBlt
meta_Pie
meta_Polygon
meta_PolyLine
meta_PolyPolygon
meta_RealizePalette
meta_Rectangle
meta_ResizePalette
meta_RestoreDC
meta_RoundRect
meta_SaveDC
meta_ScaleViewportExt
meta_ScaleWindowExt
meta_SelectClipRegion
meta_SelectObject
meta_SelectPalette
meta_SetBKColor
meta_SetBKMode
meta_SetDIBToDev
meta_SetMapMode
meta_SetMapperFlags
meta_SetPalEntries
meta_SetPixel
meta_SetPolyFillMode
meta_SetRelAbs
meta_SetROP2
meta_SetStretchBltMode
meta_SetTextAlign
meta_SetTextCharExtra
meta_SetTextColor
meta_SetTextJustification
meta_SetViewportExt
meta_SetViewportOrg
meta_SetWindowExt
meta_SetWindowOrg
meta_StretchBlt
meta_TextOut
mf_Bitmap
mf_ByCommand
mf_ByPosition
mf_Checked
mf_Disabled
mf_Enabled
mf_Grayed
mf_Help
mf_Hilite
mf_MenuBarBreak
mf_MenuBreak
mf_MouseSelect
mf_OwnerDraw
mf_Popup
mf_Separator
mf_String
mf_SysMenu

mf_Unchecked
mf_Unhilite
MFComment
mk_Control
mk_LButton
mk_MButton
mk_RButton
mk_Shift
mm_Anisotropic
mm_HiEnglish
mm_HiMetric
mm_Isotropic
mm_LoEnglish
mm_LoMetric
mm_Text
mm_Twips
msgf_DialogBox
msgf_DialogBoxFilter
msgf_Menu
msgf_MenuFilter
msgf_MessageBoxFilter
NewFrame
NextBand
NoParity
NotSrcCopy
NotSrcErase
Null_Brush
Null_Pen
NullRegion
NumBrushes
NumColors
NumFonts
NumMarkers
NumPens
NumReserved
obj_Pen
obj_Brush
obm_BtnCorners
obm_BtSize
obm_Check
obm_CheckBoxes
obm_Close
obm_Combo
obm_DnArrow
obm_DnArrowD
obm_LfArrow
obm_LfArrowD
obm_MnArrow
obm_Old_Close
obm_Old_DnArrow
obm_Old_LfArrow
obm_Old_Reduce
obm_Old_Restore
obm_Old_RgArrow
obm_Old_UpArrow

obm_Old_Zoom
obm_Reduce
obm_Reduced
obm_Restore
obm_Restored
obm_RgArrow
obm_RgArrowD
obm_Size
obm_UpArrow
obm_UpArrowD
obm_Zoom
obm_ZoomD
oda_DrawEntire
oda_Focus
oda_Select
ods_Checked
ods_Disabled
ods_Focus
ods_Grayed
ods_Selected
odt_Menu
odt_ListBox
odt_ComboBox
odt_Button
OddParity
OEM_CharSet
OEM_Fixed_Font
of_Cancel
of_Create
of_Delete
of_Exist
of_Parse
of_Prompt
of_Read
of_ReadWrite
of_ReOpen
of_Share_Compat
of_Share_Deny_None
of_Share_Deny_Read
of_Share_Deny_Write
of_Share_Exclusive
of_Verify
of_Write
OneStopBit
One5StopBits
Opaque
OpenFile
Out_Character_Precis
Out_Default_Precis
Out_String_Precis
Out_Stroke_Precis
PassThrough
PatCopy
PatInvert
PatPaint

pc_Explicit
pc_Interiors
pc_NoCollapse
pc_None
pc_Polygon
pc_Rectangle
pc_Reserved
pc_ScanLine
pc_Styled
pc_Trapezoid
pc_Wide
pc_WideStyled
pc_WindPolygon
PDeviceSize
pm_NoRemove
pm_NoYield
pm_Remove
PolygonalCaps
pr_JobStatus
proc_ExtDeviceMode
proc_DeviceCapabilities
proc_OldDeviceMode
Proof_Quality
ps_Dash
ps_DashDot
ps_DashDotDot
ps_Dot
ps_InsideFrame
ps_Null
ps_Solid
QueryEscSupport
r2_Black
r2_CopyPen
r2_MaskNotPen
r2_MaskPen
r2_MaskPenNot
r2_MergeNotPen
r2_MergePen
r2_MergePenNot
r2_No en
r2_Nop
r2_Not
r2_NotCopyPen
r2_NotMaskPen
r2_NotMergePen
r2_White
r2_XorPen
RasterCaps
Raster_FontType
rc_Banding
rc_BigFont
rc_BitBlt
rc_Bitmap64
rc_DI_Bitmap
rc_DIBToDev

rc_FloodFill
rc_GDI20_Output
rc_Palette
rc_Scaling
rc_StretchBlt
rc_StretchDIB
ResetDev
Restore_CTM
rgn_And
rgn_Copy
rgn_Diff
rgn_Or
rgn_Xor
rt_Accelerator
rt_Bitmap
rt_Cursor
rt_Dialog
rt_Font
rt_FontDir
rt_Icon
rt_Menu
rt_RcData
rt_String
s_AllThreshold
s_Legato
s_Normal
s_Period1024
s_Period2048
s_Period512
s_PeriodVoice
s_QueueEmpty
s_SerBDNT
s_SerDCC
s_SerDDR
s_SerDFQ
s_SerDLN
s_SerDMD
s_SerDPT
s_SerDSH
s_SerDSR
s_SerDST
s_SerDTP
s_SerDVL
s_SerDVNA
s_SerMACT
s_SerOFM
s_SerQFUL
s_Staccato
s_Threshold
s_White1024
s_White2048
s_White512
s_WhiteVoice
Save_Ctm
sb_Both

sb_Bottom
sb_Ctl
sb_EndScroll
sb_Horz
sb_LineDown
sb_LineUp
sb_PageDown
sb_PageUp
sb_ThumbPosition
sb_ThumbTrack
sb_Top
sb_Vert
sbs_BottomAlign
sbs_Horz
sbs_LeftAlign
sbs_RightAlign
sbs_SizeBox
sbs_SizeBoxBottomRightAlign
sbs_SizeBoxTopLeftAlign
sbs_TopAlign
sbs_Vert
sc_Close
sc_HScroll
sc_KeyMenu
sc_Maximize
sc_Minimize
sc_MouseMenu
sc_Move
sc_NextWindow
sc_PrevWindow
sc_Restore
sc_Size
sc_TaskList
sc_VScroll
SelectPaperSource
Set_Arc_Direction
Set_Background_Color
Set_Bounds
Set_Clip_Box
Set_Poly_Mode
Set_Screen_Angle
Set_Spread
SetAbortProc
SetAllJustValues
SetColorTable
SetCopyCount
SetDTR
SetKernTrack
SetLineJoin
SetMiterLimit
SetRTS
SetXoff
SetXon
ShiftJIS_CharSet
show_IconWindow

show_FullScreen
show_OpenNoActivate
show_OpenWindow
SimpleRegion
SizeFullScreen
SizeIconic
SizeNormal
SizePalette
SizeZoomHide
SizeZoomShow
sm_CXBorder
sm_CXCursor
sm_CXDlgFrame
sm_CXFrame
sm_CXFullScreen
sm_CXHScroll
sm_CXHThumb
sm_CXIcon
sm_CXMin
sm_CXMinTrack
sm_CXScreen
sm_CXSize
sm_CXVScroll
sm_CYBorder
sm_CYCaption
sm_CYCursor
sm_CYDlgFrame
sm_CYFrame
sm_CYFullScreen
sm_CYHScroll
sm_CYIcon
sm_CYKanjiWindow
sm_CYMenu
sm_CYMin
sm_CYMinTrack
sm_CYScreen
sm_CYSize
sm_CYVScroll
sm_CYVThumb
sm_Debug
sm_MousePresent
sm_SwapButton
sp_AppAbort
sp_Error
sp_OutOfDisk
sp_OutOfMemory
sp_UserAbort
SpaceParity
SrcAnd
SrcCopy
SrcErase
SrcInvert
SrcPaint
ss_BlackFrame
ss_BlackRect

ss_Center
ss_GrayFrame
ss_GrayRect
ss_Icon
ss_Left
ss_LeftNoWordWrap
ss_NoPrefix
ss_Right
ss_Simple
ss_UserItem
ss_WhiteFrame
ss_WhiteRect
StartDoc
sw_Hide
sw_Maximize
sw_Minimize
sw_Normal
sw_OtherZoom
sw_OtherUnzoom
sw_ParentClosing
sw_ParentOpening
sw_Restore
sw_Show
sw_ShowMaximized
sw_ShowMinimized
sw_ShowMinNoActive
sw_ShowNA
sw_ShowNoActivate
sw_ShowNormal
swp_DrawFrame
swp_HideWindow
swp_NoActivate
swp_NoMove
swp_NoRedraw
swp_NoSize
swp_NoZOrder
swp_ShowWindow
Symbol_CharSet
syspal_NoStatic
syspal_Static
System_Fixed_Font
System_Font
ta_BaseLine
ta_Bottom
ta_Center
ta_Left
ta_NoUpdateCP
ta_Right
ta_Top
ta_UpdateCP
tc_cp_Stroke
tc_cr_90
tc_cr_Any
tc_ea_Double
tc_ia_Able

tc_op_Character
tc_op_Stroke
tc_ra_Able
tc_sa_Contin
tc_sa_Double
tc_sa_Integer
tc_sf_X_YIndep
tc_so_Able
tc_ua_Able
tc_va_Able
tf_ForceDrive
Technology
TextCaps
TwoStopBits
Transform_CTM
Transparent
vk_0 to vk_9
vk_A to vk_Z
vk_Add
vk_Back
vk_Cancel
vk_Capita
vk_Clear
vk_Control
vk_Decimal
vk_Delete
vk_Divide
vk_Down
vk_End
vk_Escape
vk_Execute
vk_F1 to vk_F16
vk_Help
vk_Home
vk_Insert
vk_LButton
vk_Left
vk_MButton
vk_Menu
vk_Multiply
vk_Next
vk_NumLock
vk_NumPad0 to vk_NumPad0
vk_Pause
vk_Print
vk_Prior
vk_RButton
vk_Return
vk_Right
vk_Select
vk_Separator
vk_Shift
vk_SnapShot
vk_Space
vk_Subtract

vk_Tab
vk_Up
Variable_Pitch
VertRes
VertSize
wep_Free_DLL
wep_System_Exit
wf_80x87
wf_CPU086
wf_CPU186
wf_CPU286
wf_CPU386
wf_CPU486
wf_Enhanced
wf_LargeFrame
wf_PMode
wf_SmallFrame
wf_Standard
wf_Win286
wf_Win386
wh_CallWndProc
wh_GetMessage
wh_JournalPlayback
wh_JournalRecord
wh_Keyboard
wh_MsgFilter
wh_SysMsgFilter
White_Brush
White_Pen
Whiteness
WhiteOnBlack
Winding
ws_Border
ws_Caption
ws_Child
ws_ChildWindow
ws_ClipChildren
ws_ClipSiblings
ws_Disabled
ws_DlgFrame
ws_ex_DlgModalFrame
ws_ex_NoParentNotify
ws_Group
ws_HScroll
ws_Iconic
ws_Maximize
ws_MaximizeBox
ws_Minimize
ws_MinimizeBox
ws_Overlapped
ws_OverlappedWindow
ws_Popup
ws_PopupWindow
ws_SizeBox
ws_SysMenu

ws_TabStop
ws_ThickFrame
ws_Tiled
ws_TiledWindow
ws_Visible
ws_VScroll

Alphabetischer Index der Windows API Prozeduren und Funktionen

Dieser Index ist eine strikt alphabetische Liste der Windows API Routinen.

Eine kategorische (funktionale) Liste dieser Routinen finden Sie im **Kategorischen Index der Windows API Prozeduren und Funktionen**.

AccessResource
AddAtom
AddFontResource
AdjustWindowRect
AdjustWindowRectEx
AllocDStoCSAlias
AllocResource
AllocSelector
AnimatePalette
AnsiLower
AnsiLowerBuff
AnsiNext
AnsiPrev
AnsiToOem
AnsiToOemBuff
AnsiUpper
AnsiUpperBuff
AnyPopup
AppendMenu
Arc
ArrangeIconicWindows
BeginDeferWindowPos
BeginPaint
BitBlt
BringWindowToTop
BuildCommDCB
CallMsgFilter
CallWindowProc
Catch
ChangeClipboardChain
CheckDlgButton
CheckMenuItem
CheckRadioButton
ChildWindowFromPoint
Chord
ClearCommBreak
ClientToScreen
ClipCursor
CloseClipboard
CloseComm
CloseMetaFile
CloseSound
CloseWindow
CombineRgn
CopyMetaFile
CopyRect
CountClipboardFormats
CountVoiceNotes
CreateBitmap

CreateBitmapIndirect
CreateBrushIndirect
CreateCaret
CreateCompatibleBitmap
CreateCompatibleDC
CreateCursor
CreateDC
CreateDialog
CreateDialogIndirect
CreateDialogIndirectParam
CreateDialogParam
CreateDIBitmap
CreateDIBPatternBrush
CreateDiscardableBitmap
CreateEllipticRgn
CreateEllipticRgnIndirect
CreateFont
CreateFontIndirect
CreateHatchBrush
CreateIC
CreateIcon
CreateMenu
CreateMetaFile
CreatePalette
CreatePatternBrush
CreatePen
CreatePenIndirect
CreatePolygonRgn
CreatePolyPolygonRgn
CreatePopupMenu
CreateRectRgn
CreateRectRgnIndirect
CreateRoundRectRgn
CreateSolidBrush
CreateWindow
CreateWindowEx
DebugBreak
DefDlgProc
DeferWindowPos
DefFrameProc
DefHookProc
DefMDIChildProc
DefWindowProc
DeleteAtom
DeleteDC
DeleteMenu
DeleteMetaFile
DeleteObject
DestroyCaret
DestroyCursor
DestroyIcon
DestroyMenu
DestroyWindow
DialogBox
DialogBoxIndirect

DialogBoxIndirectParam
DialogBoxParam
DispatchMessage
DlgDirList
DlgDirListComboBox
DlgDirSelect
DlgDirSelectComboBox
DPtoLP
DrawFocusRect
DrawIcon
DrawMenuBar
DrawText
Ellipse
EmptyClipboard
EnableHardwareInput
EnableMenuItem
EnableWindow
EndDeferWindowPos
EndDialog
EndPaint
EnumChildWindows
EnumClipboardFormats
EnumFonts
EnumMetaFile
EnumObjects
EnumProps
EnumTaskWindows
EnumWindows
EqualRect
EqualRgn
Escape
EscapeCommFunction
ExcludeClipRect
ExcludeUpdateRgn
ExitWindows
ExtFloodFill
ExtTextOut
FatalExit
FillRect
FillRgn
FindAtom
FindResource
FindWindow
FlashWindow
FloodFill
FlushComm
FrameRect
FrameRgn
FreeLibrary
FreeModule
FreeProcInstance
FreeResource
GetActiveWindow
GetAspectRatioFilter
GetAsyncKeyState

GetAtomHandle
GetAtomName
GetBitmapBits
GetBitmapDimension
GetBkColor
GetBkMode
GetBrushOrg
GetBValue
GetCapture
GetCaretBlinkTime
GetCaretPos
GetCharWidth
GetClassInfo
GetClassLong
GetClassName
GetClassWord
GetClientRect
GetClipboardData
GetClipboardFormatName
GetClipboardOwner
GetClipboardViewer
GetClipBox
GetCodeHandle
GetCodeInfo
GetCommError
GetCommEventMask
GetCommState
GetCurrentPDB
GetCurrentPosition
GetCurrentTask
GetCurrentTime
GetCursorPos
GetDC
GetDCOrg
GetDesktopWindow
GetDeviceCaps
GetDialogBaseUnits
GetDIBits
GetDlgCtrlID
GetDlgItem
GetDlgItemInt
GetDlgItemText
GetDOSEnvironment
GetDoubleClickTime
GetDriveType
GetEnvironment
GetFocus
GetFreeSpace
GetGValue
GetInputState
GetInstanceData
GetKBCodePage
GetKeyboardState
GetKeyboardType
GetKeyNameText

GetKeyState
GetLastActivePopup
GetMapMode
GetMenu
GetMenuCheckMarkDimensions
GetMenuItemCount
GetMenuItemID
GetMenuState
GetMenuString
GetMessage
GetMessagePos
GetMessageTime
GetMetaFile
GetMetaFileBits
GetModuleFileName
GetModuleHandle
GetModuleUsage
GetNearestColor
GetNearestPaletteIndex
GetNextDlgGroupItem
GetNextDlgTabItem
GetNextWindow
GetNumTasks
GetObject
GetPaletteEntries
GetParent
GetPixel
GetPolyFillMode
GetPriorityClipboardFormat
GetPrivateProfileInt
GetPrivateProfileString
GetProcAddress
GetProfileInt
GetProfileString
GetProp
GetRgnBox
GetROP2
GetRValue
GetScrollPos
GetScrollRange
GetStockObject
GetStretchBltMode
GetSubMenu
GetSysColor
GetSysModalWindow
GetSystemDirectory
GetSystemMenu
GetSystemMetrics
GetSystemPaletteEntries
GetSystemPaletteUse
GetTabbedTextExtent
GetTempDrive
GetTempFileName
GetTextAlign
GetTextCharacterExtra

GetTextColor
GetTextExtent
GetTextFace
GetTextMetrics
GetThresholdEvent
GetThresholdStatus
GetTickCount
GetTopWindow
GetUpdateRect
GetUpdateRgn
GetVersion
GetViewportExt
GetViewportOrg
GetWindow
GetWindowDC
GetWindowExt
GetWindowLong
GetWindowOrg
GetWindowRect
GetWindowsDirectory
GetWindowTask
GetWindowText
GetWindowTextLength
GetWindowWord
GetWinFlags
GlobalAddAtom
GlobalAlloc
GlobalCompact
GlobalDeleteAtom
GlobalFindAtom
GlobalFix
GlobalFlags
GlobalFree
GlobalGetAtomName
GlobalHandle
GlobalLock
GlobalLRUNewest
GlobalLRUOldest
GlobalNotify
GlobalPageLock
GlobalPageUnlock
GlobalReAlloc
GlobalSize
GlobalUnfix
GlobalUnlock
GlobalUnWire
GlobalWire
GrayString
HideCaret
HiliteMenuItem
HiWord
InflateRect
InitAtomTable
InSendMessage
InsertMenu

IntersectClipRect
IntersectRect
InvalidateRect
InvalidateRgn
InvertRect
InvertRgn
IsCharAlpha
IsCharAlphaNumeric
IsCharLower
IsCharUpper
IsChild
IsClipboardFormatAvailable
IsDialogMessage
IsDlgButtonChecked
IsIconic
IsRectEmpty
IsWindow
IsWindowEnabled
IsWindowVisible
IsZoomed
KillTimer
_lclose
_lcreat
LimitEmsPages
LineDDA
LineTo
_llseek
LoadAccelerators
LoadBitmap
LoadCursor
LoadIcon
LoadLibrary
LoadMenu
LoadMenuIndirect
LoadModule
LoadResource
LoadString
LocalAlloc
LocalCompact
LocalFlags
LocalFree
LocalHandle
LocalInit
LocalLock
LocalReAlloc
LocalShrink
LocalSize
LocalUnlock
LockData
LockResource
LockSegment
_lopen
LoWord
LPtoDP
_lread

lstrcat
lstrcmp
lstrcmpi
lstrcpy
lstrlen
_lwrite
MakeLong
MakeProcInstance
MapDialogRect
MapVirtualKey
MessageBeep
MessageBox
ModifyMenu
MoveTo
MoveWindow
MulDiv
OemKeyScan
OemToAnsi
OemToAnsiBuff
OffsetClipRgn
OffsetRect
OffsetRgn
OffsetViewportOrg
OffsetWindowOrg
OpenClipboard
OpenComm
OpenFile
OpenIcon
OpenSound
OutputDebugString
PaintRgn
PaletteRGB
PatBlt
PeekMessage
Pie
PlayMetaFile
PlayMetaFileRecord
Polygon
Polyline
PolyPolygon
PostAppMessage
PostMessage
PostQuitMessage
PtInRect
PtInRegion
PtVisible
ReadComm
RealizePalette
Rectangle
RectInRegion
RectVisible
RegisterClass
RegisterClipboardFormat
RegisterWindowMessage
ReleaseCapture

ReleaseDC
RemoveFontResource
RemoveMenu
RemoveProp
ReplyMessage
ResizePalette
RestoreDC
RGB
RoundRect
SaveDC
ScaleViewportExt
ScaleWindowExt
ScreenToClient
ScrollDC
ScrollWindow
SelectClipRgn
SelectObject
SelectPalette
SendDlgItemMessage
SendMessage
SetActiveWindow
SetBitmapBits
SetBitmapDimension
SetBkColor
SetBkMode
SetBrushOrg
SetCapture
SetCaretBlinkTime
SetCaretPos
SetClassLong
SetClassWord
SetClipboardData
SetClipboardViewer
SetCommBreak
SetCommEventMask
SetCommState
SetCursor
SetCursorPos
SetDIBits
SetDIBitsToDevice
SetDlgItemInt
SetDlgItemText
SetDoubleClickTime
SetEnvironment
SetErrorMode
SetFocus
SetHandleCount
SetKeyboardState
SetMapMode
SetMapperFlags
SetMenu
SetMenuItemBitmaps
SetMessageQueue
SetMetaFileBits
SetPaletteEntries

SetParent
SetPixel
SetPolyFillMode
SetProp
SetRect
SetRectEmpty
SetRectRgn
SetResourceHandler
SetROP2
SetScrollPos
SetScrollRange
SetSoundNoise
SetStretchBltMode
SetSwapAreaSize
SetSysColors
SetSysModalWindow
SetSystemPaletteUse
SetTextAlign
SetTextCharacterExtra
SetTextColor
SetTextJustification
SetTimer
SetViewportExt
SetViewportOrg
SetVoiceAccent
SetVoiceEnvelope
SetVoiceNote
SetVoiceQueueSize
SetVoiceSound
SetVoiceThreshold
SetWindowExt
SetWindowLong
SetWindowOrg
SetWindowPos
SetWindowsHook
SetWindowText
SetWindowWord
ShowCaret
ShowCursor
ShowOwnedPopups
ShowScrollBar
ShowWindow
SizeofResource
StartSound
StopSound
StretchBlt
StretchDIBits
SwapMouseButton
SwapRecording
SwitchStackBack
SwitchStackTo
SyncAllVoices
TabbedTextOut
TextOut
Throw

ToAscii
TrackPopupMenu
TranslateAccelerator
TranslateMDISysAccel
TranslateMessage
TransmitCommChar
UngetCommChar
UnhookWindowsHook
UnionRect
UnlockData
UnlockResource
UnlockSegment
UnrealizeObject
UnregisterClass
UpdateColors
UpdateWindow
ValidateCodeSegments
ValidateFreeSpaces
ValidateRect
ValidateRgn
VkKeyScan
WaitMessage
WaitSoundState
WindowFromPoint
WinExec
WinHelp
WriteComm
WritePrivateProfileString
WriteProfileString
wvsprintf
Yield

Kategorischer Index der Windows API Prozeduren und Funktionen

Die API-Routinen von Turbo Pascal für Windows werden in drei Kategorien unterteilt: Fensterverwaltungs-, GDI- (grafische Geräteschnittstelle) und Systemschnittstellen-Routinen.

Fensterverwaltungs-Routinen tun folgendes:
- Fenster erzeugen, verlagern und ändern
- Meldungen bearbeiten
- Systemausgaben erzeugen

GDI-Routinen führen geräteunabhängige Grafikoperationen aus

Systemschnittstellen-Routinen tun folgendes:
- auf Code und Daten in Modulen zugreifen
- Speicherplatz zuweisen und verwalten
- Dateien erzeugen und öffnen
- Geräusche erzeugen
- Programm-Ressourcen laden
- Tasks verwalten
- die Windows-Initialisierungsdatei ändern
- Ein/Ausgabe-Schnittstellen-Kommunikation
- Strings übersetzen

Die API-Routinen können Sie namentlich im **Alphabetischen Index von Windows API Prozeduren und Funktionen** nachschlagen.

Fensterverwaltungs-Routinen

Caret
Clipboard
Cursor
Dialog-Box
Display and Movement
Error
Hardware
Hook
Information
Input
Menu
Message
Painting
Property
Scrolling
System
Window-Creation

GDI-Routinen (Grafische Geräteschnittstelle)

Sie können sich die GDI als Grafik-Engine vorstellen, die von Windows-Anwendungen zur Anzeige und Bearbeitung von Grafiken eingesetzt wird.
Über die Gerätetreiber, die die GDI-Funktionsaufrufe in Befehle übersetzen, die von der Ausgabeeinheit bearbeitet werden können, statten die GDI-Routinen Ihre Anwendungen mit Funktionen zum Zeichnen aus, die unabhäging von der Anzeigeeinheit sind.

Abbildung
Bitmap
Clipping
Druckersteuerung
Ellipse und Polygon
Farbpalette
Gerätekontext
Geräteunhängige Bitmaps
Koordinatenabbildung
Linienzeichnen
Metadatei
Rechteck
Region
Schrift
Text
Umgebung
Zeichnen
Zeichnungsattribute

Systemschnittstellen-Routinen

Anwendungen ausführen
Atome verwalten
Betriebssystem Interrupt
Datei-I/O
Hilfs-Makros
Initialisierungsdatei
Klang
Kommunikation
Modulverwaltung
Ressourcenverwaltung
Segment
Speicherverwaltung
Stringmanipulation
Task

Gerätekontext-Routinen

Ein Gerätekontext repräsentiert einen Gerätetreiber, ein Ausgabegerät und (manchmal) auch die Kommunikationsschnittstelle.
Ihre Anwendung führt Grafikoperationen abhängig von einem bestimmten Gerätekontext aus.

CreateCompatibleDC
CreateDC
CreateIC
DeleteDC
GetDCOrg
RestoreDC
SaveDC

Zeichnen-Routinen

Der Anzeigekontext bestimmt die Bildschirmanzeige von Grafiken. Sie können Grafiken auf unterschiedliche Weise anzeigen lassen, indem Sie die Routinen, die Sie zum Zeichnen der Grafik verwenden, verändern.

- Stifte bestimmen das Aussehen gezeichneter Linien.
- Pinsel bestimmen das Aussehen gefüllter Flächen.
- Schriften bestimmen das Aussehen gezeichneten Textes.

Windows-Programme ordnen einer Zeichnen-Routine Attribute zu, indem Sie eine virtuelle oder in Windows vordefinierte Routinen in einem Anzeigekontext auswählen.

- Eine virtuelle Routine wird von Ihrem Programm durch das Ausfüllen der Felder eines bestimmten Records **(TLogPen**, **TLogBrush** oder **TLogFont)**. erzeugt.
- Eine in Windows vordefinierte Routine respräsentiert die gebräuchlichsten Attributeinstellungen.

Allgemein

DeleteObject
GetBrushOrg
GetObject
SelectObject
SetBrushOrg
UnrealizeObject

Logische Pinsel

CreateBrushIndirect
CreateDIBPatternBrush
CreateHatchBrush
CreatePatternBrush
CreateSolidBrush
EnumObjects

Logische Pinsel

CreatePen
CreatePenIndirect
EnumObjects

Logische Schriften

CreateFont
CreateFontIndirect

In Windows vordefinierte Routine

GetStockObject

Farbpaletten-Routinen

Einige Anzeigegeräte können zwar viele Farben darstellen, aber nur wenige davon gleichzeitig anzeigen.

Die System- oder physikalischen Palette ist die Gruppe bzw. Auswahl an Farben, die aktuell auf einem Anzeigegerät gleichzeitig angezeigt werden kann.

Anwendungen können unter Windows in einem bestimmten Ausmaß steuern, welche Farben in die Systempalette eines Gerätes aufgenommen werden. Falls Ihre Anwendung nur einfache Farben verwendet, müssen Sie die Paletten nicht direkt bearbeiten.

AnimatePalette
CreatePalette
GetNearestPaletteIndex
GetPaletteEntries
GetSystemPaletteEntries
GetSystemPaletteUse
RealizePalette
SelectPalette
SetPaletteEntries
SetSystemPaletteUse
UpdateColors

Wenn Sie die Systempalette modifizieren, werden davon alle Bildschirmausgaben betroffen, auch andere Anwendungen. Eine Anwendung kann also bewirken, daß alle anderen Anwendungen mit den falschen Farben angezeigt werden.

Die Farbpalettenverwaltung von Windows löst dieses Problem, indem zuerst die verschiedenen Anwendungen untersucht werden und aufgrund dieser Daten versucht, die Systempalette zu ändern. Windows ordnet jeder Anwendung eine logische Palette zu (die Gruppe der von der Anwendung benötigten Farben).

Die Farbpalettenverwaltung ordnet die in der logischen Palette geforderten Farben den in der Systempalette verfügbaren Farben zu.

- Falls die geforderte Farbe in der aktuellen Systempalette nicht verzeichnet ist, kann sie in die Systempalette eingefügt werden.
- Wenn die logische Palette mehr Farben spezifiziert als in der Systempalette aufgenommen werden können, werden die überzähligen Farben den in der Systempalette verfügbaren Farben zugeordnet, die ihnen am ehesten entsprechen.

Windows behält 20 permanente Farben in der Systempalette, um das Farbschema für die einzelnen Anwendungen und die Windows-Oberfläche zu wahren.

Zeichnungsattribute-Routinen

Windows Programme schreiben an eine logische Einheit, die Anzeigekontext genannt wird (eine virtuelle Oberfläche mit zugehörigen Attributen). Der Anzeigekontext verwaltet die Bildschirmanzeige von Grafiken. Sie können die Anzeigekontrextattribute nicht bestimmen.

Allerdings können Sie bestimmte Zeichnungsatttribute ändern, wie Hintergrundfarbe und Modus, Polygonfüllmodus, Zeichnungsmodus (Kombination von Farben), Bitmapmanipulationsmodus und Vordergrundfarbe für Text.

Wenn Sie die Attribute von Grafikdarstellungen ändern wollen, können Sie auch die Grafik-Routinen ändern, mit denen Sie Grafiken definieren. Die Attribute dieser Routinen legen das Erscheinungsbild von Grafiken fest, die mit Hilfe der GDI-Funktionen gezeichnet werden.

(Nähere Angaben finden Sie im Hilfebildschirm **Zeichnen-Routinen**)

GetBkColor
GetBkMode
GetPolyFillMode
GetROP2
GetStretchBltMode
GetTextColor
SetBkColor
SetBkMode
SetPolyFillMode
SetROP2
SetStretchBltMode
SetTextColor

Abbildungs-Routinen

Abbildungsmodi beschreiben die relative Größe, Position aud Ausrichtung der logischen Koordinaten in Bezug auf die Bildschirmkoordinaten.

GetMapMode
GetViewportExt
GetViewportOrg
GetWindowExt
GetWindowOrg
OffsetViewportOrg
OffsetWindowOrg
ScaleViewportExt
ScaleWindowExt
SetMapMode
SetViewportExt
SetViewportOrg
SetWindowExt
SetWindowOrg

Die acht verschiedenen Abbildungsmodi werden im Hilfebildschirm **mm_xxx** erklärt.

Koordinatenabbildungs-Routinen

Manchmal ist es notwendig, logische Koordinaten (die zum Zeichnen verwendet werden) in physikalische Bitmap-Koordinaten (in die der aktuelle Abbildungsmodus logische Koordinaten übersetzt) zu übersetzen.

GDI enthält folgende Funktionen zur Bearbeitung solcher Koordinatenabbildungen:

ChildWindowFromPoint
ClientToScreen
DPtoLP
LPtoDP
ScreenToClient
WindowFromPoint

Region-Routinen

Eine Region ist eine Fläche (ein Polygon oder eine Ellipse) innerhalb eines Fensters. Regionen können mit Grafikausgaben gefüllt werden.

Region-Routinen erzeugen, modifizieren und erhalten Informationen über Regionen.

CombineRgn
CreateEllipticRgn
CreateEllipticRgnIndirect
CreatePolygonRgn
CreatePolyPolygonRgn
CreateRectRgn
CreateRectRgnIndirect
CreateRoundRectRgn
EqualRgn
FillRgn
FrameRgn
GetRgnBox
InvertRgn
OffsetRgn
PaintRgn
PtInRegion
RectInRegion
SetRectRgn

Clipping-Routinen

Um sicherzustellen, daß nur innerhalb des vorgesehenen Bereichs gezeichnet wird, verfügt jeder Anzeigekontext über Clipping-Attribute.

ExcludeClipRect
GetClipBox
IntersectClipRect
OffsetClipRgn
PtVisible
RectVisible
SelectClipRgn

Routinen zum Zeichnen von Linien

Die Routinen zum Zeichnen von Linien verwenden den im Anzeigekontext definierten aktuellen Stift zum Zeichnen.

Arc
LineDDA
LineTo
MoveTo
Polyline

Routinen zum Zeichnen von Flächen

Diese Routinen verwenden den im Anzeigekontext definierten aktuellen Stift, um den Umriß zu zeichnen, und den aktuellen Pinsel, um die damit definierte Fläche auszufüllen. Sie haben keinen Einfluß auf die aktuelle Position.

Chord
DrawFocusRect
Ellipse
Pie
Polygon
PolyPolygon
Rectangle
RoundRect

Bitmap-Routinen

Die aktuelle Oberfläche eines Anzeigekontextes wird Bitmap genannt. Da Bitmaps die Speicherkonfiguration eines bestimmten Gerätes repräsentieren, unterscheiden sie sich abhängig vom Typ des adressierten Gerätes.
Die GDI gibt Ihnen einige Techniken zur Handhabung dieses an die Hand, zu diesen Techniken gehören beispielsweise **geräteunabhängige Bitmaps**.

BitBlt
CreateBitmap
CreateBitmapIndirect
CreateCompatibleBitmap
CreateDiscardableBitmap
ExtFloodFill
FloodFill
GetBitmapBits
GetBitmapDimension
GetPixel
LoadBitmap
PatBlt
SetBitmapBits
SetBitmapDimension
SetPixel
StretchBlt

Geräteunabhängige Bitmap-Routinen

Da Bitmaps die Speicherkonfiguration eines bestimmten Gerätes repräsentieren, sind sie vom Typ des adressierten Gerätes abhängig.
Diese Abhängigkeit hat zur Folge, daß Bitmaps, die von einem Gerät gespeichert wurden, inkompatibel zu einem anderen Gerät sind.

Die GDI stellt zur Handhabung dieses Problems geräteunabhängige Bitmaps zur Verfügung. Folgende Routinen sind zur Bearbeitung geräteunabhängiger Bitmaps verfügbar:

CreateDIBitmap
GetDIBits
SetDIBits
SetDIBitsToDevice
StretchDIBits

Routinen zum Zeichen von Text

Die Routinen zum Zeichnen von Text verwenden die im Gerätekontext spezifierte aktuelle Schrift.

ExtTextOut
GetTabbedTextExtent
GetTextAlign
GetTextExtent
GetTextFace
GetTextMetrics
SetTextAlign
SetTextJustification
TabbedTextOut
TextOut

Schrift-Routinen

Eine Schrift ist eine Teilmenge einer bestimmten Schriftart. Schrift-Routinen erzeugen, selektieren, entfernen und lesen Informationen zu Schriften.

AddFontResource
CreateFont
CreateFontIndirect
EnumFonts
GetCharWidth
RemoveFontResource
SetMapperFlags

Metadatei-Routinen

Eine Metadatei ist eine Gruppe von GDI-Befehlen, die Text oder Bilder erzeugen. Metadatei-Routinen erzeugen, kopieren, schließen, löschen, laden und lesen Informationen über Metadateien.

CloseMetaFile
CopyMetaFile
CreateMetaFile
DeleteMetaFile
EnumMetaFile
GetMetaFile
GetMetaFileBits
PlayMetaFile
PlayMetaFileRecord
SetMetaFileBits

Druckersteuerungs-Routinen

DeviceCapabilities

Umgebungs-Routinen

GetEnvironment
SetEnvironment

Modulverwaltungs-Routinen

FreeLibrary
FreeModule
FreeProcInstance
GetCodeHandle
GetInstanceData
GetModuleFileName
GetModuleHandle
GetModuleUsage
GetProcAddress
GetVersion
LoadLibrary
MakeProcInstance

Speicherverwaltungs-Routinen

DefineHandleTable
GetFreeSpace
GetWinFlags
GlobalAlloc
GlobalCompact
GlobalDiscard
GlobalDosAlloc
GlobalDosFree
GlobalFlags
GlobalFree
GlobalHandle
GlobalLock
GlobalLRUNewest
GlobalLRUOldest
GlobalNotify
GlobalReAlloc
GlobalSize
GlobalUnlock
GlobalUnwire
GlobalWire
LimitEMSPages
LocalAlloc
LocalCompact
LocalDiscard
LocalFlags
LocalFree
LocalHandle
LocalInit
LocalLock
LocalReAlloc
LocalShrink
LocalSize
LocalUnlock
LockData
LockSegment
SetSwapAreaSize
SwitchStackBack
SwitchStackTo
UnlockData
UnLockSegment

Segment-Routinen

AllocDStoCSAlias
AllocSelector
ChangeSelector
DefineHandleTable
FreeSelector
GetCodeInfo
GlobalFix
GlobalPageLock
GlobalPageUnlock
GlobalUnfix
LockSegment
UnlockSegment

Betriebssystem-Interrupt-Routinen

DOS3Call
NetBIOSCall

Task-Routinen

Catch
ExitWindows
GetCurrentPDB
GetCurrentTask
GetDOSEnvironment
GetNumTasks
SetErrorMode
Throw
Yield

Ressourcenverwaltungs-Routinen

AccessResource
AllocResource
FindResource
FreeResource
LoadAccelerators
LoadBitmap
LoadCursor
LoadIcon
LoadMenu
LoadResource
LoadString
LockResource
SetResourceHandler
SizeofResource
UnlockResource

Stringmanipulations-Routinen

AnsiLower
AnsiLowerBuff
AnsiNext
AnsiPrev
AnsiToOem
AnsiToOemBuff
AnsiUpper
AnsiUpperBuff
IsCharAlpha
IsCharAlphaNumeric
IsCharLower
IsCharUpper
lstrcat
lstrcmp
lstrcmpi
lstrcpyb
lstrlen
OemToAnsi
OemToAnsiBuff
ToAscii
wvsprintf

Atomverwaltungs-Routinen

AddAtom
DeleteAtom
FindAtom
GetAtomHandle
GetAtomName
GlobalAddAtom
GlobalDeleteAtom
GlobalFindAtom
GlobalGetAtomName
InitAtomTable
MAKEINTATOM

Initialisierungsdatei-Routinen

GetPrivateProfileInt
GetPrivateProfileString
GetProfileInt
GetProfileString
WritePrivateProfileString
WriteProfileString

Kommunikations-Routinen

BuildCommDCB
ClearCommBreak
CloseComm
EscapeCommFunction
FlushComm
GetCommError
GetCommEventMask
GetCommState
OpenComm
ReadComm
SetCommBreak
SetCommEventMask
SetCommState
TransmitCommChar
UngetCommChar
WriteComm

Klang-Routinen

CloseSound
CountVoiceNotes
GetThresholdEvent
GetThresholdStatus
OpenSound
SetSoundNoise
SetVoiceAccent
SetVoiceEnvelope
SetVoiceNote
SetVoiceQueueSize
SetVoiceSound
SetVoiceThreshold
StartSound
StopSound
SyncAllVoices
WaitSoundState

Hilfs-Routinen

GetNearestPaletteIndex
HIBYTE
HIWORD
LOBYTE
LOWORD
MAKEINTATOM
MAKEINTRESOURCE
MAKELONG
MAKEPOINT
MulDiv
PALETTERGB
RGB

Datei-I/O-Routinen

GetDriveType
GetSystemDirectory
GetTempDrive
GetTempFileName
GetWindowsDirectory
_lclose
_lcreat
_llseek
_lopen
_lread
_lwrite
OpenFile
SetHandleCount

Anwendungsaufruf--Routinen

LoadModule
WinExec
WinHelp

Botschafts-Routinen

CallWindowProc
DispatchMessage
GetMessage
GetMessagePos
GetMessageTime
InSendMessage
PeekMessage
PostAppMessage
PostMessage
PostQuitMessage
ReplyMessage
SendMessage
SetMessageQueue
TranslateAccelerator
TranslateMDISysAccel
TranslateMessage
WaitMessage

Fenster-Routinen

AdjustWindowRect
AdjustWindowRectEx
CreateWindow
CreateWindowEx
DefDlgProc
DefFrameProc
DefMDIChildProc
DefWindowProc
DestroyWindow
GetClassInfo
GetClassLong
GetClassName
GetClassWord
GetLastActivePopup
GetWindowLong
GetWindowWord
RegisterClass
SetClassLong
SetClassWord
SetWindowLong
SetWindowWord
UnregisterClass

κAnzeige- und Cursorbewegungs-Routinen

ArrangeIconicWindows
BeginDeferWindowPos
BringWindowToTop
CloseWindow
DeferWindowPos
EndDeferWindowPos
GetClientRect
GetWindowRect
GetWindowText
GetWindowTextLength
IsIconic
IsWindowVisible
IsZoomed
MoveWindow
OpenIcon
SetWindowPos
SetWindowText
ShowOwnedPopups
ShowWindow

Eingabe-Routinen

EnableWindow
GetActiveWindow
GetCapture
GetCurrentTime
GetDoubleClickTime
GetFocus
GetTickCount
IsWindowEnabled
KillTimer
ReleaseCapture
SetActiveWindow
SetCapture
SetDoubleClickTime
SetFocus
SetSysModalWindow
SetTimer
SwapMouseButton

Hardware-Routinen

EnableHardwareInput
GetAsyncKeyState
GetInputState
GetKeyboardState
GetKeyNameText
GetKeyState
GetKBCodePage
OemKeyScan
SetKeyboardState
MapVirtualKey
VkKeyScan

Mal-Routinen

BeginPaint
DrawFocusRect
DrawIcon
DrawText
EndPaint
ExcludeUpdateRgn
FillRect
FrameRect
GetDC
GetUpdateRect
GetUpdateRgn
GetWindowDC
GrayString
InvalidateRect
InvalidateRgn
InvertRect
ReleaseDC
UpdateWindow
ValidateRect
ValidateRgn

Dialogfenster-Routinen

CheckDlgButton
CheckRadioButton
CreateDialog
CreateDialogIndirect
CreateDialogIndirectParam
CreateDialogParam
DefDlgProc
DialogBox
DialogBoxIndirect
DialogBoxIndirectParam
DialogBoxParam
DlgDirList
DlgDirListComboBox
DlgDirSelect
DlgDirSelectComboBox
EndDialog
GetDialogBaseUnits
GetDlgCtrlID
GetDlgItem
GetDlgItemInt
GetDlgItemText
GetNextDlgGroupItem
GetNextDlgTabItem
IsDialogMessage
IsDlgButtonChecked
MapDialogRect
SendDlgItemMessage
SetDlgItemInt
SetDlgItemText

Bildlauf-Routinen

GetScrollPos
GetScrollRange
ScrollDC
ScrollWindow
SetScrollPos
SetScrollRange
ShowScrollBar

Menü-Routinen

AppendMenu
CheckMenuItem
CreateMenu
CreatePopupMenu
DeleteMenu
DestroyMenu
DrawMenuBar
EnableMenuItem
GetMenu
GetMenuCheckMarkDimensions
GetMenuItemCount
GetMenuItemID
GetMenuState
GetMenuString
GetSubMenu
GetSystemMenu
HiliteMenuItem
InsertMenu
LoadMenuIndirect
ModifyMenu
RemoveMenu
SetMenu
SetMenuItemBitmaps
TrackPopupMenu

Informations-Routinen

AnyPopup
ChildWindowFromPoint
EnumChildWindows
EnumTaskWindows
EnumWindows
FindWindow
GetNextWindow
GetParent
GetTopWindow
GetWindow
GetWindowTask
IsChild
IsWindow
SetParent
WindowFromPoint

System-Routinen

GetCurrentTime
GetSysColor
GetSystemMetrics
SetSysColors

Zwischenablage-Routinen

ChangeClipboardChain
CloseClipboard
EmptyClipboard
EnumClipboardFormats
GetClipboardData
GetClipboardFormatName
GetClipboardOwner
GetClipboardViewer
GetPriorityClipboardFormat
IsClipboardFormatAvailable
OpenClipboard
RegisterClipboardFormat
SetClipboardData
SetClipboardViewer

Fehlerbehandlungs-Routinen

FlashWindow
MessageBeep
MessageBox

Caret-Routinen

CreateCaret
DestroyCaret
GetCaretBlinkTime
GetCaretPos
HideCaret
SetCaretBlinkTime
SetCaretPos
ShowCaret

Cursor-Routinen

ClipCursor
CreateCursor
DestroyCursor
GetCursorPos
LoadCursor
SetCursor
SetCursorPos
ShowCursor

Filter-Routinen

CallMsgFilter
DefHookProc
SetWindowsHook
UnhookWindowsHook

Eigenschafts-Routinen

EnumProps
GetProp
RemoveProp
SetProp

Rechteck-Routinen

CopyRect
EqualRect
InflateRect
IntersectRect
OffsetRect
PtInRect
SetRectEmpty
UnionRect

Index der Prozeduren und Funktionen von Turbo Pascal

Abs	Funktion	Liefert den absoluten Wert des Arguments zurück
Addr	Funktion	Liefert die Adresse des angegebenen Objekts
Append	Prozedur	Öffnet eine exisierende Datei für das Anhängen weiterer Daten
ArcTan	Funktion	Liefert den Arcustangens des Arguments zurück
Assign	Prozedur	Ordnet einer Datei-Variablen eine externe Datei zu
BlockRead	Prozedur	Liest einen oder mehrere Records in eine Puffervariable
BlockWrite	Prozedur	Schreibt einen oder mehrere Records aus einer Puffervariablen
Chr	Funktion	Liefert das Zeichen, dessen ASCII-Code dem Wert von x entspricht
Close	Prozedur	Schließt eine offene Datei.
Concat	Funktion	Verbindet zwei oder mehrere Stringteile
Copy	Funktion	Liefert einen Teil eines Strings zurück
Cos	Funktion	Liefert den Cosinus des Arguments zurück
CSeg	Funktion	Liefert die Adresse des aktuellen Code-Segments (CS-Register) zurück
Dec	Prozedur	Zählt eine Variable um einen bestimmten Wert herunter
Delete	Prozedur	Löscht count Zeichen ab der Position index im String s
Dispose	Prozedur	Gibt den einer Zeigervariablen zugeordneten Speicherplatz auf dem Heap wieder frei
DSeg	Funktion	Liefert die Adresse des Daten-Segments (d.h. den Inhalt des DS-Registers) zurück
Eof	Funktion	Prüft, ob das Ende der Datei erreicht ist
Eoln	Funktion	Prüft, ob das Zeilenende in einer Textdatei erreicht ist
Erase	Prozedur	Löscht eine externe Datei
Exit	Prozedur	Verläßt den momentanen Block mit sofortiger Wirkung
Exp	Funktion	Liefert "e hoch" dem Argument zurück
FilePos	Funktion	Liefert die aktuelle Position innerhalb einer Datei
FileSize	Funktion	Liefert die Größe einer Datei zurück
FillChar	Prozedur	Füllt count aufeinanderfolgende Speicherzellen mit dem Wert ch (vomTyp Byte oder Char)
Flush	Prozedur	Erzwingt das physikalische Schreiben des Puffers einer Textdatei, die für Ausgaben geöffnet wurde
Frac	Funktion	Liefert den nicht-ganzzahligen Teil des Arguments zurück
FreeMem	Prozedur	Gibt einen Speicherbereich bestimmter Größe auf dem Heap wieder frei
GetDir	Prozedur	Ermittelt das momentan gesetzte Verzeichnis eines Laufwerks
GetMem	Prozedur	Erzeugt eine dynamische Variable und weist die Startadresse des Speicherblocks einer Zeigervariablen
Halt	Prozedur	Bricht die Ausführung des Programms ab und führt einen Rücksprung zur DOS-Kommandoebene durch
Hi	Funktion	Liefert das höherwertige Byte des Arguments zurück
Inc	Prozedur	Erhöht eine Variable um 1 bzw. den durch n angegebenen Wert
Insert	Prozedur	Fügt einen Teilstring ab der Position "index" in die als "s" übergebene Stringvariable ein
Int	Funktion	Liefert den ganzzahligen Anteil des Arguments zurück
IOResult	Funktion	Liefert den Status der letzten Ein-/Ausgabe-Operation als Integer zurück
Length	Funktion	Liefert die aktuelle Länge eines Stringausdrucks zurück
Ln	Funktion	Liefert den natürlichen Logarithmus des Arguments
Lo	Funktion	Liefert das niederwertige Byte des Arguments zurück
MaxAvail	Funktion	Liefert die Umfang des größten freien Blocks auf dem Heap in Bytes zurück
MemAvail	Funktion	Liefert den Gesamtumfang des freien Platzes auf dem Heap in

		Bytes
MkDir	Prozedur	Erzeugt ein Unterverzeichnis
Move	Prozedur	Kopiert Bytes von Quelle zu Bestimmung
New	Prozedur	Erzeugt eine neue dynamische Variable und setzt eine Zeigervariable auf die Startadresse dieses Bereichs
Odd	Funktion	Prüft, ob das Argument eine ungerade Zahl darstellt
Ofs	Funktion	Liefert den Offset-Anteil der Adresse des angegebenen Objekts zurück
Ord	Funktion	Liefert die Ordinalzahl des Arguments zurück
ParamCount	Funktion	Liefert die Anzahl der Kommandozeilen-Parameter zurück, die dem Programm übergeben wurden
ParamStr	Funktion	Liefert den Kommandozeilen-Parameter [Index] zurück.
Pi	Funktion	Liefert den Wert der mathematischen Konstanten Pi zurück
Pos	Funktion	Sucht den als s übergebenen String-Ausdruck nach dem ersten Vorkommen des String-Ausdrucks substr ab
Pred	Funktion	Liefert den Vorgänger des Arguments zurück
Ptr	Funktion	Konvertiert zwei Angaben für Segment und Offset in einen Wert des Typs Pointer
Random	Funktion	Liefert eine Zufallszahl
Randomize	Prozedur	Initialisiert den "Zufallsgenerator" mit einem Wert, der sich aus Datum und Uhrzeit des Systems ergibt
Read	Prozedur	Bei typisierten Dateien: Liest eine oder mehrere Komponenten in eine Variable. Bei Textdateien: Liest einen oder mehrere Werte in ein oder mehrere Variablen
ReadLn	Prozedur	Führt einen Aufruf von Read aus und springt dann zum Anfang der nächsten Zeile
Rename	Prozedur	Gibt einer externen Datei einen neuen Namen
Reset	Prozedur	Öffnet eine existierende Datei
Rewrite	Prozedur	Erzeugt eine neue Datei und öffnet sie
Round	Funktion	Rundet das (Real-)Argument auf einen ganzzahligen Wert
RunError	Prozedur	Erzeugt einen Laufzeitfehler, der das Programm definiert abbricht
Seek	Prozedur	Bewegt die Corsorposition in einer Datei zur angegebenen Komponente
SeekEof	Funktion	Prüft, ob sich zwischen der momentanen Position und dem Dateiende noch "lesbare" Daten befinden
SeekEoln	Funktion	Prüft, ob sich zwischen der momentanen Position und dem nächsten Zeilenende "lesbare" Daten befinden
Seg	Funktion	Liefert die Segmentadresse des angegebenen Objekts zurück
SetTextBuf	Prozedur	Weist einer Textdatei-Variablen einen Puffer zu
Sin	Funktion	Liefert den Sinus des Arguments
SizeOf	Funktion	Liefert die Anzahl von Bytes zurück, die das Argument an Speicherplatz belegt
SPtr	Funktion	Liefert den momentanen Wert des Stackzeigers (SP-Register) zurück
Sqr	Funktion	Liefert das Quadrat des Arguments zurück
Sqrt	Funktion	Liefert die Quadratwurzel des Arguments zurück
SSeg	Funktion	Liefert die Adresse des Stack-Segments zurück, d.h. den Wert des SS-Registers
Str	Prozedur	konvertiert einen numerischen Wert in eine Zeichenfolge
Succ	Funktion	Liefert den Nachfolger des Arguments zurück
Swap	Funktion	Vertauscht das niederwertige und das höherwertige Byte des Arguments
Trunc	Funktion	Wandelt einen Realwert durch Abschneiden der Nachkommastellen in einen Integerwert um

Truncate	Prozedur	Schneidet eine Datei an der aktuellen Position ab
UpCase	Funktion	Konvertiert Klein- in Großbuchstaben. Die deutschen Umlaute werden nicht berücksichtigt
Val	Prozedur	Interpretiert die Zeichenfolge eines Strings als numerischen Wert
Write	Prozedur	Bei typisierten Dateien: Schreibt eine Variable in eine Dateikomponente. Bei Textdateien: Schreibt einen oder mehrere Werte in die Datei
WriteLn	Prozedur	Ruft Write (für Textdateien) auf und schreibt dann einen Zeilenvorschub in die Datei

Compiler-Befehle

Compiler-Befehle sind Kommentare, die einer bestimmten Syntax genügen müssen. Sie

- beginnen mit `{$` oder `(*$`
- enthalten den Namen des Befehls
- enden mit `}` oder `*)`

Es gibt drei Formen von Compiler-Befehlen:

- Mit einem **Schalterbefehl** werden bestimmte Funktionen des Compilers ein- oder ausgeschaltet. Auf den Namen des Befehls folgt dementsprechend ein Minus (-) oder ein Pluszeichen (+).
- **Parameterbefehle** legen Werte und Dateinamen fest.
- **Bedingte Anweisungen** steuern die bedingte Compilierung von Teilen des Programmcodes.

Eine Zusammenfassung der Compiler-Befehle zusammen mit den Optionen im Dialogfenster Compiler Options finden Sie im Hilfeschirm **Befehle und IDE-Äquivalente**.

Schalterbefehle

Schalter schalten Compilerfunktionen an oder aus.

Schalter können sowohl lokal als auch global wirken:
- Lokale Schalter können an jeder beliebigen Stelle des Programms stehen. Sie betreffen nur einen Teil der Compilierung.
- Globale Schalter müssen grundsätzlich nach dem einleitenden PROGRAM oder UNIT und vor jedem anderen Teil des Programms stehen. Sie betreffen die gesamte Compilierung.

Schalter	Bedeutung
$A	Ausrichtung der Variablen im Speicher
$B	Auswertung boolescher Ausdrücke
$D	Zusatzinformationen für den Debugger
$F	Erzwingen von FAR-Aufrufen
$G	Codeerzeugung für 80286
$I	Automatische Prüfung von Ein-/Ausgaben
$L	Lokale Symbole
$N	Numerischer Koprozessor
$R	Bereichsüberprüfung
$S	Stack-Prüfung
$V	Überprüfen von Var-Strings
$W	Windows-Stackrahmen
$X	Erweiterte Syntax von Turbo Pascal

Turbo Pascal erlaubt es, mehrere Schalter in einer Kommentarklammer zusammenzufassen, solange sie durch Kommata getrennt sind, z.B.

```
{$F,$R,$E,$D}
```

Parameterbefehle

Parameterbefehle legen Werte und Dateinamen zur Compilierung fest.

Parameter-Befehle verlangen nach mindestens einem Leerzeichen zwischen dem Befehlsnamen und ihrem Parameter, z.B.

```
$I TYPES.INC
$O MOONUNIT.TPU
```

Parameterbefehl	Bedeutung
$C Attribute	Code Segment Attribute
$D Text	Beschreibung
$I Dateiname	Include-Datei
$L Dateiname	Einbinden von Object-Dateien
$M Stackgröße,Heapgröße	Größe der Speicherbelegung
$R Dateiname	Ressourcen-Datei

Bedingte Compilierung: Befehle und Symbole

Die bedingte Compilierung basiert auf der Auswertung von Bedingungen.

Symbol	Bedeutung
CPU86	CPU gehört zur Familie der 80x86 Prozessoren
CPU87	Nur definiert, wenn ein 8087 numerischer Koprozessor zur Übersetzungszeit installiert ist
DEFINE	Definiert ein bedingtes Symbol
ELSE	Compilieren oder Ignorieren von Teilen des Quelltexts
ENDIF	Ende der bedingten Compilierung
IFDEF	Übersetzen des Quelltexts, wenn Name definiert ist
IFNDEF	Übersetzen des Quelltexts, wenn Name NICHT definiert ist
IFOPT	Übersetzen des Quelltexts, wenn der angegebene Compiler-Schalter in einen bestimmten Status hat (+ or -)
UNDEF	Löscht bereits definertes Symbol
VER10	Versionsnummer von Turbo Pascal für Windows
WINDOWS	Gibt Betriebssystemumgebung für MS-Windows an

Siehe auch:

Bedingte Compilierung: Konstrukte

Bedingte Compilierung: Konstrukte

Turbo Pascal stellt zwei Konstrukte zur Verfügung, mit denen Quelltextteile von der Compilierung ausgeschlossen bzw. in die Compilierung einbezogen werden können:

- `$IFxxx ... $ENDIF`
- `$IFxxx ... $ELSE ... $ENDIF`

IF ... ENDIF

Der von {$IFxxx} und {$ENDIF} eingeschlossene Quelltext wird nur dann übersetzt, wenn die in {$IFxxx} angegebene Bedingung zutrifft (True). Wenn sie nicht zutrifft (False), wird er ignoriert.

IF ... ELSE ... ENDIF

Das Konstrukt {$IFxxx} ... {$ELSE} ... {$ENDIF}
hat folgende Wirkung:

- Trifft die in {$IFxxx} angegebene Bedingung zu (True), wird der Quelltext zwischen {$IFxxx} und {$ELSE} übersetzt.
- Trifft die in {$IFxxx} angegebene Bedingung nicht zu (False), wird der Quelltext zwischen {$ELSE} und {$ENDIF} übersetzt.

Befehle zur bedingten Compilierung dürfen bis zu einer Tiefe von 16 Ebenen geschachtelt werden.

Jede Quelltextdatei muß genauso viele {$IFxxx} wie {$ENDIF} Befehle enthalten.

Siehe auch:

$ELSE
$ENDIF
$IFDEF
$IFNDEF

Befehle und IDE-Äquivalente

Befehl	Entsprechung im Dialogfenster Compiler Options (Compiler-Optionen)
$A+	**Word Align Data (Datenausrichtung)..Word**
$A-	**Align Data (Datenausrichtung)..Byte**
$B+	**Boolean Evaluation (Boolesche Auswertung)..Complete (Vollständig)**
$B-	**Boolean Evaluation (Boolesche Auswertung)..Short circuit (Kurzschluß)**
$D+	**Debug Information..On (Ein)**
$D-	**Debug Information..Off (Aus)**
$F+	**Force Far Calls (Far-Aufrufe erzeugen)..On (Ein)**
$F-	**Force Far Calls (Far-Aufrufe erzeugen)..Off (Aus)**
$G+	**286 Code..On (Ein)**
$G-	**286 Code..Off (Aus)**
$I+	**I/O Checking (I/O-Prüfung)..On (Ein)**
$I-	**I/O Checking (I/O-Prüfung)..Off (Aus)**
$L+	**Local Symbols (Lokale Symbole)..On (Ein)**
$L-	**Local Symbols (Lokale Symbole)..Off (Aus)**
$M	**Memory Sizes (Speichergrößen)..(Größenangabe)**
$N+	**80x87 Code..On (Ein)**
$N-	**80x87 Code..Off (Aus)**
$R+	**Range Checking (Bereichsüberprüfung)..On (Ein)**
$R-	**Range Checking (Bereichsüberprüfung)..Off (Aus)**
$S+	**Stack Checking (Stack-Prüfung)..On (Ein)**
$S-	**Stack Checking (Stack-Prüfung)..Off (Aus)**
$V+	**String Var Checking (String-Prüfung)..Strict (Streng)**
$V-	**String Var Checking (String-Prüfung)..Relaxed (Locker)**
$W+	**Windows Stack Frame..On (Ein)**
$W-	**Windows Stack Frame..Off (Aus)**
$X+	**Extended Syntax (Erweiterte Syntax)..On (Ein)**
$X-	**Extended Syntax (Erweiterte Syntax)..Off (Aus)**

Der bedingte Compiler-Befehl DEFINE

Definiert das durch Name angegebene Symbol

Syntax

```
$DEFINE Name
```

Das definierte Symbol ist gültig bis zum Ende der Compilierung, oder bis es durch einen entsprechenden Befehl {$**UNDEF** Name} gelöscht wird.

Wenn der Name vor diesem Befehl bereits definiert war, hat {$DEFINE Name} keine Wirkung.

Der bedingte Compiler-Befehl UNDEF

Löscht das durch Name bezeichnete Symbol.

Syntax

```
$UNDEF Name
```

Das Symbol ist bis zu seiner erneuten Definition durch {**$DEFINE** Name} oder bis zum Ende der Compilierung ungültig.

{$UNDEF Name}hat keine Wirkung, wenn das Symbol Name vorher nicht definiert wurde.

Der bedingte Compiler-Befehl IFDEF

Bewirkt die Übersetzung des folgenden Quelltexts, wenn das bezeichnete Symbol Name definiert ist.

Syntax

```
$IFDEF Name
```

Siehe auch:

$IFNDEF
$IFOPT
$ELSE
$ENDIF

Der bedingte Compiler-Befehl IFNDEF

Der nachfolgende Quelltext wird nur übersetzt, wenn das Symbol Name NICHT definiert ist.

Syntax

```
$IFNDEF Name
```

Siehe auch:

$IFDEF
$IFOPT
$ELSE
$ENDIF

Der bedingte Compiler-Befehl IFOPT

Bestimmt die Compilierung des nachfolgenden Quelltextes abhängig vom Stand eines Schalters.

Syntax

```
$IFOPT Schalter
```

Schalter ist der Name eines Compiler-Schalters, gefolgt von einem Plus- oder Minuszeichen.

`Schalter+`	Schalter ist AN
`Schalter-`	Schalter ist AUS

Siehe auch:

$IFDEF
$IFNDEF
$ELSE
$ENDIF

Der bedingte Compiler-Befehl ELSE

Compiliert oder ignoriert den folgenden Programmteil.

Syntax

```
$ELSE
```

Im Programmcode zwischen $IFDEF (oder $IFNDEF) und $ENDIF bewirkt $ELSE das Compilieren des folgenden Programmteils, wenn die $IFDEF (oder $IFNDEF)-Bedingung NICHT zutrifft.

Trifft die $IFDEF (oder $IFNDEF)-Bedingung zu, wird der folgende Programmteil ignoriert.

Siehe auch:

$IFDEF
$IFNDEF
$IFOPT
$ENDIF

Der bedingte Compiler-Befehl ENDIF

Schließt ein mit {$IFxxx} begonnenes Konstrukt ab.

Syntax

```
$ENDIF
```

Siehe auch:

$IFDEF
$IFNDEF
$IFOPT
$ELSE

Das Symbol zur bedingten Compilierung VER10

Turbo Pascal für Windows definiert das Symbol VER10, das es als Turbo Pascal Window-Version 1.0 ausweist.

Das bedingte Symbol WINDOWS

Turbo Pascal für Windows definiert das Symbol WINDOS, um zu kennzeichnen, daß dieser Compiler das Betriebssystem MS-Windows voraussetzt.

Versionen von Turbo Pascal, die unter anderen Betriebssystemen arbeiten, definieren ein entsprechendes Symbol.

CPU86 Symbol zur bedingten Compilierung

Turbo Pascal definiert immer das Symbol CPU86, um anzuzeigen, daß der Compiler einen Prozessor der 80x86 Prozessorfamilie voraussetzt.

CPU87 Symbol zur bedingten Compilierung

Das Symbol CPU87 wird nur definiert, wenn auch ein 80x87 Coprozessor zur Übersetzungszeit vorhanden ist.

Über folgendes Konstrukt läßt sich das Setzen des Compiler-Schalters $N automatisieren:

```
$IFDEF CPU87 $N+ $ELSE $N- $ENDIF
```

Turbo Pascal wählt automatisch die richtige Gleitkommagenerierung für Ihren Computer.

$A Der Schalter Align Data

Wechselt zwischen Byte- und Word-Ausrichtung von Variablen und typisierten Konstanten.

Syntax: `$A+ oder $A-`
Voreinstellung: A+
Typ: Global
Kommandozeile: `/$A+` oder `/$A-`
Menüäquivalent: Options|Compiler|**Align Data**

Anmerkung

Die Ausrichtung auf Word beschleunigt Zugriffe auf den 80x86 Prozessoren, sie hat keine Auswirkung bei einem 8088-Prozessor.

- Zugriff auf Elemente von Word-Größe an geraden Adressen erfolgt in einem Rechnerzyklus
- Zugriff auf Elemente von Word-Größe an ungeraden Adressen erfolgt in zwei Rechnerzyklen

Der Modus $A+

Im $A+ Modus werden alle Variablen und typisierten Konstanten, die größer als 1 Byte sind, an Word-Grenzen ausgerichtet (an einer geraden Adresse).

Falls nötig, werden ungenutzte Bytes zwischen den Variablen eingefügt, um die Ausrichtung auf Word zu erreichen.

{$A+} hat keine Wirkung bei Variablen von Bytegröße, Record-Feldern und Elementen von Arrays.

Ein Record-Feld wird nur dann auf Word ausgerichtet, wenn die Gesamtgröße aller Felder davor eine gerade Zahl ergibt.
Array-Elemente werden nur dann auf Word ausgerichtet, wenn die Größe des einzelnen Elements gerade ist.

Der Modus $A-

Im Modus {$A-} findet keine Ausrichtung statt.

Variablen und typisierten Konstanten werden unabhängig von ihrer Größe an der nächsten verfügbaren Adresse plaziert.

Hinweis: Ungeachtet des Standes des Schalters {$A} werden VAR- und CONST-Deklarationen vom Linker grundsätzlich nur an geraden Adressen begonnen.

Gleiches gilt für den Stackpointer (SP), der immer eine gerade Adresse enthält, bei Bedarf wird dem Stack-Frame einer Prozedur ein Leerbyte hinzugefügt.

$B Auswertung boolescher Ausdrücke

Dieser Schalter legt fest, wie boolesche Ausdrücke, die die **Operatoren AND oder OR** verwenden, ausgewertet werden.

Syntax: `$B+` oder `$B-`
Voreinstellung: `$B-`
Typ: Lokal
Kommandozeile: `/$B+` or `/$B-`
Menüäquivalent: Options|Compiler|**Boolean Evaluation**

Der Modus $B+

Im Modus {$B+} erzeugt der Compiler Code für eine vollständige Auswertung eines booleschen Ausdrucks.

Das bedeutet, daß sämtliche Teile eines mit AND und OR verknüpften Ausdrucks auch dann berechnet werden, wenn das Ergebnis des Gesamtausdrucks bereits feststeht.

Der Modus $B-

Im Modus {$B-} werden die Teile eines Ausdrucks in der Reihenfolge ihres Auftretens ausgewertet; die Auswertung bricht ab, sobald sich am Gesamtergebnis nichts mehr ändern kann (Short-Circuit; sog. Kurzschlußverfahren)

$D Informationen zur Fehlersuche

Dieser Schalter legt fest, ob der Compiler zusätzliche Informationen für den Debugger erzeugt oder nicht.

Syntax: `$D+` oder `$D-`
Voreinstellung: `$D+`
Typ: Global
Kommandozeile: `/$D+` oder `/$D-`
Menüäquivalent: Options|Compiler|**Debug-Information**

Anmerkungen:

Die Informationen für den Debugger bestehen aus einer Tabelle mit Zeilennummern für jede Routine, über die die Verbindung zwischen den Adressen der einzelnen Befehle und den Zeilen im Quelltext hergestellt werden kann.

Sind die Debug-Informationen angeschaltet {$D+}, kann Turbo Debugger für Windows den Quelltext schrittweise durchlaufen und Breakpoints in dem Programm oder der Unit setzen.

Die Option O|Linker|**Map File** erzeugt diese Informationen nur, wenn Sie das entsprechende Modul im Modus {$D+} compiliert haben.

Bei der Compilierung von Units werden diese Zusatzinformationen zusammen mit Code und Daten direkt in der TPU-datei gespeichert.

Selbstverständlich vergrößern diese Debug-Informationen die Größe von TPU-Dateien, sie haben aber keine Auswirkungen auf die Arbeitsgeschwindigkeit des Programmes.

Üblicherweise wird der Schalter für die Debug-Informationen zusammen mit dem Schalter {$L} **Local Symbols** verwendet.

Hinweis: Falls Sie Ihr Programm mit dem Turbo Debugger für Windows überprüfen wollen, stellen Sie Options|Linker|**Debug Info in Exe**ein.

$F Erzwingen von Far-Aufrufen

Bestimmt, welches Aufrufmodell für später compilierte Prozeduren und Funktionen verwendet wird.

Syntax: `$F+` oder `$F-`
Voreinstellung: `$F-`
Typ: Lokal
Kommandozeile:- `/$F+` oder `/$F-`
Menüäquivalent: Options|Compiler|**Force Far Calls**

Der Modus $F+

Prozeduren und Funktionen, die im $F+ Modus compiliert sind, verwenden immer das Aufrufmodell FAR.

Der Modus $F-

In diesem Fall arbeitet Turbo Pascal bei der Codierung von Aufrufen nach folgendem Schema:
- FAR, wenn die Prozedur oder Funktion im Interface-Teil einer Unit deklariert ist;
- NEAR, wenn sie an anderer Stelle deklariert ist.

Routinen, die in Prozedurvariablen vorkommen, müssen ebenfalls FAR codiert werden.

$G Generierung von 80286-Code

Dieser Schalter steuert, ob der Compiler Code für den 80286-Prozessor erzeugt oder nicht.

Syntax: `$G+` oder `$G-`
Voreinstellung: `$G-`
Typ: Lokal
Kommandozeile: `/$G+` or `/$G-`
Menüäquivalent: Options|Compiler|**286 Code**

Der Modus $G-

Im $G- Modus werden nur 8086-Anweisungen generiert.

Programme, die so compiliert werden, laufen auf allen 80x86 Prozessoren.

Der Modus $G+

Der Compiler verwendet im diesem Modus Assembler-Anweisungen, die erst ab einem 80286 Prozessor (oder dessen Nachfolgern) vorhanden sind.

Programme, die mit $G+ übersetzt wurden, können auf Rechnern mit 8088- oder 8086-CPU nicht mehr ausgeführt werden.

Weitere Anweisungen für den $G+ Modus:
- ENTER
- LEAVE
- PUSH immediate
- extended IMUL
- extended SHL
- extended SHR

$I Ein/Ausgabe-Prüfung

Dieser Schalter legt fest, ob Ein- und Ausgaben zur Laufzeit des Programms automatisch auf Fehler überprüft werden sollen.

Syntax: `$I+` oder `$I-`
Voreinstellung: `$I+`
Typ: Lokal
Kommandozeile: `/$I+` oder `/$I-`
Menüäquivalent: Options|Compiler|**I/O Checking**

Anmerkungen:

Tritt bei einem Ein-/Ausgabebefehl ein Fehler auf, dann bricht das Programm, wenn es im Modus {$I+} übersetzt wurde, mit einer Laufzeitfehlermeldung ab.

Im Modus {$I-} muß die Fehlerprüfung über Aufrufe der Funktion **IOResult** erfolgen.

$L Lokale Symbole

Dieser Schalter legt fest, ob Turbo Pascal weitere Informationen über lokale Symbol in ein Modul aufnimmt oder nicht.

Syntax: `$L+` oder `$L-`
Voreinstellung: `$L+`
Typ: Global
Kommandozeile: `/$L+` oder `/$L-`
Menüäquivalent: Options|Compiler|**Local Symbols**

Anmerkungen

Die Informationen über lokale Symbole bestehen aus
- den Namen aller lokalen Variablen und Konstanten des Moduls (also auch der im Implementationsteil) und
- den Symbolen innerhalb der Prozeduren und Funktionen des Moduls

Wenn ein Programm oder eine Unit mit diesen Informationen übersetzt wurde, dann können Sie im integrierten Debugger auch den Inhalt der lokalen Variablen des Moduls untersuchen und ändern.

Solange das Markierungsfeld Options|Linker|**Debug Info in Exe** nicht aktiviert oder beim Aufruf von TPCW.EXE **/V** nicht angegeben wurde, bleibt $L ohne Wirkung.

Die Optionen Options|Debug|**Debug Information** und Options|Linker|**Map File** erzeugen nur dann lokale Symbole für ein Modul, wenn es im Modus {$L+} übersetzt wurde.

Informationen über lokale Symbole von Units werden beim Compilieren der .TPU-Datei hinzugefügt. Die .TPU-Datei wird dadurch größer und braucht beim Compilieren von Programen, die die Unit verwenden, mehr Platz. Die Größe oder Geschwindigkeit der :EXE-Datei wird nicht betroffen.

Dieser Schalter wird üblicherweise zusammen mit {**$D**} verwendet.

Hinweis: Wenn keine Debug-Information erzeugt wird, ist der Schalter {$L} wirkungslos.

$N Numerische Operationen

Dieser Schalter legt fest, ob Operationen mit Realzahlen durch Routinen der Laufzeitbibliothek oder über die direkte Ansteuerung eines numerischen Coprozessor stattfinden.

Syntax: `$N+` oder `$N-`
Voreinstellung: `$N-`
Typ: Global
Kommandozeile: `/$N+` oder `/$N-`
Menüäquivalent: Options|Compiler|**80x87**

Der Modus $N-

Im Modus {$N-} steht nur der Datentyp Real zur Verfügung, alle Operationen mit diesem Typ geschehen über Aufrufe der Laufzeitbibliothek.

Der Modus $N+

Im $N+ Modus generiert Turbo Pascal Code für Gleitkommaberechnungen mit dem 8087 Koprozessor und bietet weitere vier Real-Typen: Single, Double, Extended und Comp.

$R Bereichsüberprüfung

Dieser Schalter legt fest, ob der Compiler zusätzlichen Code zur Bereichsüberprüfung generiert.

Syntax: `$R+` oder `$R-`
Voreinstellung: `$R-`
Typ: Lokal
Kommandozeile: `/$R+` oder `/$R-`
Menüäquivalent: Options|Compiler|**Range-checking**

Der Modus $R+

Im Modus {$R+}
- werden die Indizes von Arrays und Strings überprüft, ob sie in den definierten Grenzen liegen.
- werden alle Zuweisungen zu skalaren oder Unterbereichstypen überprüft, ob sie im zulässigen Wertebereich liegen.

Wird eine Bereichsüberschreitung festgestellt, bricht das Programm mit einer entsprechenden Fehlermeldung ab.

Diese zusätzliche Prüfung vergrößert den Programmumfang und setzt die Ausführungsgeschwindigkeit deutlich herab.

Sie sollten diese Option ausschließlich in der Testphase Ihrer Programme verwenden und danach ausschalten (`{$R-}`).

$S Stack-Prüfung

Dieser Schalter legt fest, ob der Compiler zusätzlichen Prüfcode vor Belegungen des Stacks erzeugt.

Syntax: `$S+` oder `$S-`
Voreinstellung: `$S+`
Typ: Lokal
Kommandozeile: `/$S+` oder `/$S-`
Menüäquivalent: Options|Compiler|**Stack Checking**

Der Modus $S+

Im Modus `{$S+}` wird vor jedem Prozedur- oder Funktionsaufruf geprüft, ob noch genügend Platz auf dem Stack zur Speicherung der Rücksprungadressen, der lokalen Variablen etc. vorhanden ist.

Ist dies nicht der Fall, bricht das Programm an dieser Stelle mit der entsprechenden Laufzeitfehlermeldung ab.

Der Modus $S-

Bei zu wenig freiem Stack verursacht ein Aufruf von Prozeduren und Funktionen, die mit $S+ compiliert sind, einen Laufzeitfehler. Bei zu wenig freiem Stack im Modus $S- erfolgt möglicherweise einen Systemabsturz.

$V Überprüfung von Var-Strings

Dieser Schalter legt fest, welche Prüfungen der Compiler bei der Übergabe von Strings als VAR-Parameter vornimmt.

Syntax: `$V+` oder `$V-`
Voreinstellung: `$V+`
Typ: Lokal
Kommandozeile: `/$V+` oder `/$V-`
Menüäquivalent: Options|Compiler|**String Var-Checking**

Der Modus $V+

Der Compiler setzt im Modus `{$V+}` voraus, daß formale und übergebene String-Parameter einen identischen Typ haben und weist jede Verletzung dieser Regel als Fehler aus.

Der Modus $V-

Im Modus `{$V-}` kann jeder String als VAR-Parameter übergeben werden, selbst dann, wenn die Länge des formalen Parameters und die des tatsächlich übergebenen String nicht übereinstimmen.

$W Windows Stack Frame

Generiert Vor- und Nachspanncode für far-Prozeduren und Funktionen.

Syntax: `$W+` oder `$W-`
Voreinstellung: `$W+`
Typ: Lokal
Kommandozeile: `/$W+` oder `/$W-`
Menüäquivalent: Options|Compiler|**Windows Stack Frame**

Der Modus $W+

Der Windows Stack Frame muß eingeschaltet sein, wenn Programme in real mode und/oder Standard oder enhanced 386 mode laufen sollen.

Der Modus $W-

Schalten Sie Windows Stack Frame aus, wenn ein Programm nur im protected mode läuft.

$X Erweiterte Syntax

Dieser Schalter legt fest, ob die erweiterte Syntax von Turbo Pascal benutzt wird oder nicht.

Syntax: `$X+` oder `$X-`
Voreinstellung: `$X+`
Typ: Global
Kommandozeile: `/$X+` oder `/$X-`
Menüäquivalent: Options|Compiler|**Extended Syntax**

Der Modus $X+

Im Modus `{$X+}` können Funktionen genauso wie Prozeduren aufgerufen werden, d.h. das Funktionsergebnis kann vernachlässig werden.

Üblicherweise macht dies keinen Sinn, werden doch Funktionen aufgrund des Wertes, den sie zurückliefern sollen, aufgerufen.

In manchen Fällen aber führen Funktionen mehrere Operationen aus, die von ihren Parametern abhängig sind. Möglicherweise kann es dabei zu unsinnigen Funktionsergebnissen kommen und dann macht es durchaus einen Sinn, Funktionen als Prozeduren behandeln zu können.

Hinweis: Der Schalter `{$X+}` kann nicht auf eingebaute Funktionen, wie z.B. die der Unit **SYSTEM** angewendet werden.

Im Modus `{$X+}` wird ausserdem ein neuer Zeichenketten-Typ, PChar, für nullterminierte Zeichenketten und ab Null indizierte Zeichenarrays unterstützt. Weitere Informationen gibt es dazu unter **Strings Unit**.

$I Include-Datei

Dieser Schalter weist den Compiler an, die angegebene Datei während des Compilierens einzulesen.

Syntax: `$I FileName`
Typ: Lokal
Kommandozeile: `/I Path`
Menüäquivalent: Options|Directories|**Include Directories**

Anmerkungen

Wenn die Angabe von FileName keine Extension enthält, wird automatisch .PAS angenommen.
Wurde kein Suchweg angegeben, sucht Turbo Pascal

- zuerst im aktuellen Verzeichnis
- und dann in den Verzeichnissen, die im Eingabefeld "Include Directories" angegeben wurden

(oder in den Verzeichnissen, die mit der Option **/I** beim Aufruf von TPWC angegeben wurden).

Die Include-Datei wird in direktem Anschluß an den Befehl {$I FileName} in den Quelltext eingebunden.
Turbo Pascal erlaubt die Verschachtelung von Include-Dateien bis zu 15 Ebenen.

Hinweis: Ein Include-Befehl darf nicht inmitten eines Anweisungsblock stehen. Alle Anweisungen, die durch BEGIN und END geklammert werden, müssen sich in derselben Quelltextdatei befinden.

$L Einbinden von Object-Dateien

Der Befehl weist den Linker an, die Datei DateiName in das Programm einzubinden.

Syntax: `$L FileName`
Typ: Lokal
Kommandozeile: `/O Path`
Menüäquivalent: Options|Directories|**Object Directories**

Anmerkungen

Die Datei muß im Intel-Object-Format vorliegen (muß also mit einem Assembler erzeugt sein) Wird für die Datei FileName kein Suffix angegeben, hängt Turbo Pascal die Endung .OBJ an.

Wird für die Datei FileName kein Suchweg angegeben, sucht Turbo Pascal
- zuerst im aktuellen Verzeichnis
- und dann in den Verzeichnissen, die in

Options|Directories|Object Directories angegeben wurden (oder in den Verzeichnissen, die mit der Option **/O** des TPWC angegeben wurden)

$M Speicherbelegung

Gibt an, wie Turbo Pascal den Speicher zuweist.

Syntax: `$M StackSize, HeapSize`
Voreinstellung: `$M 8192,8192`
Typ: Global
Kommandozeile: `/$M`
Menüäquivalent: Options|Compiler|**Memory Sizes**

Hinweis: Der Befehl {$M} hat bei der Übersetzung einer Unit keine Auswirkung.

Siehe auch:

StackSize
HeapSize

$C Codesegmentattribute

Steuert die Attribute eines Codesegments.

Syntax: `$C Attribut Attribut`
Typ: Global

Anmerkungen

Jedes Codesegment in einem Programm oder einer Bibliothek hat Attribute, die sein Verhalten beim Laden in den Speicher bestimmen.

Codesegmentattribute sind in drei Gruppen mit jeweils zwei sich wechselseitig ausschließenden Optionen unterteilt.

Option	Bedeutung
MoveAble	Windows kann Codesegment im Speicher verlagern
Fixed	Windows kann Codesegment im Speicher nicht verlagern
Preload	Codesegment wird beim Programmstart geladen
DemandLoad	Codesegment wird beim Bedarf geladen
Permanent	Wenn Windows das Codesegment geladen hat, bleibt es im Speicher
Discardable	Codesegment kann bei Bedarf entfernt werden

Die erste Option jeder Gruppe ist die Vorgabe. Werden beide genannt, gilt die zweite.

$D Description

Fügt angegebenen Text in eine EXE-Datei oder DLL ein.

Syntax: `$D Text`
Typ: Global

$R Ressourcendatei

Benennt die Ressourcendatei, die in ein Programm oder in eine Bibliothek aufgenommen werden soll.

Syntax: `$R FileName`
Typ: Global
Kommandozeile: `/R path`
Menüäquivalent: Options|Directories|**Resource Directories**

Anmerkungen

Vorgabeextension für FileName ist .RES. Es muß sich hierbei um eine Windows-Ressourcendatei handeln.

Wird bei FileName kein Verzeichnis angegeben, sucht Turbo Pascal nach der Datei
- zuerst im aktuellen Verzeichnis
- dann in den Verzeichnissen aus dem Eingabefeld Resource Directories (oder in den unter der Kommandozeilen-Option /R genannten Verzeichnissen)

Beim Verwenden in einer Unit wird die Ressourcen-Datei in die .TPU-Datei aufgenommen. Es wird dabei nicht geprüft, ob die Datei existiert.

Beim Linken eines Programms oder einer Bibliothek werden die Ressourcendateien aller Units bearbeitet. Es wird jede Ressource jeder Ressourcendatei in die .EXE oder .DLL-Datei kopiert.

Der integrierte Assembler

Der integrierte Assembler von Turbo Pascal erlaubt Ihnen, 8086/8087- bzw. 80286/80287-Assembler-Anweisungen direkt in den Pascal-Quelltext einzufügen.

Index der Assembler-Hilfe

asm
Assembler-Direktive
Eintritts- und Austrittscode
Ausdrucksklassen
Ausdrucksoperatoren
Ausdruckssymbole
Ausdruckstypen
Ausdrücke
Funktionen
Anweisungs-Opcodes
Labels
numerische Konstanten
Operanden
vordefinierte Typen
Präfix-Opcodes
Prozeduren
Registersymbole
relozierbare Ausdrücke
reservierte Wörter
spezielle Symbole
Stringkonstanten

Der Einsatz des integrierten Assembler

Assemblerblöcke werden mit einer **asm-Anweisung** eingeleitet und mit end abgeschlossen.

Für Prozeduren und Funktionen des integrierten Assembler gelten dieselben Regeln wie für **external-**Prozeduren und -Funktionen**.**

Ausdrücke

Die Operanden des integrierten Assembler sind **Ausdrücke.** Die Grundelemente eines Ausdrucks sind Konstanten, Register, Symbole und Operatoren.

Der integrierte Assembler kennt drei unterschiedliche Klassen von Ausdrücken:

Registerausdrücke
Speicherreferenzen
Konstante Ausdrücke

Symbole

Der integrierte Assembler ermöglicht den Zugriff auf fast alle Pascal-Symbole. Diese Symbole können Labels, Konstanten, Typenbezeichner, Variablen, Prozeduren und Funktionen sein.

Zusätzlich stellt der integrierte Assembler einige **vordefinierte Typsymbole** zur Verfügung.

Konstanten

Der integrierte Assembler unterstützt zwei verschiedene Arten von Konstanten:

numerische Konstanten
Stringkonstanten

Opcodes, Operatoren und Befehle

Der integrierte Assembler unterstützt:

- alle 8086/8087- und 80286/80287-Befehle.
- Opcodes
- Die meisten **Ausdrucks-Operatoren** von TASM
- Anweisungen für Datenstrukturen **:** define byte, define word und define double word (**DB, DW, und DD**)

Meist haben Operationen, die bei Turbo Assembler über Anweisungen implementiert sind, eine Entsprechung in Turbo-Pascal-Konstrukten.

Die Syntax des integrierten Assemblers entspricht weitgehend der Syntax von Turbo Assembler bzw. Microsofts Makro-Assembler.

Eintritts- und Austrittscode des integrierten Assembler

Der automatisch generierte Eintritts- und Austrittscode einer Assembler-Prozedur bzw. -Funktion sieht folgendermaßen aus:

```
PUSH   BP          ;wenn Locals <> 0 oder Params <> 0
MOV    BP,SP       ;wenn Locals <> 0 oder Params <> 0

SUB    SP,Locals   ;wenn Locals <> 0
MOV    SP,BP       ;wenn Locals <> 0
POP    BP          ;wenn Locals <> 0 oder Params <> 0
RET    Params      ;immer
```

- Locals gibt die Größe der lokalen Variablen an
- Params gibt die Größe der Parameter an

Hat sowohl Locals als auch Params den Wert Null, erzeugt der integrierte Assembler keinen Eintrittscode, und der Austrittscode besteht lediglich aus dem RET-Befehl.

Operanden des integrierten Assembler

Operanden sind **Ausdrücke**, bestehend aus einer Kombination von Konstanten, Registern, Symbolen und Operatoren.

Obwohl die Syntax des integrierten Assemblers in vielen Fällen der Pascal-Syntax angeglichen wurde, gibt es doch eine Reihe von Unterschieden.

Der integrierte Assembler

- hat einen eigenen Satz von **reservierten Wörtern**–
- interpretiert Variablenreferenzen als Adresse, nicht wie Pascal, als Inhalt der Variablen

Ausdrücke im integrierten Assembler müssen stets einen konstanten Wert ergeben.

Ausdrücke des integrierten Assembler

Die Ausdrücke des integrierten Assemblers setzen sich aus Ausdruckselementen (Konstanten, Registern und Symbolen) und **Operatoren** zusammen. Zu jedem Ausdruck gehört eine **Ausdrucksklasse** und ein **Ausdruckstyp**.

Berechnung

Ausdrücke werden im integrierten Assembler immer als 32-Bit-Werte berechnet. Fließkommazahlen und Strings, ausgenommen **Stringkonstanten**, werden nicht akzeptiert.

Der wichtigste Unterschied zwischen Ausdrücken in Turbo Pascal und Ausdrücken im integrierten Assembler besteht darin, daß Ausdrücke im integrierten Assembler stets einen konstanten Wert haben müssen. Mit anderen Worten: der Wert eines Ausdrucks muß zum Zeitpunkt der Compilierung feststehen.

Ausdrucksklassen

Der Integrierte Assembler kennt drei unterschiedliche Klassen von Ausdrücken:

Registerausdrücke
Speicherreferenzen
Konstante Ausdrücke

Ausdrucksoperatoren des integrierten Assembler

Die nachfolgende Tabelle listet die Operatoren des integrierten Assemblers gemäß ihrer Priorität in absteigender Rangfolge auf.

Die Rangfolge der Operatoren jeder Kategorie ist gleich.

Kategorie	Operator	Zweck oder Funktion
Höchste	**&**	Bezeichnervorgabe
	(...)	Teilausdrücke
	[...]	Speicherreferenzen
	.	Strukturauswahl
Unäre	**HIGH**	Liefert 8 höherwertige Bits
	LOW	Liefert 8 niederwertige Bits
	+	Unäres Plus
	-	Unäres Minus
	:	Segmentvorgabe
	OFFSET	Liefert Offset-Anteil
	SEG	Liefert Segmentanteil
	TYPE	Liefert den Typ (Größe in Byte)
	PTR	Typumwandlung
	*	Multiplikation
	/	Ganzzahlige Division
	MOD	Integer-Modulo (Rest)
	SHL	Logisches Linksschieben
	SHR	Logisches Rechtsschieben
Addition	**+**	Binäre Addition
	-	Binäre Subtraktion
Bitweise	**NOT**	Bitweise Negation
	AND	Bitweises AND
	OR	Bitweises OR
	XOR	Bitweises XOR

Ausdruckstypen

Jeder Ausdruck des integrierten Assemblers hat einen bestimmten Typ oder, genauer gesagt, eine bestimmte Größe, da der Assembler den Typ eines Ausdrucks als eine Größenangabe für den Speicherzugriff interpretiert.

Der integerierte Assembler führt Typenprüfungen durch, wo immer dies möglich ist. Scheitert die Typenprüfung, tritt ein Fehler auf.

Man kann den Typ eines Speicherausdrucks durch entsprechende Operatoren ändern. Die folgenden drei Zeilen verweisen alle auf das erste Byte der Variablen OutBufPtr.

```
asm
  MOV     DL,BYTE PTR OutBufPtr
  MOV     DL,Byte(OutBufPtr)
  MOV     DL,OutBufPtr.Byte
end;
```

In manchen Fällen ist ein Speicherausdruck untypisiert. Ein Beispiel dafür ist ein konstanter Ausdruck in eckigen Klammern:

```
asm
  MOV     AL,[100H]
  MOV     BX,[100H]
end;
```

Der integrierte Assembler erlaubt beide Anweisungen, da der Ausdruck [100H] nicht typisiert ist. Der Ausdruck [100H] stellt lediglich einen Bezug auf den Inhalt der Adresse 100H im Daten-Segment dar, der Ausdrucks-Typ ist durch das verwendete Register (AL für das Byte und BX für das Word) bestimmt.

Situationen, in denen der Assembler den Typ nicht über einen zweiten Operator feststellen kann, erfordern eine explizite Typangabe:

```
asm
  INC     BYTE PTR [100H]
  IMUL    WORD PTR [100H]
end;
```

Vordefinierte Typsymbole

Die Tabelle faßt die vordefinierten Typsymbole zusammen, die der integrierte Assembler zusätzlich zu den in Pascal deklarierten zur Verfügung stellt.

Symbol	Typ
BYTE	1
WORD	2
DWORD	4
QWORD	8
TBYTE	10
NEAR	0FFFEH
FAR	0FFFFH

NEAR und FAR

Beachten Sie insbesondere die Pseudo-Typen NEAR und FAR, die bei Prozedur- und Funktionssymbolen zur Angabe des Aufrufmodells dienen. Ebenso wie andere Symbole, können Sie in Typangaben NEAR und FAR verwenden. Angenommen, FarProc ist eine FAR-Prozedur:

```
procedure FarProc; far;
```

Wenn Sie nun im selben Modul Assembler-Code schreiben, ist es möglich, die weit effizientere NEAR-Anweisung für den FarProc Prozeduraufruf einzusetzen:

```
asm
  PUSH    CS
  CALL    NEAR PTR FarProc
end;
```

Registerausdrücke

Ein Ausdruck, der nur aus einem Registernamen besteht, ist ein Registerausdruck. Beispiele für Registerausdrücke sind AX, CL, DI und ES.

Registerausdrücke, die als Operanden verwendet werden, weisen den Assembler an, Befehle zu generieren, deren Operationen sich auf die CPU-Register beziehen.

Speicherreferenzen

Ausdrücke, die auf Speicherstellen verweisen, sind Speicherreferenzen; zu dieser Kategorie gehören Pascal-Labels, Variablen, typisierte Konstanten, Prozeduren und Funktionen.

Speicherausdrücke und konstante Ausdrücke werden zusätzlich als relozierbare oder absolute Ausdrücke klassifiziert.

Konstante Ausdrücke

Ausdrücke, die keine Register-oder Speicherausdrücke sind, sind konstante Ausdrücke; in diese Gruppe gehören untypisierte Konstanten und Typbezeichner.

Speicherausdrücke und konstante Ausdrücke unterteilen sich zusätzlich in **relozierbare** und **absolute** Ausdrücke.

Relozierbare und absolute Ausdrücke

Ausdrücke, die sich auf Prozeduren, Funktionen, Variablen und Labels beziehen, sind relozierbare Ausdrücke. Beinhaltet ein Ausdruck dagegen ausschließlich einen konstanten Wert, handelt es sich um einen absoluten Ausdruck.

Ein relozierbarer Ausdruck bezeichnet einen Wert, der zum Zeitpunkt des Linkens reloziert werden muß. (Relokation ist das Zuweisen absoluter Adressen an Symbole.)

Ein absoluter Ausdruck ist ein Wert, der keiner Relokation bedarf.

Zum Zeitpunkt der Compilierung kennt der Compiler keine endgültige Adresse von Label, Variable, Prozedur, oder Funktion.

Die endgültige Adresse ist erst bekannt, wenn der Linker Symbolen eine spezifische absolute Adresse zuweist.

Der integrierte Assembler erlaubt beliebige Operationen mit absoluten Ausdrücken, begrenzt aber die Operationen mit relozierbaren Ausdrücken auf die Addition bzw. Subtraktion konstanter Werte.

Registersymbole

Im integrierten Assembler von Turbo Pascal bezeichnen die folgenden reservierten Symbole die CPU-Register:

Symbole	Register
AX BX CX DX	allgemeine 16-Bit-Register
AL BL CL DL	niederwertige 8-Bit-Register
AH BH CH DH	höherwertige 8-Bit-Register
SP BP SI DI	16-Bit Zeiger-Register
CS DS SS ES	16-Bit-Segment-Register
ST	8087 Register-Stack

Wenn ein Operand aus nur einem Registerbezeichner besteht, nennt man diesen Operanden einen Register-Operanden. Alle Register können als Register-Operanden dienen.

Indizieren über Register

Die Basis-Register (BP und BX), ebenso wie die Index-Register (SI und DI) können - in eckigen Klammern geschrieben - für indizierte Speicherzugriffe verwendet werden. Folgende Kombinationen von Basis- und Index-Registern sind erlaubt:

```
[BP]
[BP+DI]
[BP+SI]
[BX]
[BX+DI]
[BX+SI]
[DI]
[SI]
```

Überschreiben der Segmentvorgabe

Die Segment-Register (ES, CS, SS und DS), gefolgt von einem Doppelpunkt, können zum **Überschreiben** der Segmentvorgabe benutzt werden, um ein anderes Segment zu indizieren, als der Prozessor als Vorgabe wählt.

Symbole des integrierten Assembler

Der integrierte Assembler ermöglicht den Zugriff auf fast alle Pascal-Symbole. Diese Symbole können Labels, Konstanten, Typ-Bezeichner, Variablen, Prozeduren und Funktionen sein.

Zusätzlich stellt der integrierte Assembler noch folgende **vordefinierten Typsymbole** zur Verfügung:

Symbol	Wert	Klasse	Typ
Label	Addresse des Label	Speicher	Short
Konstante	Wert der Konstanten	Konstante	0
Typ	0	Speicher	Größe des Typs
Feld	Offset des Feldes	Speicher	Größe des Typs
Variable	Variablenadresse	Speicher	Größe des Typs
Prozedur	Prozeduradresse	Speicher	**Near** oder **Far**
Funktion	Funktionsadresse	Speicher	**Near** oder **Far**
Unit	0	Konstante	0
@Code	Codesegmentadresse	Speicher	0FFF0H
@Data	Datensegmentadresse	Speicher	0FFF0H
@Result	Ergebnis (var offset)	Speicher	Größe des Typs

Folgende Symbole können nicht in Assembler-Ausdrücken verwendet werden:

- Standard-Prozeduren und -Funktionen (z.B. WriteLn, Chr)
- Die vordefinierten Pseudo-Arrays Port und PortW
- String-, Fließkomma- und Mengen-Konstanten
- Prozeduren und Funktionen, die mit der **inline-Anweisung** deklariert sind
- **Labels**, die nicht im aktuellen Block definiert sind
- Das Symbol @Result außerhalb von Funktionen

Lokale Variablen

Lokale Variablen (Variablen, die in Funktionen und Prozeduren deklariert sind), werden immer auf dem Stack angelegt und relativ zu SS:BP adressiert.

Der Wert des Symbols einer lokalen Variablen gibt den vorzeichenbehafteten Offset relativ zu SS:BP an. Bei Referenzen auf lokale Variablen fügt der Assembler automatisch den Bezug auf das Register BP hinzu.

Var Parameter

Der integrierte Assembler behandelt **var**-Parameter immer als 32-Bit-Zeiger mit einer Größe von 4 Bytes

Wenn Sie also auf den Inhalt eines **var**-Parameters zugreifen möchten, müssen Sie zunächst den 32-Bit-Zeiger laden und dann auf die Speicherstelle zugreifen, auf die der Zeiger verweist.

Speicherbereich

Der Speicherbereich eines Record- bzw. Objekt-Typs wird von dessen Typ-, Feld- und Variablen-Symbolen beschrieben.

Zusätzlich gibt ein Unit-Bezeichner, in ähnlicher Weise wie ein qualifizierter Bezeichner

in Pascal, eine bestimmte Unit als Gültigkeitsbereich an.

Operator

Manche Symbole, wie beispielsweise Record-Typen und Variablen, markieren einen Speicherbereich, dessen Felder mit einem Punkt und dem **Feldbezeichner** angesprochen werden können.

Typ-Bezeichner

Typ-Bezeichner können bei Speicheroperationen zur Erzeugung des richtigen Offsets eingesetzt werden. Für die folgenden fünf Anweisungen wird ein und derselbe Maschinen-Code erzeugt, der den Inhalt der Speicherstelle ES:[DI+4] in das AX-Register lädt:

```
asm
  MOV     AX,(Rect PTR ES:[DI]).B.X
  MOV     AX,Rect(ES:[DI]).B.X
  MOV     AX,ES:Rect[DI].B.X
  MOV     AX,Rect[ES:DI].B.X
  MOV     AX,ES:[DI].Rect.B.X
end;
```

Das Registersymbol ST(x)

Das Symbol ST bezeichnet die Spitze des Koprozessor-Register-Stack. Jedes der acht Koprozessor-Register kann durch das Symbol ST(x) angesprochen werden, wobei X die Registernummer darstellt. Die Register sind von 0 bis 7 numeriert, die Nummer gibt die Distanz zur Spitze des Register-Stack an

Stringkonstanten

Stringkonstanten müssen entweder in Apostrophe ('...') oder Anführungszeichen ("...") gesetzt werden.

Zwei Anführungszeichen bzw. Apostrophe innerhalb einer Stringkonstanten gelten als ein Zeichen.

Stringkonstanten, die über DB-Anweisungen deklariert sind, haben keine Längenbeschränkung und ergeben eine Byte-Folge, welche den ASCII-Werten der einzelnen Zeichen entspricht.

Eine Stringkonstante kann, wenn sie nicht in einer DB-Anweisung steht, nicht länder als vier Zeichen sein, und enthält einen numerischen Wert, der in einem Ausdruck stehen kann.

So wird der numerische Wert einer Stringkonstanten berechnet:

```
Ord(Ch1)
+ Ord(Ch2) shl  8
  + Ord(Ch3) shl 16
    + Ord(Ch4) shl 24
```

- Ch1 entspricht dem äußersten rechten (letzten) Zeichen,
- Ch4 dem äußersten linken (ersten) Zeichen der Zeichenkette.

Ist die Zeichenkette kürzer als vier Zeichen, werden entsprechend der Länge die ersten Bytes auf Null (0) gesetzt.

Stringkonstante	**Numerischer Wert**
'a'	00000061H
'ba'	00006261H
'cba'	00636261H
'dcba'	64636261H
'a '	00006120H
' a'	20202061H
'a'*2	000000E2H
'a'-'A'	00000020H
NOT 'a'	FFFFFF9EH

Numerische Konstanten

Numerische Konstanten sind ganze Zahlen im Wertebereich von 2.147.483.649 bis 4.294.967.295, sie müssen mit einer Zahl von 0 bis 9 oder dem Zeichen $ beginnen.

Gemäß Standardeinstellung werden numerische Konstanten in dezimaler Schreibweise geschrieben (Basis 10). Der integrierte Assembler unterstützt allerdings auch die binäre (Basis 2), die oktale (Basis 8) und die hexadezimale (Basis 16) Schreibweise.

Notation	**Schreiben Sie...**
Binär	Buchstabe B nach der Zahl
Oktal	Buchstabe O nach der Zahl
Hexadezimal	Buchstabe H nach der Zahl oder $ vor der Zahl

Für Pascal-Ausdrücke stehen ausschließlich das dezimale und das hexadezimale Zahlensystem zur Verfügung;
Hexadezimal-Zahlen werden immer mit dem Präfix $ eingeleitet.

Numerische Konstanten müssen mit einer Ziffer von 0 bis 9 beginnen bzw. mit dem Präfix $ bei hexadezimalen Zahlen. Beim Schreiben einer Hexadezimalzahl mit H als Suffix, muß auch diese Zahl mit einer Ziffer von 0 bis 9 beginnen (z.B. 0F5Fh).

Beispiele:

0BAD4H	Hexadezimal-Konstante
$BAD4	Hexadezimal-Konstante
BAD4H	Bezeichner (beginnt mit Buchstabe B, nicht mit Zahl)

Reservierte Wörter

Innerhalb von **Operanden** haben folgende reservierten Wörter eine besondere Bedeutung für den integrierten Assembler:

Tabelle reservierter Wörter

AH
AL
AND
AX
BH
BL
BP
BX
BYTE
CH
CL
CS
CX
DH
DI
DL
DS
DWORD
DX
ES
FAR
HIGH
LOW
MOD
NEAR
NOT
OFFSET
OR
PTR
QWORD
SEG
SHL
SHR
SI
SP
SS
ST
TBYTE
TYPE
WORD
XOR

Diese reservierten Wörter haben immer Vorrang vor selbstdefinierten Bezeichnern.

Um auf einen selbstdefinierten Bezeichner mit demselben Namen wie ein reserviertes Wort zugreifen zu können, muß der Operator zum **Überschreiben von Bezeichnern** (&) verwendet werden.

Achtung:vermeiden Sie möglichst Bezeichnernamen, die mit reservierten Wörtern identisch sind.

Präfix-Opcodes

Der integrierte Assembler unterstützt folgende Präfixe:

Opcode	Bedeutung
LOCK	Bus lock
REP	wiederhole String-Operation
REPE	wiederhole String-Operation, solange Equal-Flag gesetzt
REPZ	wiederhole String-Operation, solange Zero-Flag gesetzt
REPNE	wiederhole String-Operation, solange Equal-Flag nicht gesetzt
REPNZ	wiederhole String-Operation, solange Zero-Flag nicht gesetzt
SEGCS	CS-Register-Vorgabe
SEGDS	DS-Register-Vorgabe
SEGES	ES-Register-Vorgabe
SEGSS	SS-Register-Vorgabe

Ein Assembler-Befehl wird, soweit überhaupt, nur selten mit mehr als einem einzigen Präfix eingeleitet.

Wenn Sie einen Präfix-Opcode ohne einen **Befehls-Opcode** in derselben Anweisung schreiben, bezieht sich das Präfix auf den Assembler-Befehl der nächsten Anweisung.

Manche 80x86-Prozessoren bearbeiten nicht alle Präfix-Kombinationen korrekt. Vorsicht bei der Reihenfolge mehrerer Präfixe!

Befehlssätze

Der integrierte Assembler unterstützt alle 8086/8087- und 80286/80287-Befehle.

Die Befehlssätze sind nur in folgenden Modi aktiv:

Opcodes	Zustand	Bedeutung des Zustandes
8087	$N+	Numerischer Prozessor aktiv
80286	$G+	80286 Codegenerierung aktiv
80287	$G+,N+	Beide aktiv

Eine vollständige Beschreibung der Befehlssätze für den 80x86 und 80x87 finden Sie in den Referenz-Handbüchern der 80x86- und 80x87-Prozessoren.

Siehe auch:

RET-Befehl des integrierten Assembler
Automatische Sprunganpassung

RET-Befehl

Der RET-Befehl erzeugt, in Abhängigkeit vom Aufrufmodell der aktuellen Funktion oder Prozedur, für den Rücksprung automatisch einen Near- oder Far-Maschinencode.

Erzeugt NEAR Rücksprung

```
procedure NearProc; near;
begin
  asm
    RET
  end;
end;
```

Erzeugt FAR Rücksprung

```
procedure FarProc; far;
begin
  asm
    RET
  end;
end;
```

- RETN: immer ein near-Rücksprung
- RETF: immer ein far-Rücksprung

Automatische Sprunganpassung

Solange nicht anders spezifiziert, optimiert der integrierte Assembler Sprungbefehle durch Einsetzen der kürzesten und damit effizientesten Befehlsvariante.

Diese Optimierung erfolgt bei allen unbedingten Sprüngen (JMP), aber auch bei allen bedingten Sprüngen, wenn als Sprungziel ein Label, nicht jedoch eine Funktion oder Prozedur angegeben ist.

Opcode	**Distanz zum Ziel-Label**	**Assembler Generiert**
JMP	-128 bis 127 Bytes	**Short Jump**
	NICHT -127 bis 128 Bytes	**Near Jump**
Bedingte Sprünge	-128 bis127 Bytes	**Short Jump**
	NICHT -127 bis 128 Bytes	**Short inverse-Jump**

Sprünge auf Funktionen und Prozeduren werden immer entweder als Near oder Far erzeugt, niemals aber als short.

Außerdem erlaubt der integrierte Assembler keine bedingten Sprünge auf Funktionen oder Prozeduren.

DB, DW und DD Anweisungen

Der integrierte Assembler von Turbo Pascal unterstützt drei verschiedene Anweisungen zur Erzeugung von Datenstrukturen: DB (define byte), DW (define word) und DD (define double word).

	Operand Typ	**Wertebereich**	**Assembler Generiert**
DB	Konstanter Ausdruck	-128 bis 255	1 Byte
	Zeichen-String	beliebige Länge	Bytefolge, die dem ASCII-Code jedes Zeichens entspricht
DW	Konstanter Ausdruck	-32,768 bis 65,535	1 Word
	Address-Ausdruck		Near Zeiger (offset Word)
DD	Konstanter Ausdruck	-2,147,483,648 bis 4,294,967,295	1 double Word
	Addreß-Ausdruck		Far Zeiger (offset Word gefolgt von Segment Word)

Die in den Anweisungen DB, DW und DD generierten Daten werden immer im Code-Segment abgelegt.

Im Daten-Segment werden auch weiterhin nur diejenigen initialisierten oder nicht-initialisierten Daten abgelegt, die über eine entsprechende Pascal-Deklaration (**var** oder **const**) deklariert wurden.

Es folgen einige Beispielanweisungen:

	Operand	**Ergebnis**
DB	0FFH	Ein Byte
DB	0,99	Zwei Byte
DB	'A'	Ord('A')
DB	'Hello...', 0DH,0AH	String + CR/LF
DB	12,"Turbo Pascal"	Pascal-String
DW	0FFFFH	Ein Word
DW	0,9999	Zwei Word
DW	'A'	Wie DB 'A',0
DW	'BA'	Wie DB 'A','B'
DW	MyVar	Offset von MyVar
DW	MyProc	Offset von MyProc
DD	0FFFFFFFFH	Ein double Word
DD	0,999999999	Zwei double Word
DD	'A'	Wie DB 'A',0,0,0
DD	'DCBA'	Wie DB 'A','B','C','D'
DD	MyVar	Zeiger auf MyVar
DD	MyProc	Zeiger auf MyProc

Die einzigen Symbole, die man innerhalb einer asm-Anweisung definieren kann, sind **Labels**. Variablen müssen gemäß der Pascal-Syntax deklariert werden.

Die Syntax einer Assembler-Anweisung

Asm-Anweisungen haben folgende Syntax:

```
asm asmBefehl < Trennzeichen asmBefehl > end
```

- **Trennzeichen** ist ein Semikolon, ein Zeilenbeginn oder ein Pascal-Kommentar.
- **asmBefehl** ist eine Assembler-Anweisung mit der folgenden Syntax:

```
[Label ":"] <Präfix> [Opcode [Operand <"," Operand>]]
```

- **Label** stellt einen Labelbezeichner dar
- **Präfix** ist ein Assembler-Präfixopcode (Operationscode)
- **Opcode** ist ein Assembler-Befehl oder eine Anweisung
- **Operand** ist ein gültiger Assembler-Ausdruck.

Es ist zulässig, mehrere Assembler-Anweisungen in eine einzige Zeile zu schreiben, sofern die Anweisungen durch ein Semikolon voneinander getrennt werden. Kommentare müssen den Pascal-Konventionen entsprechend gekennzeichnet werden, d.h. mit **{** und **}** oder **(*** und ***)**.

Labels

Labels bestehen aus einem Labelbezeichner gefolgt von einem Doppelpunkt.

Ein Label, das in einer Assembler-Anweisung definiert ist, muß im label-Deklarationsteil des Programmabschnittes mit der **asm Anweisung** deklariert werden. Von dieser Regel gibt es nur eine Ausnahme:

Lokale Labels müssen nicht explizit definiert werden.

Die Symbole @Code, @Data und @Result

Der integrierte Assembler ermöglicht den Zugriff auf fast alle Pascal-Symbole. Zusätzlich stellt der integrierte Assembler noch folgende spezielle Symbole zu Verfügung:

Symbol	Bedeutung
@Code	Das aktuelle Codesegment
@Data	Das aktuelle Datensegment
@Result	Ergebnisvariable im Anweisungsteil einer Funktion

Die Symbole @Code und @Data sollten nur zusammen mit dem **SEG-Operator** verwendet werden.

Operator zum Überschreiben reservierter Wörter (&)

Ein Bezeichner, der unmittelbar nach einem Et-Zeichen (&) folgt, wird selbst dann als benutzerdefiniertes Symbol betrachtet, wenn seine Schreibweise mit der eines **reservierten Wortes** des integrierten Assemblers identisch ist. Beispiel:

```
&Bezeichner
```

κTeilausdrücke (...)

Ausdrücke, die innerhalb runder Klammern stehen,sind Teilausdrücke.

```
(Ausdruck)
```

Teilausdrücke werden komplett ausgewertet und als einziges Ausdruckselement behandelt.

Ein weiterer Ausdruck kann diesem Teilausdruck vorangestellt werden. Das Ergebnis ergibt sich aus der Summe beider Ausdrücke, wobei der Ergebnistyp dem Typ des vorangestellten Ausdruckes entspricht.

Speicherreferenz-Operator [...]

Ein Ausdruck, der innerhalb eckiger Klammern steht, verweist auf eine Speicherstelle.

```
[Ausdruck]
```

Diese Speicherreferenz wird komplett ausgewertet und als einzelnes Ausdruckselement behandelt.

Der Klammerausdruck kann für indizierte Speicherzugriffe eine Kombination der Register BX, BP, SI oder DI und des Plus (+)- bzw. Minus (-)-Operators enthalten.

Ist dem Klammerausdruck ein weiterer Ausdruck vorangestellt, ergibt sich das Ergebnis aus der Summe beider Ausdrücke.

Ergebnis

Das Ergebnis ist immer eine Speicherreferenz.

Strukturauswahl-Operator (xxx.yyy)

Der zweite Ausdruck ist ein Element der durch den ersten Ausdruck bezeichneten Struktur.

```
Ausdruck1.Ausdruck2
```

Ergebnis

Die Summe der beiden Ausdrücke.

Ergebnistyp

Typ des zweiten Ausdrucks.

Auf Symbole, die sich auf denselben Gültigkeitsbereich beziehen wie der erste Ausdruck, kann über den zweiten Ausdruck zugegriffen werden.

HIGH Operator

HIGH liefert die acht höherwertigen Bits des konstanten Ausdruckes, der dem Operator folgt, zurück.

```
HIGH Ausdruck
```

Der Ausdruck muß eine absolute Konstante sein.

LOW-Operator

LOW Liefert die acht niederwertigen Bits des konstanten Ausdruckes, der dem Operator folgt, zurück.

```
LOW Ausdruck
```

Der Ausdruck muß eine absolute Konstante sein.

Unäres Plus (+...)

Liefert den folgenden Ausdruck unverändert zurück.

```
+Ausdruck
```

Der Ausdruck muß eine absolute Konstante sein.

Unäres Minus (-...)

Liefert den negierten Wert eines Ausdruckes zurück.

```
-Ausdruck
```

Der Ausdruck muß ein absoluter, konstanter Wert sein.

Segmentvorgabe-Operator (: ...)

Weist den integrierten Assembler an, die Adresse des folgenden Ausdruckes relativ zu einem bestimmten Segment zu berechnen. Das Segment wird durch den Registernamen (CS, DS, SS oder ES) vor dem Doppelpunkt bestimmt.

```
XX:Ausdruck
```

mit XX = CS, DS, SS oder ES.

Ergebnis

Das Ergebnis ist eine Speicherreferenz; es wird aus dem Ausdruck nach dem Doppelpunkt errechnet.

Bei einer Segmentvorgabe erzeugt der integrierte Assembler ein Präfix, um die CPU anzuweisen, den folgenden Ausdruck auf ein anderes als das voreingestellte Segmentregister zu beziehen.

OFFSET Operator

Liefert den Offset-Anteil des folgenden Ausdruckes zurück.

```
OFFSET Ausdruck
```

Ergebnis

Das Ergebnis ist eine Konstante.

SEG Operator

Liefert den Segment-Anteil des folgenden Ausdruckes zurück.

```
SEG Ausdruck
```

Ergebnis

Das Ergebnis ist eine Konstante.

TYPE Operator

Liefert den Typ (Größe in Bytes) des folgenden Ausdruckes zurück.

```
TYPE Ausdruck
```

Bei einer Konstanten hat das Ergebnis den Wert 0.

Typecast-Operator (...PTR...)

PTR wandelt den zweiten Ausdruck in den Typ des ersten Ausdrucks um.

```
Ausdruck1 PTR Ausdruck2
```

Ergebnis

Eine Speicherreferenz mit dem Wert des zweiten Ausdrucks und dem Typ des ersten.

Multiplikation (...*...)

Der Operator * multipliziert den ersten Ausdruck mit dem zweiten Ausdruck.

```
Ausdruck1 * Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Ganzzahlige Division (.../...)

Der Operator / dividiert den ersten Ausdruck durch den zweiten Ausdruck und gibt den Integer-Anteil zurück.

```
Ausdruck1 / Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Modulo-Operator (...MOD...)

MOD dividiert den ersten Ausdruck durch den zweiten Ausdruck und gibt den Rest zurück.

```
Ausdruck1 MOD Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Logisches Linksschieben (...SHL...)

SHL schiebt den ersten Ausdruck um nnn Bits nach links, nnn ist der zweite Ausdruck.

```
Ausdruck1 SHL Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Logisches Rechtsschieben (...SHR...)

SHL Schiebt den ersten Ausdruck um nnn Bit nach rechts. nnn ist der zweite Ausdruck.

```
Ausdruck1 SHR Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Addition (...+...)

Der Operator + addiert den ersten zum zweiten Ausdruck.

```
Ausdruck1 + Ausdruck2
```

Die Ausdrücke können konstante Werte oder Speicher-Referenzen sein. Allerdings darf nur einer der beiden Ausdrücke ein relozierbarer Wert sein.

Ergebnis

Relozierbarer Wert, wenn ein Ausdruck relozierbar ist.
Sind beide Ausdrücke Speicher-Referenzen, dann ist auch das Ergebnis eine Speicherreferenz.

Subtraktion (...-...)

Der Operator + subtrahiert den zweiten vom ersten Ausdruck.

```
Ausdruck1 - Ausdruck2
```

Die Klasse des ersten Ausdruckes kann beliebig sein, der zweite Ausdruck muß ein absoluter, konstanter Wert sein.

Ergebnis

Die Klasse des Ergebnisses entspricht der Klasse des ersten Ausdruckes.

Bitweise Negation (NOT)

NOT liefert die bitweise Negation (Einerkomplement) des Ausdrucks.

```
NOT Ausdruck
```

Der Ausdruck muß ein absoluter, konstanter Ausdruck sein.

Ergebnis

Ein absoluter, konstanter Ausdruck.

Bitweise AND-Verknüpfung

AND liefert die bitweise AND-Verknüpfung zweier Ausdrücke.

```
Ausdruck1 AND Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Bitweise OR-Verknüpfung

OR liefert die bitweise OR-Verknüpfung zweier Ausdrücke.

```
Ausdruck1 OR Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Bitweise XOR-Verknüpfung

XOR liefert die bitweise XOR-Verknüpfung zweier Ausdrücke.

```
Ausdruck1 XOR Ausdruck2
```

Beide Ausdrücke müssen absolute, konstante Werte sein.

Ergebnis

Ein absoluter, konstanter Wert.

Rückgabe des Funktionsergebnisses

Bei Funktionen, die eine **asm-Anweisung** benutzen, erfolgt die Rückgabe des Funktionsergebnisses auf die folgende Weise:

Ergebnis	Geliefert in	Kommentar
Ordinal	AL (8-bit Werte) AX (16-bit Werte) DX:AX (32-bit Werte)	Integer, Char, Boolean und Aufzählungstypen
Real	DX:BX:AX	Real-Zahlen
8087	ST(0) auf dem 8087 Register-Stack	Single, Double, Extended, und Comp-Typen
Zeiger	DX:AX	
String	Ort der Zwischenspeicherung, auf den **@Result** verweist	

Short Jump

Opcode von einem Byte, gefolgt von einem Byte Distanz.

Near Jump

Opcode von einem Byte, gefolgt von zwei Bytes Distanz.

Short Jump mit invertierter Sprungbedingung

Ein Short Jump mit invertierter Sprungbedingung über einen darauffolgenden Near Jump, der zum eigentlichen Sprungziel führt. Diese Kombination besteht somit aus zwei Assembler-Befehlen und hat eine Länge von fünf Bytes.

Lokale Labels

Der Gültigkeitsbereich von lokalen Labels erstreckt sich vom Schlüsselwort **asm** bis zum zugehörigen end.

Lokale Labels:

- beginnen mit einem Klammeraffen (@)
- werden gefolgt von einem oder mehreren:
  Buchstaben (A..Z)
  Zahlen(0..9)
  Unterstrichen(_) oder
  Klammeraffen (@)
- und enden mit einem Doppelpunkt (:)

Lokale Labels beginnen mit einem Klammeraffen (@) und sind, da Pascal-Bezeichner keinen Klammeraffen enthalten dürfen, automatisch auf die Verwendung innerhalb von asm-Anweisungen beschränkt.

Da der Gültigkeitsbereich lokaler Labels ohnehin auf den asm-Block beschränkt ist, in dem sie verwendet werden, ist es zulässig, in verschiedenen asm-Blöcken gleichlautende Bezeichner für lokale Labels zu benutzen.

Lokale Labels müssen, ganz wie in Turbo Assembler, nicht explizit definiert werden.

Fehlermeldungen

Turbo Pascal generiert zwei Arten von Fehlermeldungen: Compiler-Fehlermeldungen und Laufzeit-Fehlermeldungen.

Compiler-Fehlermeldungen

Wenn ein Compiler-Fehler während einer Compilierung aus der IDE auftritt, aktiviert Turbo Pascal das Editor-Fenster und positioniert den Cursor an die Stelle, an der der Fehler im Quelltext aufgetreten ist.

Wenn der Fehler bei der Verwendung des Kommandozeilen-Compilers aufgetreten ist, zeigt Turbo Pascal die Fehlermeldung und -nummer sowie die entsprechende Zeile des Quelltextes. Ein Caret (^) indiziert den Fehler.

Laufzeit-Fehlermeldungen

Wenn Ihr Programm zur Laufzeit einen Fehler generiert, wird es beendet und eine Fehlermeldung ausgegeben:

```
Run-time error <nnn> at <xxxx:yyyy>
```

wobei:
- `nnn` ist die Fehlernummer
- `xxxx:yyyy` ist die Adresse des Laufzeit-Fehlers

Siehe auch

Laufzeit-Fehlermeldungen
Compiler-Fehlermeldungen

Anwenderfehler

Dieser Fehler wurde in einem modalen Dialogfenster angezeigt. Sie müssen das Dialogfenster schließem, bevor Sie den Fehler beheben können.

Um das Dialogfenster zu schließen, klicken Sie in OK (oder drücken Enter).

Titelleiste

Durch einen Doppelklick in der Titelleiste können Sie das Fenster vergrößern. Sie verschieben es, indem Sie die Titelleiste verschieben.

Bildlaufleisten

Bildlaufleisten sind horizontale oder vertikale Leisten, die sowohl Maus- als auch Tastaturanwendern anzeigen, wo sie sich innerhalb der Datei befinden. Wenn Sie eine Maus verwenden, verschieben Sie über diese Leisten den Inhalt der Fenster.

- Klicken Sie auf einen der Pfeile am Ende der BIldlaufleiste, und der Fensterinhalt wird um eine Zeile nach oben oder unten verschoben. Kontinuierliches Rollen erreichen Sie, indem Sie die Maustaste gedrückt halten.
- Wenn Sie in einen der schattierten Bereiche, die die Bildlaufleiste begrenzen, klicken, wird der Fensterinhalt um jeweils eine Seite verschoben.
- Verschieben Sie die Positionsmarkierung innerhalb der Bildlaufleiste, und im Fenster wird der Teil der Datei gezeigt, der der Position der Markierung entspricht.

Eingabeaufzeichnungsliste

Eingabeaufzeichnungslisten von Eingabefeldern zeigen den Text Ihrer letzten Eingaben in das Eingabefeld an.

Sie gelangen zur Eingabeaufzeichnungsliste, wenn das Eingabefeld rechts mit einem Pfeil nach unten gekennzeichnet ist.

Einträge in Eingabeaufzeichnungslisten können auch editiert werden.

Mit Esc verlassen Sie die Eingabeaufzeichnungsliste, ohne einen Eintrag zu übernehmen.

Der Turbo Pascal Editor

Es gibt verschiedene Möglichkeiten, ein Editorfenster zu aktivieren:

- Klicken Sie in das Fenster.

- Drücken Sie Alt+Bindestrich, und das **Systemmenü** des aktiven Editorfensters wird angezeigt. Wählen Sie dann den Befehl **Next**, um zu dem offenen Editorfenster zu wechseln.

- Mit Crtl+F6 wechseln Sie von einem offenen Fenster zum nächsten.

- Wählen Sie das Fenster aus der Liste der offenen Fenster des **Window-Menüs.**

In Editorfenstern geben Sie Text ein, wie Sie es gewohnt sind. Sie beenden eine Zeile mit Enter.

Eine Zeile darf maximal 249 Zeichen lang sein. Das Programm zeigt durch einen Piepton an, wenn diese Begrenzung überschritten wird. Beachten Sie, daß der Compiler nur bis zu 126 Zeichen pro Zeile erkennt.

Siehe auch

Editorbefehle

Zu diesem Thema ist keine Hilfe verfügbar.

Stellen Sie sicher, daß der Cursor auf dem Eintrag positioniert ist, zu dem Sie Hilfeinformationen möchten.

Zu diesem Thema ist noch keine Hilfe verfügbar.

Editorbefehle

Turbo Pascal's Editorbefehle sind in den folgenden Tabellen zusammengefasst:

Blockbefehle

Blockbefehle (CUA und Alternate)
Blockbefehle (Borland-Stil)
Erweiterung markierter Blöcke
Mehr über Blockbefehle

Weitere Befehle

Befehle zur Cursorbewegung
Einfügen- und Löschenoperationen
Mehr über Editorbefehle
Verschiedene Tastaturbefehle

Blockbefehle (CUA und alternativ)

Bewegung	CUA	Beide	Alternativ
Block löschen	Ctrl+Del	Ctrl+Del	-----
Block kopieren	Ctrl+Ins	Ctrl+Ins	-----
Block verschieben	Shift+Del	Shift+Del	-----
Block einfügen	Shift+Ins	Shift+Ins	-----
Block einlesen	Shift+Ctrl+R	-----	Ctrl+K R
Block speichern	Shift+Ctrl+W	-----	Ctrl+K W
Block einrücken	Shift+Ctrl+I	-----	Ctrl+K I
Block ausrücken	Shift+Ctrl+U	-----	Ctrl+K U

Siehe auch

Blockbefehle (Borland-Stil)
Erweiterung markierter Blöcke
Mehr über Blockbefehle

Erweiterung markierter Blöcke

Bewegung	CUA	Beide	Alternativ		
Ein Zeichen nach links		Shift+Left	-----		
Ein Zeichen nach rechts		Shift+Right	-----		
Zeilenende		Shift+End	Shift+End	-----	
Zeilenanfang		Shift+Home	Shift+Home	-----	
Gleiche Spalte nächste Zeile		Shift+Down	Shift+Down	-----	
Gleiche Spalte der letzten Zeile			Shift+Up	Shift+Up	-----
Eine Seite nach unten		Shift+PgDn	-----		
Eine Seite nach oben		Shift+PgUp	Shift+PgUp	-----	
Ein Wort nach links		Shift+Ctrl+Left	Shift+Ctrl+Left	-----	
Ein Wort nach rechts		Shift+Ctrl+Right		-----	
Dateiende		Shift+Ctrl+End	Shift+Ctrl+End	Shift+Ctrl+PgDn	
Dateianfang		Shift+Ctrl+Home		Shift+Ctrl+Home Shift+Ctrl+PgUp	

Siehe auch

Blockbefehle (CUA und alternativ)
Blockbefehle (Borland-Stil)
Mehr über Blockbefehle

Blockbefehle (Borland-Stil)

Beachten Sie: Turbo Pascal's Blockbefehle im Borland-Stil arbeiten nur mit dem alternativen Befehlssatz.

Befehl	Tasten	Funktion
Textauswahl einleiten	Ctrl+K B	Leitet die Textauswahl ein, die mit Kopieren (Ctrl K K) oder Löschen in die Zwischenablage (Ctrl K V) oder Abbrechen der Textauswahl (Ctrl K H) beendet wird.
Textauswahl abbrechen	Ctrl+K H	Beendet die Textauswahl, kein Text ist mehr markiert.
Text in Zwischenabl. kop.	Ctrl+K K	Kopiert den markierten Text in die Zwischenablage.
Text in Zwischenablage ausschneiden	Ctrl+K V	Schneidet den markierten Text in die Zwischenablage aus.
Text aus Zwischenablage einfügen	Ctrl+K C	Fügt den Inhalt der Zwischenablage in das aktive Editorfenster ein.

Siehe auch

Blockbefehle (CUA und alternativ)
Erweiterung markierter Blöcke
Mehr über Blockbefehle

Mehr über Blockbefehle

Block kopieren..... (Ctrl+Ins, dann Shift+Ins) Kopiert einen vorher markierten Block in die Zwischenablage und fügt ihn an der aktuellen Cursorposition wieder ein. Der Originalblock bleibt unverändert. Ist kein Block makiert, geschieht nichts.

Text kopieren..... (Ctrl+Ins) Kopiert den markierten Text in die Zwischenablage.

Text ausschneiden..... (Shift+Del) Löscht einen markierten Block in die Zwischenablage.

Block löschen..... (Ctrl+Del oder Ctrl+K Y) Löscht einen markierten Block. Mit Undo können Sie den Block "wiederherstellen".

Block bewegen.... (Shift+Del, dann Shift+Ins) Bewegt einen vorher markierten Block von seiner ursprünglichen Position in die Zwischenablage und fügt ihn an der Cursorposition wieder ein. Der Block wird an der ursprünglichen Position gelöscht. Falls kein Block markiert ist, wird nichts ausgeführt.

Zwischenablage auslesen..... (Shift+Ins) Holt den Inhalt der Zwischenablage zurück.

Block aus der Datei lesen..... (Shift+Ctrl+R oder Ctrl+K R) Liest eine Datei und fügt sie an der Cursorposition in den aktuellen Text als Block ein. Der eingelesene Text ist dann als Block markiert. Dieser Befehl fragt nach dem Namen der zu lesenden Datei. Sie können im Dateinamen Jokerzeichen verwenden, worauf ein Verzeichnis angezeigt wird. Die angegebene Datei kann eine beliebigen Dateinamen haben.

Block in eine Datei schreiben. (Shift+Ctrl+W oder Ctrl+KW) Schreibt einen markierten Block in eine Datei. Dieser Befehl fragt nach dem Namen der zu schreibenden Datei. Jeder beliebige Dateiname kann angegeben werden (wobei die voreingestellte Erweiterung .PAS lautet). Wenn Sie lieber einen Namen ohne Erweiterung verwenden wollen, können Sie einen Punkt (.) an den Dateinamen anhängen.

Beachten Sie: Wenn der angegebene Dateiname schon existiert, wird eine Warnung ausgegeben, bevor die Datei überschrieben wird. Wenn kein Block selektiert ist, passiert nichts.

Siehe auch

Blockbefehle (CUA und alternativ)
Blockbefehle (Borland-Stil)
Erweiterung markierter Blöcke

Cursorbewegung (CUA und alternativ)

Bewegung	CUA	Beide	Alternativ	
Cursor nach links	Links	Links	Ctrl+S	
Cursor nach rechts	Ctrl+Rechts	Rechts	Ctrl+D	
Wort nach links	Ctrl+Links	Ctrl+Links	Ctrl+A	
Wort nach rechts	Ctrl+Rechts	Ctrl+Z	Ctrl+Z	
Zeile nach oben	Oben	Oben	Ctrl+E	
Zeile nach unten	Unten	Unten	Ctrl+X	
Eine Zeile aufwärts rollen		Ctrl+W	Ctrl+W	Ctrl+W
Eine Zeile abwärts rollen		Ctrl+Z	Ctrl+Z	Ctrl+Z
Seite nach oben	PgUp	PgUp	Ctrl+R	
Seite nach unten	PgDn	PgDn	Ctrl+C	
Letzte Cursorposition	-----	-----	Ctrl+Q P	
Zeilenanfang	Home	Home	Ctrl+Q S	
Zeilenende	End	End	Ctrl+Q D	
Oberer Fensterrand	Ctrl+E	-----	Ctrl+Q E	
Unterer Fensterrand	Ctrl+X	-----	Ctrl+Q X	
Dateianfang	Ctrl+Home	Ctrl+Home	Ctrl+Q R,	Ctrl+PgUp
Dateiende	Ctrl+End	Ctrl+End	Ctrl+Q C,	Ctrl+PgDn

Einfüge- und Löschbefehle (CUA und alternativ)

Bewegung	CUA	Beide	Alternativ
Zeichen löschen	Del	Del	Ctrl+G
Zeichen links löschen	Backspace	Backspace	--------
Zeile köschen	Ctrl+Y	Ctrl+Y	--------
Rest der Zeile löschen	Shift+Ctrl+Y	--------	Ctrl+Q Y
Wort löchen	--------	--------	Ctrl+T
Zeile einfügen	Ctrl+N	--------	Ctrl+N
Einfügemodus ein/aus	Ins	Ins	Ctrl+V

Verschiedene Tastaturbefehle

Bewegung	CUA	Beide	Alternativ
Auto-Einzug ein/aus	--------	--------	Ctrl+O I
Aktuelle Compiler-Optionen	--------	--------	Ctrl+O O
Cursorpositionierung zwischen Tabulatoren	--------	--------	Ctrl+O R
Marke *N* suchen	Ctrl+*N**	--------	Ctrl+Q+*N**
Hilfe	F1	F1	--------
Hilfeindex	Shift+F1	Shift+F1	--------
Fenster maximieren	--------	--------	F5
Datei öffnen	--------	--------	F3
Optimaler Füllmodus ein/aus	--------	--------	Ctrl+O F
Begrenzerpaare suchen	Alt+[, Alt+]		Ctrl+Q [, Ctrl+Q]
Datei sichern	--------	--------	F2, Ctrl+K S
Suchen	--------	--------	Ctrl+Q F
Erneut suchen	F3	F3	--------
Suchen und Ersetzen	--------	--------	Ctrl+Q A
Letzten Compilerfehler zeigen	--------	--------	Ctrl+Q W
Marke setzen	Shift+Ctrl+*N**	--------	Ctrl+K+*N**
Tabulatormodus ein/aus	--------	--------	Ctrl+O T
Kontextbezogene Hilfe	Ctrl+F1	Ctrl+F1	--------
Beenden	Alt+F4	--------	Alt+X
Undo	Alt+Backspace	Alt+Backspace	--------
Ausrückmodus ein/aus	--------	--------	Ctrl+O U
Steuerzeichen einfügen	Ctrl+P **	Ctrl+P **	--------

* *N* ist eine Zahl zwischen 0 und 9.
** Steuerzeichen geben Sie folgendermaßen ein: Drücken Sie zuerst Ctrl+P und dann das gewünschte Steuerzeichen.

Mehr über Editorbefehle

Auto-Einzug..... Schaltet den automatischen Einzug von aufeinanderfolgenden Zeilen ein und aus. Wenn der Auto-Einzug aktiv ist, wird der Einzug der aktuellen Zeile in den folgenden Zeilen wiederholt. Sie können den automatischen Einzug auch über die Menüoption Options/Preferences/**AutoIndent** der IDE ein- und ausschalten.

Aktuelle Compiler-Optionen..... Fügt die aktuellen Einstellungen der Compiler-Optionen am Anfang des aktiven Fensters ein.

Cursorpositionierung zwischen Tabs... Wenn diese Option eingeschaltet ist, können Sie den Cursor zwischen die Tabulatorpositionen bewegen; ansonsten überspringt der Cursor mehrere Spalten bis zum nächsten Tabulator. Ctrl+O R wirkt wie ein Schalter.

Marke suchen..... Sucht bis zu 10 Marken im Text (wobei *N* eine Zahl zwischen 0 und 9 sein kann). Durch Eingabe von Ctrl+Q gefolgt von der Nummer der Marke kann der Cursor zu der vorher gesetzten Marke gebracht werden.

Datei öffnen..... Lädt eine vorhandene Datei in ein Editorfenster.

Optimaler Füllmodus..... Schaltet den optimalen Füllmodus ein oder aus. Wenn der optimale Füllmodus aktiv ist, wird jede Zeile mit einer möglichst minimalen Anzahl von Zeichen begonnen, wobei entsprechende Leerzeichen und Tabulatoren verwendet werden. Dadurch wird die Anzahl der Zeichen pro Zeile geringer.

Datei sichern..... Sichert die Datei und kehrt zum Editor zurück.

Marke setzen..... Setzt im Text bis zu 10 Marken; dazu wird Ctrl+K gefolgt von einer Ziffer von 0 bis 9 für die Marke eingegeben. Nach dem Markieren einer Textstelle können Sie von jeder Stelle der Datei mit dem Befehl Ctrl+Qn direkt zur Marke zurückkehren. Achten Sie darauf, für die Marke die gleiche Nummer zu verwenden. In jedem Fenster können bis zu zehn Marken gesetzt sein.

Letzten Compiler-fehler zeigen..... Markiert den letzten Syntaxfehler, den der Compiler während der letzten Übersetzung gefunden hat. Die zugehörige Fehlermeldung erscheint in der Statuszeile. Wenn die entsprechende Datei schon geschlossen ist, öffnet sie Turbo Pascal wieder und markiert die fehlerhafte Anweisung.

Tabulatormodus.. Die Verwendung von Tabualtorzeichen stellen Sie in der IDE mit der Option Options|Preferences|**Use Tab Character** ein.

Ausrückmodus..... Innerhalb der IDE schalten Sie über die Option Options|Preferences|**Backspace Unindents** den Ausrückmodus ein oder aus.

{Abs.PAS}

{Beispielcode für die Funktion Abs.}

```
var
  r: Real;
  i: Integer;
begin
  r := Abs(-2.3);        { 2.3 }
  i := Abs(-157);        { 157 }
end.
```

{Addr.PAS}

{Beispielcode für die Funktion Addr.}

```
var
  P: Pointer;
begin
  P := Addr(P);          { Now points to itself }
end.
```

{Append.PAS}

{Beispielcode für die Prozedur Append.}

```
var F: Text;
begin
  Assign(F, 'TEST.TXT');
  Rewrite(F);               { Create new file }
  Writeln(F, 'original text');
  Close(F);                 { Close file, save changes }
  Append(F);                { Add more text onto end }
  Writeln(F, 'appended text');
  Close(F);                 { Close file, save changes }
end.
```

{Arctan.PAS}

{Beispielcode für die Funktion ArcTan.}

```
var
  R: Real;
begin
  R := ArcTan(Pi);
end.
```

{Assign.PAS}

{Beispielcode für die Prozedur Assign. Versuchen Sie, dieses Programm von DOS zu PRN, in eine Datei usw. umzuleiten.}

```
var F: Text;
begin
  Assign(F, '');            { Standard output }
  Rewrite(F);
  Writeln(F, 'standard output...');
  Close(F);
end.
```

{Blockrd.PAS}

{Beispielcode für dieProzeduren BlockRead und BlockWrite.}

```
program CopyFile;
{ Simple, fast file copy program with NO error-checking }
var
  FromF, ToF: file;
  NumRead, NumWritten: Word;
  Buf: array[1..2048] of Char;
begin
  Assign(FromF, ParamStr(1));     { Open input file }
  Reset(FromF, 1);          { Record size = 1 }
  Assign(ToF, ParamStr(2));{ Open output file }
  Rewrite(ToF, 1);          { Record size = 1 }
  Writeln('Copying ', FileSize(FromF), ' bytes...');
  repeat
    BlockRead(FromF, Buf, SizeOf(Buf), NumRead);
    BlockWrite(ToF, Buf, NumRead, NumWritten);
  until (NumRead = 0) or (NumWritten <> NumRead);
  Close(FromF);
  Close(ToF);
end.
```

{Chdir.PAS}

{Beispielcode für die Prozedur ChDir.}

```
begin
  {$I-}
  { Get directory name from command line }
  ChDir(ParamStr(1));
  if IOResult <> 0 then
    Writeln('Cannot find directory');
end.
```

{Close.PAS}

{Beispielcode für die Prozedur Close.}

```
var F: file;
begin
  Assign(F, '\AUTOEXEC.BAT');     { open file }
  Reset(F, 1);
  Writeln('File size = ', FileSize(F));
  Close(F);                  { Close file }
end.
```

{Concat.PAS}

{Beispielcode für die Funktion Concat.}

```
var
  S: String;
begin
  S := Concat('ABC', 'DEF');      { 'ABCDE' }
end.
```

{Copy.PAS}

{Beispielcode für die Funktion Copy.}

```
var S: String;
begin
  S := 'ABCDEF';
  S := Copy(S, 2, 3) { 'BCD' }
end.
```

{Cos.PAS}

{Beispielcode für die Funktion Cos.}

```
var R: Real;
begin
  R := Cos(Pi);
end.
```

{Dec.PAS}

{Beispielcode für die Prozedur Dec.}

```
var
  IntVar: Integer;
  LongintVar: Longint;
begin
  Dec(IntVar);              { IntVar := IntVar - 1 }
  Dec(LongintVar, 5);       { LongintVar := LongintVar - 5 }
end.
```

{Diskfree.PAS}

{Beispielcode für die Funktion DiskFree. Diese Funktion verwendet die **WinDos Unit**.}

```
begin
  Writeln(DiskFree(0) div 1024, ' Kbytes free ');
end.
```

{Disksize.PAS}

{Beispielcode für die Funktion DiskSize. Diese Funktion verwendet **WinDos Unit**.}

```
begin
  Writeln(DiskSize(0) div 1024, ' Kbytes capacity');
end.
```

{Dispose.PAS}

{Beispielcode für die Prozeduren New und Dispose.}

```
type
  Str18 = string[18];
var
  P: ^Str18;
begin
  New(P);
  P^ := 'Now you see it...';
  Dispose(P);         { Now you don't... }
end.
```

{DOSVrsn.PAS}

{Beispielcode für die Funktion DOSVersion. Diese Funktion verwendet die **WinDos Unit**.}

```
var
  Ver: Word;
begin
  Ver := DosVersion;
  Writeln('This is DOS version ', Lo(Ver), '.', Hi(Ver));
end.
```

{Eof.PAS}

{Beispielcode für die Funktionen Eof, Read und Write (Textdateien).}

```
var
  F: Text;
  Ch: Char;
begin
  { Get file to read from command line }
  Assign(F, ParamStr(1));
  Reset(F);
  while not Eof(F) do
  begin
    Read(F, Ch);
    Write(Ch);        { Dump text file }
  end;
end.
```

{Erase.PAS}

{Beispielcode für Erase. }

```
var
  F:  file;
  Ch: Char;
begin
  { Get file to delete from command line }
  Assign(F, ParamStr(1));
  {$I-}
  Reset(F);
  {$I+}
  if IOResult <> 0 then
    Writeln('Cannot find ', ParamStr(1))
  else
  begin
    Close(F);
    Write('Erase ', ParamStr(1), '? ');
    Readln(Ch);
    if UpCase(ch) = 'Y' then
      Erase(F);
  end;
end.
```

{Filesize.PAS}

{Beispielcode für die Funktion FileSize. }

```
var
  F: file of Byte;
begin
  { Get file name from command line }
  Assign(F, ParamStr(1));
  Reset(F);
  Writeln('File size in bytes: ', FileSize(F));
  Close(F);
end.
```

{Fillchar.PAS}

{Beispielcode für die Prozedur FillChar.}

```
var
  S: string[80];
begin
  { Set a string to all spaces }
  FillChar(S, SizeOf(S), ' ');
  S[0] := #80;        { Set length byte }
end.
```

{Findfrst.PAS}

{Beispielcode für die Prozeduren FindFirst und FindNext. Die **WinDos Unit** wird verwendet.}

```
var
  DirInfo: SearchRec;
begin
  FindFirst('*.PAS', Archive, DirInfo);       { Same as DIR *.PAS }
  while DosError = 0 do
  begin
    Writeln(DirInfo.Name);
    FindNext(DirInfo);
  end;
end.
```

{Frac.PAS}

{Beispielcode für die Funktion Frac.}

```
var
  R: Real;
begin
  R := Frac(123.456);     { 0.456 }
  R := Frac(-123.456);    { -0.456 }
end.
```

{Getfattr.PAS}

{Beispielcode für die Prozedur GetFAttr. Diese Prozedur verwendet **WinDos Unit**.}

```
var
  F: file;
  Attr: Word;
begin
  { Get file name from command line }
  Assign(F, ParamStr(1));
  GetFAttr(F, Attr);
  Writeln(ParamStr(1));
  if DosError <> 0 then
    Writeln('DOS error code = ', DosError)
  else
  begin
    Write('Attribute = ', Attr);
    { Determine file attribute type using flags in WinDos unit }
    if Attr and faReadOnly <> 0 then
      Writeln('Read only file');
    if Attr and faHidden <> 0 then
      Writeln('Hidden file');
    if Attr and faSysFile <> 0 then
      Writeln('System file');
    if Attr and faVolumeID <> 0 then
      Writeln('Volume ID');
    if Attr and faDirectory <> 0 then
      Writeln('Directory name');
    if Attr and faArchive <> 0 then
      Writeln('Archive (normal file)');
  end; { else }
end.
```

{Hi.PAS}

{Beispielcode für die Funktion Hi.}

```
var W: Word;
begin
  W := Hi($1234);    { $12 }
end.
```

{Inc.PAS}

{Beispielcode für die Prozedur Inc.}

```
var
  IntVar: Integer;
  LongintVar: Longint;
begin
  Inc(IntVar);        { IntVar := IntVar + 1 }
  Inc(LongintVar, 5); { LongintVar := LongintVar + 5 }
end.
```

{Insert.PAS}

{Beispielcode für die Prozedur Insert.}

```
var
  S: String;
begin
  S := 'Honest Lincoln';
  Insert('Abe ', S, 8);           { 'Honest Abe Lincoln' }
end.
```

{Int.PAS}

{Beispielcode für die Funktion Int.}

```
var R: Real;
begin
  R := Int(123.456);     { 123.0 }
  R := Int(-123.456);    { -123.0 }
end.
```

{IOResult.PAS}

{Beispielcode für die Funktion IOResult.}

```
var F: file of Byte;
begin
  { Get file name command line }
  Assign(F, ParamStr(1));
  {$I-}
  Reset(F);
  {$I+}
  if IOResult = 0 then
    Writeln('File size in bytes: ', FileSize(F))
  else
    Writeln('File not found');
end.
```

{Length.PAS}

{Beispielcode für die Funktion Length.}

```
var
  F: Text;
  S: String;
begin
  Assign(F, 'GARY.PAS');
  Reset(F);
  Readln(F, S);
  Writeln('"', S, '"');
  Writeln('length = ', Length(S));
end.
```

{Lo.PAS}

{Beispielcode für die Funktion Lo.}

```
var W: Word;
begin
  W := Lo($1234);    { $34 }
end.
```

{Maxavail.PAS}

{Beispielcode für die Funktion MaxAvail.}

```
type
  FriendRec = record
    Name: string[30];
    Age: Byte;
  end;
var
  P: Pointer;
begin
  if MaxAvail < SizeOf(FriendRec) then
    Writeln('Not enough memory')
  else
  begin
    { Allocate memory on heap }
    GetMem(P, SizeOf(FriendRec));
    { ... }
  end;
end.
```

{Memavail.PAS}

{Beispielcode für die Funktion MemAvail.}

```
begin
  Writeln(MemAvail, ' bytes available');
  Writeln('Largest free block is ', MaxAvail, ' bytes');
end.
```

{Mkdir.PAS}

{Beispielcode für die Prozedur MkDir.}

```
begin
  {$I-}
  { Get directory name from command line }
  MkDir(ParamStr(1));
  if IOResult <> 0 then
    Writeln('Cannot create directory')
  else
    Writeln('New directory created');
end.
```

{Move.PAS}

{Beispielcode für die Prozedur Move.}

```
var
  A: array[1..4] of Char;
  B: Longint;
begin
  Move(A, B, SizeOf(A));    { SizeOf = safety! }
end.
```

{Parcount.PAS}

{Beispielcode für die Funktion ParamCount.}

```
begin
  if ParamCount < 1 then
    Writeln('No parameters on command line')
  else
    Writeln(ParamCount, ' parameter(s)');
end.
```

{Parstrng.PAS}

{Beispielcode für die Funktion Paramstr.}

```
var I: Word;
begin
  for I := 1 to ParamCount do
    Writeln(ParamStr(I));
end.
```

{Pos.PAS}

{Beispielcode für die Funktion Pos.}

```
var S: String;
begin
  S := '   123.5';
  { Convert spaces to zeroes }
  while Pos(' ', S) > 0 do
    S[Pos(' ', S)] := '0';
end.
```

{Ptr.PAS}

{Beispielcode für die Funktion Ptr.}

```
var P: ^Byte;
begin
  P := Ptr($40, $49);
  Writeln('Current video mode is ', P^);
end.
```

{Reset.PAS}

{Beispielcode für die Prozedur Reset.}

```
function FileExists(FileName: String): Boolean;
{ Boolean function that returns True if the file exists;otherwise,
   it returns False. Closes the file if it exists. }
var
  F: file;
begin
  {$I-}
  Assign(F, FileName);
  Reset(F);
  Close(F);
  {$I+}
  FileExists := (IOResult = 0) and (FileName <> '');
end;  { FileExists }

begin
  if FileExists(ParamStr(1)) then {Get file name from command line}
    Writeln('File exists')
  else
    Writeln('File not found');
end.
```

{Rewrite.PAS}

{Beispielcode für die Prozedur Rewrite.}

```
var F: Text;
begin
  Assign(F, 'NEWFILE.$$$');
  Rewrite(F);
  Writeln(F, 'Just created file with this text in it...');
  Close(F);
end.
```

{Runerror.PAS}

{Beispielcode für die Prozedur RunError.}

```
begin
{$IFDEF Debug}
  if P = nil then
    RunError(204);
{$ENDIF}
end.
```

{Setfattr.PAS}

{Beispielcode für die Prozedur SetFAttr. Diese Prozedur verwendet die **WinDos Unit**.}

```
var
  F: file;
begin
  Assign(F, 'C:\AUTOEXEC.BAT');
  SetFAttr(F, faHidden);    { Uh-oh }
  Readln;
  SetFAttr(F, faArchive);   { Whew! }
end.
```

{Settxtbf.PAS}

{Beispielcode für die Prozedur SetTextBuf.}

```
var
  F: Text;
  Ch: Char;
  Buf: array[1..10240] of Char;         { 10K buffer }
begin
  { Get file to read from command line }
  Assign(F, ParamStr(1));
  { Bigger buffer for faster reads }
  SetTextBuf(F, Buf);
  Reset(F);
  { Dump text file onto screen }
  while not Eof(f) do
  begin
    Read(F, Ch);
    Write(Ch);
  end;
end.
```

{Sin.PAS}

{Beispielcode für die Funktion Sin.}

```
var
  R: Real;
begin
  R := Sin(Pi);
end.
```

{Sizeof.PAS}

{Beispielcode für die Funktion SizeOf.}

```
type
  CustRec = record
    Name: string[30];
    Phone: string[14];
  end;
var
  P: ^CustRec;
begin
  GetMem(P, SizeOf(CustRec));
end.
```

{Str.PAS}

{Beispielcode für die Prozedur Str.}

```
function IntToStr(I: Longint): String;
{ Convert any integer type to a string }
var
  S: string[11];
begin
  Str(I, S);
  IntToStr := S;
end;
begin
  Writeln(IntToStr(-5322));
end.
```

{Swap.PAS}

{Beispielcode für die Funktion Swap.}

```
var
  X: Word;
begin
  X := Swap($1234);    { $3412 }
end.
```

{Val.PAS}

{Beispielcode für die Prozedur Val.}

```
var I, Code: Integer;
begin
{ Get text from command line }
  Val(ParamStr(1), I, Code);
  { Error during cnversion to integer? }
  if code <> 0 then
    Writeln('Error at positon: ', Code)
  else
    Writeln('Value = ', I);
end.
```

{Chr.PAS}

{Beispielcode für die Funktion Chr.}

```
uses WinCrt;

var
  I: Integer;
begin
  for I := 32 to 126 do Write(Chr(I));
end.
```

{CSeg.PAS}

{Beispielcode für die Funktionen CSeg, DSeg, SSeg, SPtr, Ofs und Seg.}

```
uses WinCrt;

procedure WriteHexWord(w: Word);
const
  hexChars: array [0..$F] of Char =
    '0123456789ABCDEF';
begin
  Write(hexChars[Hi(w) shr 4],
        hexChars[Hi(w) and $F],
        hexChars[Lo(w) shr 4],
        hexChars[Lo(w) and $F]);
end;
var
  i: Integer;
begin
  Write('The current code segment is $');
  WriteHexWord(CSeg); Writeln;
  Write('The global data segment is $');
  WriteHexWord(DSeg); Writeln;
  Write('The stack segment is $');
  WriteHexWord(SSeg); Writeln;
  Write('The stack pointer is at $');
  WriteHexWord(SPtr); Writeln;
  Write('i is at offset $');
  WriteHexWord(Ofs(i));
  Write(' in segment $');
  WriteHexWord(Seg(i));
end.
```

{Delete.PAS}

{Beispielcode für die Prozedur Delete.}

```
uses WinCrt;

var
  s: string;
begin
  s := 'Honest Abe Lincoln';
  Delete(s,8,4);
  Writeln(s); { 'Honest Lincoln' }
end.
```

{Eoln.PAS}

{Beispielcode für die Funktion Eoln.}

```
uses WinCrt;

begin
{ Tells program to wait for keyboard input }
  Writeln(Eoln);
end.
```

{Exit.PAS}

{Beispielcode für die Prozedur Exit.}

```
uses WinCrt;

procedure WasteTime;
begin
  repeat
    if KeyPressed then Exit;
    Write('Xx');
  until False;
end;
begin
  WasteTime;
end.
```

{Exp.PAS}

{Beispielcode für die Funktion Exp.}

```
uses WinCrt;

begin
  Writeln('e = ',Exp(1.0));
end.
```

{FilePos.PAS}

{Beispielcode für die Funktionen FilePos, FileSize und Seek.}

```
uses WinCrt;

var
  f: file of Byte;
  size : Longint;
begin
  { Get file name from command line }
  Assign(f, ParamStr(1));
  Reset(f);
  size := FileSize(f);
  Writeln('File size in bytes: ',size);
  Writeln('Seeking halfway into file...');
  Seek(f,size div 2);
  Writeln('Position is now ',FilePos(f));
  Close(f);
end.
```

{Flush.PAS}

{Beispielcode für die Prozedur Flush.}

```
uses WinDos, WinCrt;

procedure ReportError(s : String);
  { Redirect output to the DOS standard
   error handle and emit an error message,
   then halt.
   The file "output" must be flushed before
   its handle is changed, or some earlier
   output may appear with the error
   message.}
  begin
  Flush(output); { Must flush current output}
  { Redirect output to standard error handle}
  TTextRec(output).handle := 2;
  Writeln(s);
  Halt(1);
  end;
begin
ReportError('An error occurred');
end.
```

{FreeMem.PAS}

{Beispielcode für die Prozeduren FreeMem, GetMem und MaxAvail.}

```
uses WinCrt;

type
  TFriendRec = record
                 Name: string[30];
                 Age : Byte;
               end;
var
  p: pointer;
begin
  if MaxAvail < SizeOf(TFriendRec) then
    Writeln('Not enough memory')
  else
  begin
    { Allocate memory on heap }
    GetMem(p, SizeOf(TFriendRec));
    { ...}
    { ...Use the memory... }
    { ...}
    { Then free it when done }
    FreeMem(p, SizeOf(TFriendRec));
  end;
end.
```

{GetDir.PAS}

{Beispielcode für die Prozedur GetDir.}

```
uses WinCrt;

var
  s : String;
begin
  GetDir(0,s); { 0 = Current drive }
  Writeln('Current drive and directory: ',
          s);
end.
```

{Halt.PAS}

{Beispielcode für die Prozedur Halt.}

```
uses WinCrt;

begin
  if 1 = 1 then
    begin
      if 2 = 2 then
        begin
          if 3 = 3 then
            begin
              Halt(1); { Halt right here! }
            end;
        end;
    end;
  Writeln('This will not be executed');
end.
```

{Ln.PAS}

{Beispielcode für die Funktion Ln.}

```
uses WinCrt;

var
  e : real;
begin
  e := Exp(1.0);
  Writeln('ln(e) = ',ln(e));
end.
```

{Odd.PAS}

{Beispielcode für die Funktion Odd.}

```
uses WinCrt;

begin
  if Odd(5) then
    Writeln('5 is odd')
  else
    Writeln('Something is odd');
end.
```

{Ord.PAS}

{Beispielcode für die Funktion Ord.}

```
uses WinCrt;

type
  Colors = (RED,BLUE,GREEN);
begin
  Writeln('BLUE has an ordinal value of ',
          Ord(BLUE));
  Writeln('The ASCII code for "c" is ',
          Ord('c'), ' decimal');
end.
```

{Pi.PAS}

{Beispielcode für die Funktion Pi.}

```
uses WinCrt;

begin
  Writeln('Pi = ',Pi);
end.
```

{Pred.PAS}

{Beispielcode für die Funktionen Pred und Succ.}

```
uses WinCrt;

type
  Colors = (RED,BLUE,GREEN);
begin
  Writeln('The predecessor of 5 is ',
          Pred(5));
  Writeln('The successor of 10 is ',
          Succ(10));
  if Succ(RED) = BLUE then
    Writeln('In the type Colors, RED is ',
            'the predecessor of BLUE.');
end.
```

{Random.PAS}

{Beispielcode für die Funktionen Random und Randomize.}

```
uses Crt;

begin
  Randomize;
  repeat
    { Write text in random colors }
    TextAttr := Random(256);
    Write('!');
  until KeyPressed;
end.
```

{ReadIn.PAS}

{Beispielcode für die Prozeduren Readln und Writeln.}

```
uses WinCrt;

var
  s : String;
begin
  Write('Enter a line of text: ');
  Readln(s);
  Writeln('You typed: ',s);
  Writeln('Hit <Enter> to exit');
  Readln;
end.
```

{Rename.PAS}

{Beispielcode für die Prozedur Rename.}

```
uses WinCrt;

var
  f : file;
begin
  { Rename a file. Old and new names
    given on command line. }
  if ParamCount <> 2 then
  begin
    Writeln('Wrong number of parameters');
    Halt(1);
  end;
  Assign(f,ParamStr(1));
  Writeln('Renaming ',ParamStr(1),
          ' to ',ParamStr(2));
  Rename(f,ParamStr(2));
end.
```

{RmDir.PAS}

{Beispielcode für die Prozedur RmDir.}

```
uses WinCrt;

begin
  {$I-}
  { Get directory name from command line }
  RmDir(ParamStr(1));
  if IOResult <> 0 then
    Writeln('Cannot remove directory')
  else
    Writeln('directory removed');
end.
```

{Round.PAS}

{Beispielcode für die Funktion Round.}

```
uses WinCrt;

begin
  Writeln(1.4, ' rounds to ', Round(1.4));
  Writeln(1.5, ' rounds to ', Round(1.5));
  Writeln(-1.4, ' rounds to ', Round(-1.4));
  Writeln(-1.5, ' rounds to ', Round(-1.5));
end.
```

{SeekEof.PAS}

{Beispielcode für die Funktionen SeekEof und SeekEoln.}

```
uses WinCrt;

var
  f : Text;
  i, j : Integer;
begin
  Assign(f,'TEST.TXT');
  Rewrite(f);
  { Create a file with 8 numbers and some
    whitespace at the ends of the lines }
  Writeln(f,'1 2 3 4 ');
  Writeln(f,'5 6 7 8 ');
  Reset(f);
  { Read the numbers back. SeekEoln returns
    TRUE if there are no more numbers on
    the current line; SeekEof returns TRUE
    if there is no more text (other than
    whitespace) in the file. }
  while not SeekEof(f) do
  begin
    if SeekEoln(f) then
      Readln; { Go to next line }
    Read(f,j);
    Writeln(j);
  end;
end.
```

{Sqr.PAS}

{Beispielcode für die Funktionen Sqr und Sqrt.}

```
uses WinCrt;

begin
  Writeln('5 squared is ', Sqr(5));
  Writeln('The square root of 2 is ',
          Sqrt(2.0));
end.
```

{Trunc.PAS}

{Beispielcode für die Funktion Trunc.}

```
uses WinCrt;

begin
  Writeln(1.4, '  becomes ', Trunc(1.4));
  Writeln(1.5, ' becomes ', Trunc(1.5));
  Writeln(-1.4, ' becomes ', Trunc(-1.4));
  Writeln(-1.5, ' becomes ', Trunc(-1.5));
end.
```

{Truncate.PAS}

{Beispielcode für die Prozedur Truncate.}

```
uses WinCrt;

var
  f: file of Integer;
  i,j: Integer;
begin
  Assign(f,'TEST.INT');
  Rewrite(f);
  for i := 1 to 6 do
    Write(f,i);
  Writeln('File before truncation:');
  Reset(f);
  while not Eof(f) do
  begin
    Read(f,i);
    Writeln(i);
  end;
  Reset(f);
  for i := 1 to 3 do
    Read(f,j); { Read ahead 3 records }
  Truncate(f); { Cut file off here }
  Writeln;
  Writeln('File after truncation:');
  Reset(f);
  while not Eof(f) do
  begin
    Read(f,i);
    Writeln(i);
  end;
  Close(f);
  Erase(f);
end.
```

{UpCase.PAS}

{Beispielcode für die Funktion UpCase.}

```
uses WinCrt;

var
  s : string;
  i : Integer;
begin
  Write('Enter a string: ');
  Readln(s);
  for i := 1 to Length(s) do
    s[i] := UpCase(s[i]);
  Writeln('Here it is in all uppercase: ',s);
end.
```

{CreateDr.PAS}

{Beispielcode für die Prozedur CreateDir.}

```
uses WinCrt, WinDOS;

const
  Create: PChar = 'C:\NEWDIR';
  CDrive: Byte = 3;

begin
  CreateDir(Create);
  Writeln('There is now a new directory at ', Create,'.');
end.
```

{FileExp.PAS}

{Beispielcode für die Funktion FileExpand.}

```
uses WinCrt, WinDOS;

const
  MyFile: PChar = 'TEST.FIL';

var
  Where: PChar;

begin
  GetMem(Where, 80);
  FileExpand(Where, MyFile);
  Writeln(MyFile, ' is in the current directory at ', Where, '.');
end.
```

{GetArgCo.PAS}

{Beispielcode für die Funktion GetArgCount.}

```
uses WinCrt, WinDOS;

var
  TheArgCount:  Integer;

begin
  TheArgCount := GetArgCount;
  Writeln('There were ', TheArgCount,' arguments to the prompt (in
addition to the');
  Writeln('program name) when running this program.');
end.
```

{GetArgSt.PAS}

{Beispielcode für die Funktion GetArgStr.}

```
uses WinCrt, WinDOS;

var
  ArgN: PChar;

begin
  GetMem(ArgN, 20);
  GetArgStr(ArgN, 1, 19);
  Writeln('The 1st argument to the prompt (in addition to the');
  Writeln('program name) was ''', ArgN, '''.');
end.
```

{GetCBrk.PAS}

{Beispielcode für die Prozeduren GetCBreak und SetCBreak.}

```
uses WinDos, WinCrt;

const
  OffOn : array [Boolean] of String[3] =
    ('off','on');
var
  cb : Boolean;
begin
  GetCBreak(cb);
  Writeln('Control-Break checking is ',
          OffOn[cb]);
  cb := not(cb);
  Writeln('Turning Control-Break checking ',
          OffOn[cb]);
  SetCBreak(cb);
end.
```

{GetCurDr.PAS}

{Beispielcode für die Funktion GetCurDir.}

```
uses WinCrt, WinDOS;

const
  CDrive:Byte = 3;

var
  CurDir: PChar;

begin
  GetMem(CurDir, 80);
  GetCurDir(CurDir, CDrive);
  Writeln('The current directory on drive c is ', CurDir,'.');
end.
```

{GetDate.PAS}

{Beispielcode für die Prozedur GetDate.}

```
uses WinDos, WinCrt;

const
  days : array [0..6] of String[9] =
    ('Sunday','Monday','Tuesday',
     'Wednesday','Thursday','Friday',
     'Saturday');
var
  y, m, d, dow : Word;
begin
  GetDate(y,m,d,dow);
  Writeln('Today is ', days[dow],', ',
          m:0, '/', d:0, '/', y:0);
end.
```

{GetEnvVa.PAS}

{Beispielcode für die Funktion GetEnvVar.}

```
uses WinCrt, WinDOS;

var
  EnvironmentVar: PChar;

begin
  GetMem(EnvironmentVar, 255);
  EnvironmentVar := GetEnvVar('PATH');
  Writeln('The PATH is currently: ');
  Writeln(EnvironmentVar);
end.
```

{GetFTime.PAS}

{Beispielcode für die Prozeduren GetFTime, PackTime, SetFTime und UnpackTime.}

```
uses WinDos, WinCrt;

var
  f: text;
  h, m, s, hund : Word; { For GetTime}
  ftime : Longint; { For Get/SetFTime}
  dt : DateTime; { For Pack/UnpackTime}
function LeadingZero(w : Word) : String;
var
  s : String;
begin
  Str(w:0,s);
  if Length(s) = 1 then
    s := '0' + s;
  LeadingZero := s;
end;
begin
  Assign(f, 'TEST.TXT');
  GetTime(h,m,s,hund);
  Rewrite(f); { Create new file }
  GetFTime(f,ftime); { Get creation time }
  Writeln('File created at ',LeadingZero(h),
          ':',LeadingZero(m),':',
          LeadingZero(s));
  UnpackTime(ftime,dt);
  with dt do
    begin
      Writeln('File timestamp is ',
              LeadingZero(hour),':',
              LeadingZero(min),':',
              LeadingZero(sec));
      hour := 0;
      min := 1;
      sec := 0;
      PackTime(dt,ftime);
      Writeln('Setting file timestamp ',
              'to one minute after midnight');
      Reset(f); { Reopen file for reading }
      { (Otherwise, close will update time) }
      SetFTime(f,ftime);
    end;
  Close(f);   { Close file }
end.
```

{GetIntVc.PAS}

{Beispielcode für die Prozeduren GetIntVec und SetIntVec.}

```
uses Dos;

const
  BreakFlag : Boolean = FALSE;
var
  Int1BSave : Pointer;
{$F+}
procedure BreakHandler; interrupt;
  begin
  BreakFlag := TRUE;
  end;
{$F-}
begin
  GetIntVec($1B,Int1BSave);
  SetIntVec($1B,Addr(BreakHandler));
  Writeln('Press Ctrl-Break to exit');
  repeat until BreakFlag;
  SetIntVec($1B,Int1BSave);
end.
```

{GetTime.PAS}

{Beispielcode für die Prozedur GetTime.}

```
uses WinDos, WinCrt;

var
  h, m, s, hund : Word;
function LeadingZero(w : Word) : String;
var
  s : String;
begin
  Str(w:0,s);
  if Length(s) = 1 then
    s := '0' + s;
  LeadingZero := s;
end;
begin
  GetTime(h,m,s,hund);
  Writeln('It is now ',LeadingZero(h),':',
          LeadingZero(m),':',LeadingZero(s),
          '.',LeadingZero(hund));
end.
```

{GetVerify.PAS}

{Beispielcode für die Prozeduren GetVerify und SetVerify.}

```
uses Dos;

const
  OffOn : array [Boolean] of String[3] =
    ('off','on');
var
  v : Boolean;
begin
  GetVerify(v);
  Writeln('Write verify is ', OffOn[v]);
  v := not(v);
  Writeln('Turning write verify ',
          OffOn[v]);
  SetVerify(v);
end.
```

{Intr.PAS}

{Beispielcode für die Prozedur Intr.}

```
uses DOS;

const
  msg : String = 'Hello, world!$';
var
  regs : Registers;
begin
  regs.ah := 9; { Print string w/$ at end }
  regs.ds := Seg(msg); { Set DS:DX to addr }
  regs.dx := Ofs(msg[1]); { of first char }
  Intr($21,regs); { Call DOS }
end.
```

{MsDos.PAS}

{Beispielcode für die Prozedur MsDos.}

```
uses DOS;

const
  msg : String = 'Hello, world!$';
var
  regs : Registers;
begin
  regs.ah := 9; { Print string w/$ at end }
  regs.ds := Seg(msg); { Set DS:DX to addr }
  regs.dx := Ofs(msg[1]); { of first Char }
  MsDos(regs); { Call DOS }
end.
```

{RemoveDr.PAS}

{Beispielcode für die Prozedur RemoveDir.}

```
uses WinCrt, WinDOS;

const
  Remove: PChar = 'C:\NEWDIR';
  CDrive: Byte = 3;

begin
  RemoveDir(Remove);
  Writeln('There is no longer a directory at ', Remove,'.');
end.
```

{SetCurDr.PAS}

{Beispielcode für die Prozedur SetCurDir.}

```
uses WinCrt, WinDOS;

const
  ChangeTo: PChar = 'C:\';
  CDrive: Byte = 3;

var
  CurDir: PChar;

begin
  GetMem(CurDir, 80);
  SetCurDir(ChangeTo);
  GetCurDir(CurDir, CDrive);
  Writeln('The current directory on drive c is ', CurDir,'.');
end.
```

{SetDate.PAS}

{Beispielcode für die Prozedur SetDate.}

```
uses WinDos;

begin
  { Set the system date to 2/10/89 }
  SetDate(1989,2,10);
end.
```

{SetTime.PAS}

{Beispielcode für die Prozedur SetTime.}

```
uses Dos;

begin
  { Set system clock to 12:01 AM }
  SetTime(0,1,0,0);
end.
```

{InitWCrt.PAS}

{Beispielcode für die Prozedur InitWinCrt.}

```
uses Strings, WinCrt, WinDOS;

begin
  StrCopy(WindowTitle,'InitWinCRT created CRT Window');
  Writeln('This is the my manually created CRT window.');
  Writeln('Notice the title.');
  InitWinCRT;
end.
```

{DoneWCrt.PAS}

{Beispielcode für die Prozedur DoneWinCrt.}

```
uses WinCrt;

begin
  Writeln('This is the first CRT window.');
  Writeln('  Please hit return.');
  Readln;
  DoneWinCRT;
  Writeln('This is the second CRT window.');
end.
```

{WriteBuf.PAS}

{Beispielcode für die Prozedur WriteBuf.}

```
uses WinCrt;

var
  MyBuffer: PChar;

begin
  GetMem(MyBuffer, 80);
  MyBuffer := 'This is an example of WriteBuf';
  WriteBuf(MyBuffer, 30);
end.
```

{WriteChr.PAS}

{Beispielcode für die Prozedur WriteChar.}

```
uses WinCrt;

begin
  Write('ABCDE');
  WriteChar('F');
end.
```

{KeyPress.PAS}

{Beispielcode für die Funktion KeyPressed.}

```
uses WinCrt;

begin
  repeat
    Write('Xx');
  until KeyPressed;
end.
```

{ReadKey.PAS}

{Beispielcode für die Funktion ReadKey.}

```
uses WinCrt;

var
  C: Char;

begin
  Writeln('Please press a key');
  C := Readkey;
  Writeln(' You pressed ', C, ', whose ASCII value is ', Ord(C), '.');
end.
```

{ReadBuf.PAS}

{Beispielcode für die Funktion ReadBuf.}

```
uses WinCrt;

var
  C: PChar;

begin
  GetMem(C, 20);
  C := #0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0#0;
  Writeln('Type a phrase up to 20 characters long:');
  ReadBuf(C, 20);
  Writeln(' You typed: ');
  Writeln(C);
end.
```

{GotoXY.PAS}

{Beispielcode für die Prozedur GotoXY.}

```
uses WinCrt;

var
  C: PChar;

begin
  GotoXY(10,10);
  Writeln('Hello');
end.
```

{WhereX.PAS}

{Beispielcode für die Funktion WhereX.}

uses WinCrt;

```
begin
  Write('The number in this sentence is in the #');
  Writeln(WhereX, ' column in this window.');
end.
```

{WhereY.PAS}

{Beispielcode für die Funktion WhereY.}

```
uses WinCrt;

begin
  Writeln;
  Writeln;
  Write('This sentence is on the #');
  Writeln(WhereY, ' line in this window.');
end.
```

{ClrScr.PAS}

{Beispielcode für die Prozedur ClrScr.}

```
uses WinCrt;

begin
  Writeln('Hello.   Please any key...');
  Readln;
  ClrScr;
end.
```

{ClrEol.PAS}

{Beispielcode für die Prozedur ClrEol.}

```
uses WinCrt;

begin
  Writeln('Hello there, how are you today?');
  Writeln('Press any key...');
  Readln;
  CursorTo(13,0);
  ClrEol;
end.
```

{ScrollTo.PAS}

{Beispielcode für die Prozedur ScrollTo.}

```
uses WinCrt;

begin
  GotoXY(1,10);
  Writeln('Hello');
  Writeln('Type in a line and press Enter.');
  Readln;
  ScrollTo(0,10);
end.
```

{TrackCur.PAS}

{Beispielcode für die Prozedur TrackCursor.}

```
uses WinCrt;

var
  x: integer;

begin
  for x := 1 to 30 do
    Write('Xx');
  TrackCursor;
  Readln;
end.
```

{AssignCr.PAS}

{Beispielcode für die Prozedur AssignCrt.}

```
uses WinCrt;

var
  MyScreenFile: Text;

begin
  AssignCRT(MyScreenFile);
  Rewrite(MyScreenFile);
  Writeln(MyScreenFile,'The file variable MyScreenFile is associated with
the screen.');
  Close(MyScreenFile);
end.
```

{FileSch.PAS}

{Beispielcode für die Funktion FileSearch.}

```
uses WinCrt, WinDos;

var
  S: array[0..fsPathName] of Char;
begin
  FileSearch(S, 'TPW.EXE', GetEnvVar('PATH'));
  if S[0] = #0 then
    Writeln('TPW.EXE not found')
  else
    Writeln('Found as ', FileExpand(S, S));
end.
```

{FileSplt.PAS}

{Beispielcode für die Funktion FileSplit.}

```
uses Strings, WinCrt, WinDos;

var
  Path: array[0..fsPathName] of Char;
  Dir: array[0..fsDirectory] of Char;
  Name: array[0..fsFileName] of Char;
  Ext: array[0..fsExtension] of Char;
begin
  Write('Filename (WORK.PAS): ');
  Readln(Path);
  FileSplit(Path, Dir, Name, Ext);
  if Name[0] = #0 then StrCopy(Name, 'WORK');
  if Ext[0] = #0 then StrCopy(Ext, '.PAS');
  StrECopy(StrECopy(StrECopy(Path, Dir), Name), Ext);
  Writeln('Resulting name is ', Path);
end.
```

{StrCat.PAS}

{Beispielcode für die Funktion StrCat.}

```
uses Strings, WinCrt;

const
  Turbo: PChar = 'Turbo';
  Pascal: PChar = 'Pascal';
var
  S: array[0..15] of Char;
begin
  StrCopy(S, Turbo);
  StrCat(S, ' ');
  StrCat(S, Pascal);
  Writeln(S);
end.
```

{StrComp.PAS}

{Beispielcode für die Funktionen StrComp und StrlComp.}

```
uses Strings, WinCrt;

var
  C: Integer;
  Result: PChar;
  S1, S2: array[0..79] of Char;
begin
  Readln(S1);
  Readln(S2);
  C := StrComp(S1, S2);
  if C < 0 then Result := ' is less than ' else
    if C > 0 then Result := ' is greater than ' else
      Result := ' is equal to ';
  Writeln(S1, Result, S2);
end.
```

{StrCopy.PAS}

{Beispielcode für die Funktion StrCopy.}

```
uses Strings, WinCrt;

var
  S: array[0..15] of Char;
begin
  StrCopy(S, 'Turbo Pascal');
  Writeln(S);
end.
```

{StrECopy.PAS}

{Beispielcode für die Funktion StrECopy.}

```
uses Strings, WinCrt;

const
  Turbo: PChar = 'Turbo';
  Pascal: PChar = 'Pascal';
var
  S: array[0..15] of Char;
begin
  StrECopy(StrECopy(StrECopy(S, Turbo), ' '), Pascal);
  Writeln(S);
end.
```

{StrEnd.PAS}

{Beispielcode für die Funktion StrEnd.}

```
uses Strings, WinCrt;

var
  S: array[0..79] of Char;
begin
  Readln(S);
  Writeln('String length is ', StrEnd(S) - S);
end.
```

{StrLCat.PAS}

{Beispielcode für die Funktion StrLCat.}

```
uses Strings, WinCrt;

var
  S: array[0..9] of Char;
begin
  StrLCopy(S, 'Turbo', SizeOf(S) - 1)
  StrLCat(S, ' ', SizeOf(S) - 1);
  StrLCat(S, 'Pascal', SizeOf(S) - 1);
  Writeln(S);
end.
```

{StrLComp.PAS}

{Beispielcode für die Funktion StrLComp.}

```
uses Strings, WinCrt;

var
  Result: PChar;
  S1, S2: array[0..79] of Char;
begin
  Readln(S1);
  Readln(S2);
  if StrLComp(S1, S2, 5) = 0 then
    Result := 'equal'
  else
    Result := 'different';
  Writeln('The first five characters are ', Test);
end.
```

{StrLCopy.PAS}

{Beispielcode für die Funktion StrLCopy.}

```
uses Strings, WinCrt;

var
  S: array[0..9] of Char;
begin
  StrLCopy(S, 'Turbo Pascal', SizeOf(S) - 1);
  Writeln(S);
end.
```

{StrLen.PAS}

{Beispielcode für die Funktion StrLen.}

```
uses Strings, WinCrt;

var
  S: array[0..79] of Char;
begin
  Readln(S);
  Writeln('String length is ', StrLen(S));
end.
```

{StrLower.PAS}

{Beispielcode für die Funktion StrLower.}

```
uses Strings, WinCrt;

var
  S: array[0..79] of Char;
begin
  Readln(S);
  Writeln(StrLower(S));
  Writeln(StrUpper(S));
end.
```

{StrMove.PAS}

{Beispielcode für die Funktionen StrMove und StrDispose.}

```
{ Allocate string on heap }

function StrNew(S: PChar): PChar;
var
  L: Word;
  P: PChar;
begin
  if (S = nil) or (S^ = #0) then StrNew := nil else
  begin
    L := StrLen(S) + 1;
    GetMem(P, L);
    StrNew := StrMove(P, S, L);
  end;
end;

{ Dispose string on heap }

procedure StrDispose(S: PChar);
begin
  if S <> nil then FreeMem(S, StrLen(S) + 1);
end;
```

{StrNew.PAS}

{Beispielcode für die Funktion StrNew.}

```
uses Strings, WinCrt;

var
  P: PChar;
  S: array[0..79] of Char;
begin
  Readln(S);
  P := StrNew(S);
  Writeln(P);
  StrDispose(P);
end.
```

{StrPas.PAS}

{Beispielcode für die Funktion StrPas.}

```
uses Strings, WinCrt;

var
  A: array[0..79] of Char;
  S: string[79];
begin
  Readln(A);
  S := StrPas(A);
  Writeln(S);
end.
```

{StrPCopy.PAS}

{Beispielcode für die Funktion StrPCopy.}

```
uses Strings, WinCrt;

var
  A: array[0..79] of Char;
  S: string[79];
begin
  Readln(S);
  StrPCopy(A, S);
  Writeln(A);
end.
```

{StrPos.PAS}

{Beispielcode für die Funktion StrPos.}

```
uses Strings, WinCrt;

var
  P: PChar;
  S, SubStr: array[0..79] of Char;
begin
  Readln(S);
  Readln(SubStr);
  P := StrPos(S, SubStr);
  if P = nil then
    Writeln('Substring not found');
  else
    Writeln('Substring found at index ', P - S);
end.
```

{StrRScan.PAS}

{Beispielcode für die Funktion StrRScan.}

```
{ Return pointer to name part of a full path name }

function NamePart(FileName: PChar): PChar;
var
  P: PChar;
begin
  P := StrRScan(FileName, '\');
  if P = nil then
  begin
    P := StrRScan(FileName, ':');
    if P = nil then P := FileName;
  end;
  NamePart := P;
end;
```

{StrScan.PAS}

{Beispielcode für die Funktion StrScan.}

```
Return true if file name has wildcards in it.

function HasWildcards(FileName: PChar): Boolean;
begin
  HasWildcards := (StrScan(FileName, '*') <> nil) or
    (StrScan(FileName, '?') <> nil);
end;
```

{StrUpper.PAS}

{Beispielcode für die Funktion StrUpper.}

```
uses Strings, WinCrt;

var
  S: array[0..79] of Char;
begin
  Readln(S);
  Writeln(StrUpper(S));
  Writeln(StrLower(S));
end.
```

{CursorTo.PAS}

{Beispielcode für die Prozedur CursorTo.}

```
uses WinCrt;

var
  C: PChar;

begin
  CursorTo(15,10);
  Writeln('Hi there.');
end.
```

Beispielprogramme

AssignCr.PAS
Chr.PAS
ClrEol.PAS
ClrScr.PAS
CreateDr.PAS
CSeg.PAS
CursorTo.PAS
Delete.PAS
DoneWCrt.PAS
Eoln.PAS
Exit.PAS
Exp.PAS
FileExp.PAS
FilePos.PAS
FileSch.PAS
FileSplt.PAS
Flush.PAS
FreeMem.PAS
GetArgCo.PAS
GetArgSt.PAS
GetCBrk.PAS
GetCurDr.PAS
GetDate.PAS
GetDir.PAS
GetEnvVa.PAS
GetFTime.PAS
GetIntVc.PAS
GetTime.PAS
GetVerify.PAS
GotoXY.PAS
Halt.PAS
InitWCrt.PAS
Intr.PAS
KeyPress.PAS
Ln.PAS
MsDos.PAS
Odd.PAS
Ord.PAS
Pi.PAS
Pred.PAS
Random.PAS
ReadBuf.PAS
ReadKey.PAS
Readln.PAS
RemoveDr.PAS
Rename.PAS
RmDir.PAS
Round.PAS
ScrollTo.PAS
SeekEof.PAS
SetCurDr.PAS
SetDate.PAS
SetFAttr.PAS

SetTime.PAS
Sqr.PAS
StrCat.PAS
StrComp.PAS
StrCopy.PAS
StrECopy.PAS
StrEnd.PAS
StrLCat.PAS
StrLComp.PAS
StrLCopy.PAS
StrLen.PAS
StrLower.PAS
StrMove.PAS
StrNew.PAS
StrPas.PAS
StrPCopy.PAS
StrPos.PAS
StrRScan.PAS
StrScan.PAS
StrUpper.PAS
TrackCur.PAS
Trunc.PAS
Truncate.PAS
UpCase.PAS
WhereX.PAS
WhereY.PAS
WriteBuf.PAS
WriteChr.PAS

Laufzeit-Fehlermeldungen

Um zum Hilfebildschirm einer bestimmten Fehlernummer zu kommen, wählen Sie die Nummer des Fehlers aus und drücken RETURN.

Fehler#	Fehlermeldung
1	Invalid function number
2	File not found
3	Path not found
4	Too many open files
5	File access denied
6	Invalid file handle
12	Invalid file access code
15	Invalid drive number
16	Cannot remove current directory
17	Cannot rename across drives
100	Disk read error
101	Disk write error
102	File not assigned
103	File not open
104	File not open for input
105	File not open for output
106	Invalid numeric format
200	Division by zero
201	Range check error
202	Stack overflow error
203	Heap overflow error
204	Invalid pointer operation
205	Floating point overflow
206	Floating point underflow
207	Invalid floating point operation
210	Object not initialized
211	Call to abstract method
212	Stream registration error
213	Collection index out of range
214	Collection overflow error

Siehe auch:

Fehlermeldungen des Compilers

Fehlermeldungen des Compilers

Um zum Hilfebildschirm einer bestimmten Fehlernummer zu kommen, wählen Sie die Nummer des Fehlers aus und drücken RETURN.

Fehler#	Fehlermeldung
1	Out of memory
2	Identifier expected
3	Unknown identifier
4	Duplicate identifier
5	Syntax error
6	Error in real constant
7	Error in integer constant
8	String constant exceeds line
9	Too many nested files
10	Unexpected end of file
11	Line too long
12	Type identifier expected
13	Too many open files
14	Invalid file name
15	File not found
16	Disk full
17	Invalid compiler directive
18	Too many files
19	Undefined type in pointer definition
20	Variable identifier expected
21	Error in type
22	Structure too large
23	Set base type out of range
24	File components may not be files or objects
25	Invalid string length
26	Type mismatch
27	Invalid subrange base type
28	Lower bound > than upper bound
29	Ordinal type expected
30	Integer constant expected
31	Constant expected
32	Integer or real constant expected
33	Pointer type identifier expected
34	Invalid function result type
35	Label identifier expected
36	BEGIN expected
37	END expected
38	Integer expression expected
39	Ordinal expression expected
40	Boolean expression expected
41	Operand types do not match operator
42	Error in expression
43	Illegal assignment
44	Field identifier expected
45	Object file too large
46	Undefined External
47	Invalid object file record
48	Code segment too large
49	Data segment too large

50	DO expected
51	Invalid PUBLIC definition
52	Invalid EXTRN definition
53	Too many EXTRN definitions
54	OF expected
55	INTERFACE expected
56	Invalid relocatable reference
57	THEN expected
58	TO or DOWNTO expected
59	Undefined forward
61	Invalid typecast
62	Division by zero
63	Invalid file type
64	Cannot Read or Write variables of this type
65	Pointer variable expected
66	String variable expected
67	String expression expected
68	Circular unit reference
69	Unit name mismatch
70	Unit version mismatch
72	Unit file format error
73	IMPLEMENTATION expected
74	Constant and case types don't match
75	Record variable expected
76	Constant out of range
77	File variable expected
78	Pointer expression expected
79	Integer or real expression expected
80	Label not within current block
81	Label already defined
82	Undefined label in preceding statement part
83	Invalid @ argument
84	UNIT expected
85	";" expected
86	":" expected
87	"," expected
88	"(" expected
89	")" expected
90	"=" expected
91	":=" expected
92	"[" or "(." expected
93	"]" or ".)" expected
94	"." expected
95	".." expected
96	Too many variables
97	Invalid FOR control variable
98	Integer variable expected
99	File and procedure types are not allowed here
100	String length mismatch
101	Invalid ordering of fields
102	String constant expected
103	Integer or real variable expected
104	Ordinal variable expected
105	INLINE error
106	Character expression expected

112	CASE constant out of range
113	Error in statement
114	Cannot call an interrupt procedure
116	Must be in 8087 mode to compile
117	Target address not found
118	Include files are not allowed here
120	NIL expected
121	Invalid qualifier
122	Invalid variable reference
123	Too many symbols
124	Statement part too large
126	Files must be var parameters
127	Too many conditional symbols
128	Misplaced conditional directive
129	ENDIF directive missing
130	Error in initial conditional defines
131	Header does not match previous definition
132	Critical disk error
133	Cannot evaluate this expression
136	Invalid indirect reference
137	Structured variables are not allowed here
140	Invalid floating-point operation
142	Procedure or function variable expected
143	Invalid procedure or function reference
146	File access denied
147	Object type expected
148	Local object types are not allowed
149	VIRTUAL expected
150	Method identifier expected
151	Virtual constructors are not allowed
152	Constructor identifier expected
153	Destructor identifier expected
154	Fail only allowed within constructors
155	Invalid combination of opcode and operands
156	Memory reference expected
157	Cannot add or subtract relocatable symbols
158	Invalid register combination
159	286/287 instructions are not enabled
160	Invalid symbol reference
161	Code generation error
162	ASM expected
163	Duplicate dynamic method index
164	Duplicate resource identifier
165	Duplicate or invalid export index
166	Procedure or function identifier expected
167	Cannot export this symbol
168	Duplicate export name

Siehe auch:

Laufzeitfehlermeldungen

Laufzeitfehler 1: Invalid function number

(ungültige Funktion)
Diese Fehlermeldung wird ausgelöst, wenn versucht wird, eine nicht existierende Dos-Funktion aufzurufen.

Laufzeitfehler 2: File not found

(Datei nicht gefunden)

Ausgelöst von **Reset**, **Append**, **Rename** und **Erase**, wenn der der Dateivariablen zugeordnete Name keine exisiterende Datei bezeichnet bzw. diese Datei im aktuellen Verzeichnis nicht gefunden wurde.

Laufzeitfehler 3: Path not found

(Suchweg nicht gefunden)
Ausgelöst von **Reset**, **Rewrite**, **Append**, **Rename** und **Erase**, wenn der der Dateivariablen zugeordnete Name einen ungültigen Suchweg oder ein nicht existierendes Verzeichnis bezeichnet.

Ausgelöst von **ChDir**, **MkDir** oder **RmDir**, wenn der angegebene Suchweg ungültig ist oder ein nicht existierendes Verzeichnis bezeichnet.

Laufzeitfehler 4: Too many open files

(Maximalzahl an Dateien bereits offen)
Ausgelöst von **Reset**, **Rewrite** und **Append**, wenn das Betriebssystem die Öffnung einer weiteren Datei aufgrund fehlender Dateipuffer verweigert.

Maximal sind 15 offene Dateien pro Prozeß erlaubt.

Erhalten Sie diesen Fehler bei weniger als 15 offenen Dateien, sollten Sie den Parameter der Datei CONFIG.SYS überprüfen (FILES=xx) und ihn gegebenfalls erhöhen, z.B. FILES = 20.

Danach ist ein Neustart des Computers nötig, um Änderungen wirksam werden zu lassen.

Laufzeitfehler 5: File access denied

(Dateizugriff verweigert)

Ausgelöst von	wann tritt dieses Problem auf?
Reset	Wenn FileMode Schreib- und Lesezugriffe erlaubt, die Datei-Variable aber einer gegen Überschreiben gesperrten Datei oder einem (Unter-)Verzeichnis zugeordnet ist.
Append	FileMode erlaubt Schreibzugriffe, die Datei-Variable gibt ein Verzeichnis oder nur Lesezugriff an.
Rewrite	Wenn das Verzeichnis keine weiteren Dateieinträge mehr erlaubt oder die Datei gegen Überschreiben geschützt ist.
Rename	Wenn der Name ein (Unter-)Verzeichnis bezeichnet oder bereits eine Datei mit diesem Namen existiert.
Erase	Wenn die Datei gegen Schreibzugriffe geschützt ist oder ein (Unter-)Verzeichnis bezeichnet.
MkDir	Wenn bereits eine normale Datei mit diesem Namen in diesem Verzeichnis existiert, das Verzeichnis keinen Platz mehr bietet oder der Pfad ein Peripheriegerät bezeichnet.
RmDir	Wenn das Verzeichnis nicht leer ist, es sich beim angegebenen Suchweg um eine normale Datei oder um das Root-Verzeichnis des Datenträgers handelt.
Read	(bei einer typisierten oder nicht-typisierten Datei): Wenn die Datei nicht zum Lesen geöffnet wurde.
BlockRead	(bei einer typisierten oder nicht-typisierten Datei): Wenn die Datei nicht zum Lesen geöffnet wurde.
Write	(bei einer typisierten oder nicht-typisierten Datei): Wenn die Datei nicht zum Schreiben geöffnet wurde.
BlockWrite	(bei einer typisierten oder nicht-typisierten Datei): Wenn die Datei nicht zum Schreiben geöffnet wurde

Laufzeitfehler 6: Invalid file handle

(Handle nicht definiert / ungültig)
Dieser Fehler sollte nie aufteten; falls doch, dann ist die Dateivariable aus irgendeinem Grund überschrieben oder zerstört worden.

Laufzeitfehler 12: Invalid file access code

(ungültiger Dateimodus)
Ausgelöst von **Reset** und **Append** bei der Anwendung auf typisierte und nicht-typisierte Dateien, wenn der angegebene Zugriff nicht möglich ist, weil das Feld FileMode einen ungültigen Wert hat.

Laufzeitfehler 15: Invalid drive number

(Laufwerksnummer unzulässig)
GetDir und **ChDir** melden diesen Fehler, wenn die angegebene Laufwerksnummer außerhalb der zulässigen Grenzen liegt.

Laufzeitfehler 16: Cannot remove current directory

(als Standard gesetztes Verzeichnis kann nicht gelöscht werden)
RmDir meldet diesen Fehler beim Versuch, das aktuelle Verzeichnis zu löschen.

Laufzeitfehler 17: Cannot rename across drives

(Rename kann nicht kopieren)
Rename meldet diesen Fehler, wenn für das Umbennen einer Datei zwei verschiedene Laufwerke angegeben wurden.

Laufzeitfehler 100: Disk read error

(Fehler beim Lesen von der Diskette)
Read löst diesen Fehler aus, wenn versucht wird, über das Ende einer typisierten Datei hinaus weitere Daten zu lesen.

Laufzeitfehler 101: Disk write error

(Fehler beim Schreiben auf die Diskette)
Close, **Write**, **Writeln** und **Flush** melden diesen Fehler, wenn die Zieldiskette voll ist.

Laufzeitfehler 102: File not assigned

(Dateivariable ist keiner Datei zugeordnet)
Reset, **Rewrite**, **Append**, **Rename** und **Erase** melden diesen Fehler, wenn die Dateivariable noch nicht durch den Aufruf von Assign einer Datei zugeordnet wurde.

Laufzeitfehler 103: File not open

(Datei ist nicht offen)
Dieser Fehler wird ausgelöst, wenn die folgenden Routinen bei der Anwendung auf eine Datei feststellen, daß diese noch nicht geöffnet wurde:

BlockRead
BlockWrite
Close
Eof
FilePos
FileSize
Flush
Read
Seek
Write

Laufzeitfehler 104: File not open for input

(Datei wurde nicht für Leseoperationen geöffnet)
Ausgelöst von den folgenden Routinen beim Anwenden auf eine Textdatei, die nur für Schreiboperationen geöffnet wurde:

Eof
Eoln
Read
ReadLn
SeekEof
SeekEoln

Laufzeitfehler105: File not open for output

(Datei wurde nicht für Schreibeoperationen geöffnet)
Write und **WriteLn** melden diesen Fehler, wenn die Textdatei nur zum Lesen geöffnet wurde.

Laufzeitfehler 106: Invalid numeric format

(ungültiges numerisches Format)
Read und **ReadLn** melden diesen Fehler, wenn die von einer Textdatei gelesenen Werte nicht dem numerischen Format der angegebenen Variablen entsprechen.

Laufzeitfehler 200: Division by zero

(Division durch Null)
Versuch, eine Zahl mit den Operatoren **/**, **mod** oder **div** durch Null zu teilen.

Siehe auch

Binäre arithmetische Operatoren (/, mod und div)

Laufzeitfehler 201: Range check error

(Bereichsüberprüfung: Fehler entdeckt)
Diesen Fehler erhalten Sie, wenn das Programm im Modus `{$R+}` compiliert wurde und eine der folgenden Bedingungen zutrifft:

- Der Wert des Index beim Zugriff auf ein Array überschreitet die Array-Grenzen.

- Das Programm hat versucht, einer Variablen einen Wert außerhalb ihres Wertebereichs zuzuweisen.

- Das Programm hat versucht, einer Prozedur oder Funktion einen Wert zu übergeben, der außerhalb des zulässigen Wertebereichs des Parameters dieser Routine liegt.

Laufzeitfehler 202: Stack overflow error

(Stack-Überprüfung: Überlauf entdeckt)
Diesen Fehler erhalten Sie, wenn das Programm im Modus **{$S+}** compiliert wurde und sich beim Aufruf der Prozedur herausstellt, daß nicht mehr genügend Platz auf dem Stack vorhanden ist, um die lokalen Variablen des Unterprogramms abzulegen.

Verwenden Sie den Compiler-Befehl **$M**, um die Größe des Stacks zu erhöhen.

Ein Stacküberlauf kann auch durch eine nicht terminierende Rekursion oder durch ein Assembler- Programm, das den Stack nicht korrekt verwaltet, ausgelöst werden.

Laufzeitfehler 203: Heap overflow error

(kein Platz mehr im Heap-Bereich)
New und **GetMem** melden diesen Fehler, wenn nicht mehr genügend Platz auf dem Heap vorhanden ist, um einen Speicherbereich der gewünschten Größe zu belegen.

Eine eingehende Beschreibung des Heap-Managers finden Sie in Kapitel 16 "Speicherbelegung und Datenformate" im Programmierhandbuch.

Laufzeitfehler 204: Invalid pointer operation

(ungültige Zeiger-Operation)
Dispose und **FreeMem** melden diesen Fehler, wenn die übergebene Zeigervariable den Wert NIL hat oder auf eine Adresse außerhalb des Heap zeigt.

Laufzeitfehler 205: Floating point overflow

(Gleitkomma-Überlauf)
Eine Gleitkomma-Operation hatte ein Ergebnis, das den zulässigen Wertebereich (bei Verwendung eines numerischen Coprozessors, falls vorhanden) überschritten hat.

Laufzeitfehler 206: Floating point underflow

(Gleitkomma-Unterlauf)
Unterläufe liefern normalerweise den Wert 0 - diesen Fehler erhalten Sie nur bei der Verwendung des numerischen Coprozessors und der Freigabe des Bits für Unterläufe im Statusregister des 80x87.

Laufzeitfehler 207: Invalid floating point operation

(Gleitkomma-Fehler)
Einer der folgenden Fehler ist aufgetreten:

- Die an **Trunc** oder **Round** übergebene Real-Zahl konnte nicht innerhalb des Wertebereichs von Longint (-2147483648 bis 2147483647) in einen **Integer-Wert** konvertiert werden.

- Der Funktion **Sqrt** wurde ein negatives Argument übergeben.

- Der Funktion **Ln** wurde ein negatives Argument oder ein Argument gleich 0 übergeben.

- Der Stack des 80x87 ist übergelaufen.

Laufzeitfehler 210: Object not initialized

(Objekt nicht initialisiert)
Sie haben versucht, die virtuelle Methode eines Objekts aufzurufen, obwohl dieses Objekt noch nicht (mittels eines Konstruktor-Aufrufs) initialisiert ist.

Laufzeitfehler 211: Call to abstract method

(Aufruf einer abstrakten Methode)
Dieser Fehler wird von der Prozedur **Abstract** in der Unit **WObjects** ausgelöst. Sie haben versucht, in Ihrem Programm eine abstrakte virtuelle Methode auszuführen.

Laufzeitfehler 212: Stream registration error

(Fehler bei Registrierung eines Objekts)
Dieser Fehler wird von der Prozedur **RegisterType** in der Unit **WObjects** ausgelöst.

Folgende Gründe führen zu diesem Fehler:

- Der Registrierungsrecord des Stream ist nicht im Datensegment.
- Das Datenfeld ObjType des Registrierungsrecords enthält den Wert 0.
- Dieser Typ wurde bereits registriert.
- Ein anderer Typ mit demselben Wert von ObjType existiert bereits.

Laufzeitfehler 213: Collection Index out of Range

(Index außerhalb des zulässigen Bereichs)
Der Index, der beim Aufruf einer Methode von **TCollection** übergeben wurde, liegt außerhalb des zulässigen Bereichs.

Laufzeitfehler 214: Collection overflow error

(Überlauf einer Kollektion)
Dieser Fehler wird von **TCollection** ausgelöst, wenn Sie versuchen, der Kollektion ein weiteres Objekt anzufügen, obwohl diese Kollektion nicht mehr erweitert werden kann.

Compiler-Fehler 1: Out of memory

(Platz im Hauptspeicher reicht nicht aus)
Dieser Fehler tritt auf, wenn der Platz im Hauptspeicher nicht ausreicht, um Ihr Programm zu übersetzen. Hier sind einige mögliche Lösungen:

- Versuchen Sie, den verügbaren Speicherplatz in Windows zu vergrößern.

- Setzen Sie die Einstellung Options|Linker|**Link buffer** auf Disk.

- Verwenden Sie die Option **/L**, damit der Kommandozeilen-Compiler eine temporäre Diskettendatei als Linkerpuffer verwendet.

Hilft keiner dieser Vorschläge, ist Ihr Programm möglicherweise zu groß, um im vorhandenen Speicher compiliert zu werden. Sie müssen es in kleinere **Units** zerlegen.

Compiler-Fehler 2: Identifier expected

(Bezeichner erwartet)
Der Compiler erwartet an dieser Stelle einen Bezeichner.

Diesen Fehler erhalten Sie auch beim Versuch, ein **reserviertes** Wort zu redeklarieren.

Compiler-Fehler 3: Unknown identifier

(unbekannter Bezeichner)
Der angegebene Bezeichner ist entweder nicht deklariert oder es kann auf ihn nicht zugegriffen werden, weil er sich außerhalb des aktuellen Gültigkeitsbereiches (Scope) befindet.

Compiler-Fehler 4: Duplicate identifier

(doppelter Bezeichner)
Der angegebene Bezeichner ist innerhalb des aktuellen Blocks bereits vergeben.

Compiler-Fehler 5: Syntax error

(Syntaxfehler)
Der Compiler hat ein unzulässiges Zeichen im Quelltext entdeckt.

Vielleicht haben Sie vergessen, einen String in Hochkommata einzuschließen.

Compiler-Fehler 6: Error in real constant

(Fehlerhafte Ausgabe einer Real-Konstanten)
Die Syntax von Real-Konstanten ist in Kapitel 1 "Sprachelemente und Konstanten" des Programmierhandbuches näher beschrieben.

Compiler-Fehler 7: Error in Integer constant

(Fehlerhafte Ausgabe einer Integer-Konstanten)
Die Syntax von Integer-Konstanten ist im Kapitel Sprachelemente und Konstanten des Windows-Programmierhandbuches näher beschrieben.

Hinweis: Ganzzahlige Werte müssen mit einem Dezimalpunkt und einer folgenden Null ausgegeben werden, wenn sie außerhalb des Wertebereichs für Integertypen liegen, z.B.

12345678912.0

Compiler-Fehler 8: String constant exceeds line

(String-Konstante geht über das Zeilenende hinaus)
Wahrscheinlich haben Sie ein abschließendes Hochkomma vergessen.

Compiler-Fehler 9: Too many nested files

(Include-Verschachtelung zu tief)
Verschachtelungen über Include-Dateien sind maximal bis zu einer Tiefe von 15 Ebenen erlaubt.

Compiler-Fehler 10: Unexpected end of file

(unerwartetes Quelltext-Ende)
Für diese Fehlermeldung gibt es folgende Gründe:

- Ihre Quelltext-Datei endet vor dem abschließenden **end** des Hauptprogramms. Es könnte sein, daß der Quelltext eine unausgeglichene Anzahl von **begin** und **end** enthält.

- Eine Include-Datei endet inmitten eines Anweisungsteils. Die Aufteilung eines Prozedur- oder Funktionsblocks auf mehrere Dateien ist nicht möglich.

- Sie haben vergessen, einen Kommentar abzuschließen.

Compiler-Fehler 11: Line too long

(Eine Programmzeile hat mehr als 126 Zeichen.)

Compiler-Fehler 12: Type identifier expected

(Typ-Bezeichner erwartet)
Der Compiler erwartet an dieser Stelle einen Typ-Bezeichner.

Compiler-Fehler 13: Too many open files

(zuviele Dateien geöffnet)
Wenn dieser Fehler auftritt, enthält entweder Ihre CONFIG.SYS Datei keine Parameterangabe für FILES=xx oder die Zahl xx ist zu niedrig gewählt.

Erhöhen Sie die Zahl z.B. auf 20.

Compiler-Fehler 14: Invalid file name

(ungültiger Dateiname)
Der in einem Compiler-Befehl angegebene Dateiname und/oder Suchweg ist ungültig bzw. der Suchweg existiert nicht.

Compiler-Fehler 15: File not found

(Datei nicht gefunden)
Die in einem Compiler-Befehl angegebene Datei läßt sich weder im momentan gesetzten Verzeichnis noch über die Suchwege für den entsprechenden Dateityp finden.

Compiler-Fehler 16: Disk full

(Diskette / Festplatte voll)
Löschen Sie nicht mehr benötigte Dateien.

Compiler-Fehler 17: Invalid compiler directive

(Compiler-Befehl ungültig)
Eine der folgenden Möglichkeiten trifft zu:

- Der Compiler-Befehl existiert nicht.

- Einer der Parameter für den Compiler-Befehl ist ungültig.

- Sie verwenden einen Compiler-Befehl im Rumpf des Programmes, der globale Wirkung hat und deshalb nur vor der ersten Prozedur- oder Funktionsdeklaration angegeben werden darf.

Compiler-Fehler 18: Too many files

(zu viele Dateien inerhalb eines Projekts)
Die interne Dateinamen-Tabelle des Compilers ist voll. Nehmen Sie Include-Dateien in Ihre Quelltexte mit auf, fassen Sie sämtliche Dateien in einem Verzeichnis zusammen, und verwenden Sie kürzere Dateinamen.

Compiler-Fehler 19: Undefined type in pointer definition

(Zieltyp der Zeigerdeklaration nicht definiert)
Der bei der Definition eines Zeigers angegebene Datentyp ist dem Compiler nicht bekannt.

Compiler-Fehler 20: Variable identifier expected

(Variablen-Bezeichner erwartet)
Der Bezeichner bezeichnet keine Variable.

Compiler-Fehler 21: Error in type

(Unzulässiges Symbol/Zeichen in einer Typ-Definition)

Compiler-Fehler 22: Structure too large

(Datenstruktur ist größer als 65520 Bytes)

Compiler-Fehler 23: Set base type out of range

(Mengen-Grundtyp außerhalb des möglichen Wertebereichs)
Der Basistyp einer Menge muß ein Teilbereich sein, der zwischen 0 und 255 liegt, oder ein Aufzählungstyp mit maximal 256 Werten.

Compiler-Fehler 24: File components may not be files or objects

(Dateikomponenten dürfen kein Dateien oder Objekte sein)
Als zusammengesetzten Datentyp einer Datei darf kein Objekt- oder Dateityp angeben werden. Es darf auch kein strukturierter Datentyp mit einer Objekt- oder Dateikomponente angegeben werden.

Compiler-Fehler 25: Invalid string length

(String außerhalb des Bereichs 1...255)

Compiler-Fehler 26:Type mismatch

(Typen nicht miteinander vereinbar)
Dieser Fehler kann eine der folgenden Ursachen haben:

- Der Typ der Variablen und der Ergebnistyp einer Zuweisung sind nicht kompatibel.
- Formaler und aktueller Parameter beim Aufruf einer Prozedur oder Funktion sind nicht kompatibel.
- Der Ergebnistyp des Ausdrucks und der Indextyp des Arrays sind nicht kompatibel.
- Die Kombination von Operanden und Operatoren sind nicht miteinander verträglich.

Compiler-Fehler 27: Invalid subrange base type

(Der Basistyp des Unterbereichs ist nicht ordinal)

Compiler-Fehler 28: Lower bound > than upper bound

(Untergrenze eines Teilbereichs > Obergrenze)

Compiler-Fehler 29: Ordinal type expected

(Ordinaltyp erwartet)
An dieser Stelle sind keine Real-Typen, Strings, strukturierte Typen oder Zeiger erlaubt.

Compiler-Fehler 30: Integer constant expected

(Integer-Konstante erwartet)

Compiler-Fehler 31: Constant expected

(Konstante erwartet)

Compiler-Fehler 32: Integer or real constant expected

(Integer- oder Real-Konstante erwartet)

Compiler-Fehler 33: Pointer Type identifier expected

(Zeiger-Typ-Bezeichner erwartet)
An dieser Stelle erwartet der Compiler eine Variable vom Typ Zeiger.

Compiler-Fehler 34: Invalid function result type

(Ergebnistyp der Funktion nicht zulässig)
Gültige Ergebnistypen von Funktionen sind alle einfachen Typen, Strings und Zeiger.

Compiler-Fehler 35: Label identifier expected

(Label-Bezeichner erwartet.)

Compiler-Fehler 36: BEGIN expected

(begin erwartet)
An dieser Stelle wird ein **BEGIN** erwartet oder in der Blockstruktur Ihres Programmes stimmt etwas nicht.

Compiler-Fehler 37: END expected

(end erwartet)
An dieser Stelle wird ein **END** erwartet oder aber in der Blockstruktur Ihres Programmes oder Ihrer Unit stimmt etwas nicht.

Compiler-Fehler 38: Integer expression expected

(Ergebnistyp des Ausdrucks muß Integer sein)

Compiler-Fehler 39: Ordinal expression expected

(Der Ergebnistyp des Ausdrucks muß ordinal sein)

Compiler-Fehler 40: Boolean expression expected

(Der Ergebnistyp des Ausdrucks muß Boolean sein)

Compiler-Fehler 41: Operand types do not match operator

(Operandentypen passen nicht zum Operator)
Der Operator kann nicht auf Operanden dieses Typs angewendet werden, z.B. 'A' div '2'.

Compiler-Fehler 42: Error in expression

(Fehler innerhalb des Ausdrucks)
Der Compiler hat zwar erkannt, daß es sich um einen Ausdruck handelt, kann diesen aber nicht auswerten (z.B. wegen eines fehlenden Operators).

Compiler-Fehler 43: Illegal assignment

(Zuweisung nicht erlaubt)
Die wahrscheinlichsten Ursachen für diese Fehlermeldung sind:

Der Versuch, einer Dateivariablen oder einer nicht-typisierten Variablen direkt einen Wert zuzuweisen oder
der Versuch, einer Funktion einen Wert außerhalb ihres Anweisungsteils zuzuordnen.

Compiler-Fehler 44: Field identifier expected

(Feld-Bezeichner erwartet)
Der angegebene Bezeichner steht nicht für ein Feld der angegebenen Record-Variablen.

Compiler-Fehler 45: Object file too large

(Linker: .Obj-Datei ist größer als 64 KByte)
Turbo Pascal für Windows kann keine OBJ-Dateien linken, die größer als 64 KByte sind.

Compiler-Fehler 46: Undefined External

(Linker: external-Bezeichner ist nirgendwo definiert)
Bei einer **external**-Prozedur oder -Funktion fehlt die passende PUBLIC-Definition in der Objektdatei.

Prüfen Sie, ob auch alle Objekt-Dateien über den Compilerbefehl **$L Dateiname** angegeben wurden, und prüfen Sie Ihre ASM-Dateien auf eventuelle Tippfehler.

Compiler-Fehler 47: Invalid object file record

(Linker:.OBJ-Dateiformat unleserlich)
Der Linker kann eine OBJ-Datei nicht fehlerfrei einbinden.

Compiler-Fehler 48: Code segment too large

(Codesegment-Grenzen überrschritten)
Die Übersetzung eines Programms oder einer Unit erzeugt mehr als 65520 Bytes Code.

- Wenn Sie ein Programm übersetzen, verteilen Sie einige Prozeduren oder Funktionen auf eine oder mehrere Units.

- Wenn Sie eine Unit übersetzen, teilen Sie sie auf.

Compiler-Fehler 49: Data segment too large

(Datensegment-Grenzen überschritten)
Ihr Programm und sämtliche aufgenommenen Units deklarieren zusammen mehr als 65520 Bytes Daten.

Wenn Sie Platz für mehr Daten brauchen, sollten Sie die größeren Datenstrukturen über Zeiger deklarieren und sie über Aufrufe von **NewNew** dynamisch erzeugen.

Compiler-Fehler 50: DO expected

(do erwartet)
An dieser Stelle sollte das reservierte Wort **DO** stehen.

Compiler-Fehler 51: Invalid PUBLIC definition

(Linker: PUBLIC-Deklaration inkorrekt)
Gründe für diese Fehlermeldung sind:

- Ein und derselbe Bezeichner wird in mehreren Objekt-Dateien als **PUBLIC** definiert.

- Ein als **PUBLIC** definiertes Symbol befindet sich außerhalb des **Code**-Segments einer OBJ-Datei.

Compiler-Fehler 52: Invalid EXTRN definition

(Linker: EXTRN-Definition nicht korrekt)
Mögliche Gründe für diese Fehlermeldung sind:

- Ein Bezeichner ist innerhalb des Assembler-Programms als **EXTRN** definiert, ist aber in keiner Pascal-Datei als verfügbar definiert.

- Der Bezeichner stellt eine **absolute** Variable dar.

- Der Bezeichner bezieht sich auf eine **inline** Prozedur oder Funktion.

Compiler-Fehler 53: Too many EXTRN definitions

(zu viele EXTRN-Definitionen)
Maximal sind 256 **EXTRN** Definitionen pro OBJ-Datei erlaubt.

Compiler-Fehler 54: OF expected

(OF erwartet)

Compiler-Fehler 55: INTERFACE expected

(INTERFACE erwartet)

Compiler-Fehler 56: Invalid relocatable reference

(Zugriff auf relozierbares Symbol inkorrekt)
Mögliche Gründe für diesen Fehler sind:

- Die OBJ-Datei definiert Daten und andere relozierbare Objekte außerhalb des Segments **CODE** - beispielsweise initialisierte Variablen im **DATA**-Segment.

- Die OBJ-Datei enthält Zugriffe auf einzelne Bytes relozierbarer Symbole - beispielsweise die Operatoren **HIGH** und **LOW** oder DB-Direktiven.

- Ein Operand bezieht sich auf ein relozierbares Symbol, das weder im Segment **CODE** noch in **DATA** definiert ist.

- Ein Operand bezieht sich auf eine als **EXTRN** deklarierte Prozedur oder Funktion über eine relative Adressangabe, z.B. `CALL SortProc+8`

Compiler-Fehler 57: THEN expected

(THEN erwartet.)

Compiler-Fehler 58: TO or DOWNTO expected

(TO oder DOWNTO erwartet.)

Compiler-Fehler 59: Undefined forward

(forward-Deklaration: Definition fehlt)
Diese Fehlermeldung kann zwei Gründe haben:

- Die Prozedur oder Funktion wurde zwar im **Interface-Teil** deklariert, aber im **Implementationsteil** nicht definiert.

- Eine Prozedur oder Funktion wurde als **forward** deklariert, ihre Definition kann aber nicht gefunden werden.

Compiler-Fehler 61: Invalid typecast

(Typ-Umwandlung nicht möglich)
Diese Fehlermeldung kann folgende Ursachen haben:

- Bei der Umwandlung einer Variablen stimmen die Größen von Original- und Zieltyp nicht überein.

- Sie versuchen eine Typ-Umwandlung an einer Stelle, an der nur Variablen-Zugriffe stehen können.

Compiler-Fehler 62: Division by zero

(Division durch Null)
Der Compiler berechnet konstante Ausdrücke bereits während der Übersetzung und hat dabei offensichtlich eine Division durch Null vornehmen müssen.

Compiler-Fehler 63: Invalid file type

(Operation mit diesem Dateityp nicht erlaubt)
Die Operation kann auf eine Dateivariable des angegebenen Typs nicht angewendet werden (wie z.B. ein Seek auf eine Textdatei oder ein ReadLn auf eine typisierte Datei).

Compiler-Fehler 64: Cannot Read or Write variables of this type

(Variablen dieses Typs können nicht gelesen/geschrieben werden)

Beim Lesen:
Read und **ReadLn** können nur folgende Variablen einlesen:

Char
integer
real
string

Beim Schreiben:
Write und **WriteLn** können nur diese Variablen ausgeben:

Char
Iiteger
real
string
Boolean

Compiler-Fehler 65: Pointer variable expected

(Zeiger-Variable erwartet)

Compiler-Fehler 66: String variable expected

(String-Variable erwartet)

Compiler-Fehler 67: String expression expected

(String-Ausdruck erwartet)

Compiler-Fehler 68: Circular unit reference

(überkreuzender Bezug zweier Units)

Sie haben zwei Units definiert, die sich gegenseitig im Interface-Teil voraussetzen.

Compiler-Fehler 69: Unit name mismatch

(Unit-Name und Unit-Dateiname stimmen nicht überein)
Die Unit hat einen anderen Namen (der hinter unit angegebene Bezeichner) als die TPU-Datei.

Compiler-Fehler 70: Unit version mismatch

(Unit-Versionen stimmen nicht überein)
Eine oder mehrere Units, die diese Unit importiert, sind seit der letzten Compilierung verändert worden.

Verwenden Sie Compile|**Make** oder Compile|**Build**, um diese Units wieder auf den aktuellen Stand zu bringen.

Compiler-Fehler 72: Unit file format error

(LInker: Unit-Dateiformat ungültig)
Der Linker kann den Inhalt der angegebenen Datei nicht als Unit interpretieren. Stellen Sie sicher, daß diese Unit auch mit der aktuellen Version von Turbo Pascal für Windows übersetzt wurde.

Compiler-Fehler 73: IMPLEMENTATION expected

(IMPLEMENTATION erwartet.)

Compiler-Fehler 74: Constant and case types don't match

(Case-Konstante und Selektor haben verschiedene Datentypen.)

Compiler-Fehler 75: Record variable expected

(Record-Variable erwartet.)

Compiler-Fehler 76: Constant out of range

(Konstante außerhalb des zulässigen Wertebereichs)
Hierfür kann es mehrere Gründe geben:

- Indizierung eines Arrays mit einer Konstanten, deren Wert außerhalb der Arraygrenzen liegt.

- Zuweisung eines (konstanten) Wertes an eine Variable, der außerhalb des Wertebereichs der Variablen liegt.

- Übergabe eines (konstanten) Parameters an eine Prozedur oder Funktion, dessen Wert außerhalb des Wertebereichs des entsprechenden formalen Parameters liegt.

Compiler-Fehler 77: File variable expected

(Datei-Vvariable erwartet)

Compiler-Fehler 78: Pointer expression expected

(Zeiger-Ausdruck erwartet)

Compiler-Fehler 79: Integer or real expression expected

(Integer- oder Real-Ausdruck erwartet.)

Compiler-Fehler 80: Label not within current block

(goto-Sprungziel muß innerhalb des momentanen Blocks liegen)
Das Sprungziel einer goto-Anweisung muß innerhalb des aktuellen Blocks liegen.

Compiler-Fehler 81: Label already defined

(Label wurde bereits zur Markierung einer Anweisung verwendet)

Compiler-Fehler 82: Undefined label in preceding statement part

(vorhergehender Anweisungsteil läßt Label undefiniert)

Compiler-Fehler 83: Invalid @ argument

(unzulässiges Argument für @)
Als Argument für den Adreß-Operator @ können nur Variablenbezüge sowie Prozedur- und Funktions- Bezeichner verwendet werden.

Compiler-Fehler 84: UNIT expected

(UNIT erwartet.)

Compiler-Fehler 85: ";" expected

(";" erwartet)

Compiler-Fehler 86: ":" expected

(":" erwartet)

Compiler-Fehler 87: "," expected

("," erwartet)

Compiler-Fehler 88: "(" expected

("(" erwartet)

Compiler-Fehler 89: ")" expected

(")" erwartet)

Compiler-Fehler 90: "=" expected

("=" erwartet)

Compiler-Fehler 91: ":=" expected

(Zuweisungsoperator ":="erwartet)

Compiler-Fehler 92: "[" or "(." expected

("[" oder "(."erwartet)

Compiler-Fehler 93: "]" or ".)" expected

("]" oder ".)" erwartet)

Compiler-Fehler 94: "." expected

("." erwartet)

Compiler-Fehler 95: ".." expected

(".." erwartet)

Compiler-Fehler 96: Too many variables

(zuviele Variablen)

Global

Die Größe der globalen Variablen eines Programmes und darin aufgenommener Units darf 64 KByte nicht überschreiten.

Lokal

Innerhalb einer Prozedur oder Funktion dürfen nicht mehr als 64 KByte lokale Variablen deklariert sein.

Compiler-Fehler 97: Invalid FOR control variable

(Variable nicht als Laufvariable einer for-Schleife verwendbar)
Als Laufvariable einer **FOR**-Schleife dürfen nur einfache ordinale Typen verwendet werden, die entweder global zum Programm oder lokal zum momentanen Block deklariert sein müssen.

Compiler-Fehler 98: Integer variable expected

(Integer-Variable erwartet)

Compiler-Fehler 99: File and procedure types are not allowed here

(Datei- und Prozedurtypen sind hier nicht erlaubt)

Compiler-Fehler 100: String length mismatch

(Stringlänge und Arraygröße stimmen nicht überein.)

Compiler-Fehler 101: Invalid ordering of fields

(Reihenfolge der Felder stimmt nicht überein)
Die Reihenfolge der Felder muß bei einer Record-Konstanten dieselbe sein wie bei der Definition des Records.

Compiler-Fehler 102: String constant expected

(String-Konstante erwartet)

Compiler-Fehler 103: Integer or real variable expected

(Integer- oder Real-Variable erwartet)

Compiler-Fehler 104: Ordinal variable expected

(Variable ordinalen Typs erwartet)

Compiler-Fehler 105: INLINE error

(INLINE-Operator nicht auf Variablen anwendbar)
Der **< Operator** ist nicht auf Variablen anwendbar. Diese Referenzen haben immer Word-Größe.

Compiler-Fehler 106: Character expression expected

(Ausdruck des Ergebnistyps Char erwartet)

Compiler-Fehler 112: CASE constant out of range

(case-Konstante außerhalb des zulässigen Bereichs)
Die Werte der Konstante in der **case**-Anweisung müssen im Bereich -32768..32767 liegen, solange mit einem Selektor vom Typ Integer gearbeitet wird.

Compiler-Fehler 113: Error in statement

(Syntaxfehler in Befehl/Anweisung)
Mit dem von Ihnen angegebenen Symbol darf eine Anweisung nicht beginnen

Compiler-Fehler 114: Cannot call an interrupt procedure

(direkter Aufruf von interrupt-Prozeduren ist nicht erlaubt)

Compiler-Fehler 116: Must be in 8087 mode to compile

(Datentyp/Konstrukt nur im Modus {$N+} möglich)

Operationen mit den Variablentypen Single, Double, Extended oder Comp setzen die Aufnahme der Bibliothek für den Coprozessor 80x87 durch den Compilerschalter **{$N+}** voraus.

Wenn Ihr Rechner nicht mit einem Coprozessor ausgerüstet ist, müssen Sie zusätzlich den Emulator mit dem Schalter {$E+} einbinden.

Compiler-Fehler 117: Target address not found

(angegebene Adresse ist nicht Teil des Programms)
Die Suche nach einem Laufzeitfehler mittels Search|**Find Error** konnte an der angegebenen Adresse keinen Befehl des Programmes finden.

Compiler-Fehler 118: Include files are not allowed here

(Include-Befehl an dieser Stelle nicht erlaubt)

Compiler-Fehler 120: NIL expected

(nil erwartet)
Typisierte Zeigerkonstanten können nur mit NIL vorbesetzt werden.

Compiler-Fehler 121: Invalid qualifier

(ungültige Qualifizierung)
Diese Fehlermeldung erhalten Sie bei dem Versuch,

- eine Variable zu indizieren, die kein Array ist,
- auf Felder einer Variablen zuzugreifen, die kein Record ist, oder
- eine Variable zu dereferenzieren, die kein Zeiger ist.

Compiler-Fehler 122: Invalid variable reference

(ungültiger Variablenbezug)
Das angegebene Konstrukt folgt den Regeln eines Variablenbezugs, ergibt aber letztendlich nicht die Speicheradresse einer Variablen.

Wahrscheinlich haben Sie eine Zeigervariable verwendet und vergessen, das Ergebnis dieser Funktion in eine Adresse umzuwandeln.

Compiler-Fehler 123: Too many symbols

(Symbole belegen mehr als 64 KByte Speicherplatz)

Das Programm oder die Unit deklariert so viele Symbole, daß 64 KByte Speicherplatz dafür nicht mehr ausreichen.

Wenn Sie mit dem Schalter **{$D+}** übersetzen, schalten Sie ihn ab - allerdings ist dann eine Zuordnung von Laufzeitfehlern nicht mehr möglich.

Sie könnten auch einige Prozeduren und Funktionen in eine separate Unit auslagern.

Compiler-Fehler 124: Statement part too large

(Anweisungsteil zu groß)
Der Anweisungsteil von Prozeduren, Funktionen oder des Hauptprogramms ist auf jeweils rund 24 KByte beschränkt.

Sie sollten versuchen, Teile des Anweisungsteils als Prozeduren oder Funktionen auszulagern.

Compiler-Fehler 126: Files must be var parameters

(Datei-Variablen müssen VAR-Parameter sein)

Compiler-Fehler 127: Too many conditional symbols

(Zu viele Symbole für die bedingte Compilierung)
Die Tabelle, in der die Compiler-Symbole für die bedingte Compilierung eingetragen werden, ist voll.

Versuchen Sie, kürzere Symbolnamen zu verwenden, oder löschen Sie nicht mehr benötigte Symbole.

Compiler-Fehler 128: Misplaced conditional directive

({$ELSE} oder {$ENDIF} ohne vorangehendes {$IF...})
Der Compiler hat eine **{$ELSE}** oder **{$ENDIF}**−Direktive gefunden, ohne daß zuvor eine Bedingung mit **{$IFDEF}**, **{$IFNDEF}** oder **{$IFOPT}** eingeleitet worden wäre.

Compiler-Fehler 129: ENDIF directive missing

({$ENDIF} fehlt)
Innerhalb der Quelltext-Datei wurde eine Bedingung mit {$IFxxx} begonnen, die am Ende der Datei noch nicht mit **{$ENDIF}** abgeschlossen wurde.

Compiler-Fehler 130: Error in initial conditional defines

(Direkt definierte bedingte Symbole enthalten Fehler)

Die mit Options|Compiler|**Conditional Defines** (oder mit dem Kommandozeilen-Parameter **/D**) definierten Symbole sind ungültig.

Turbo Pascal für Windows erwartet Null oder mehr Bezeichner, die durch Leerzeichen, Kommata oder Strichpunkte getrennt sind.

Compiler-Fehler 131: Header does not match previous definition

(Prozedur-/Funktions-Definition ungleich Deklaration)
Der Kopf dieser Prozedur oder Funktion stimmt nicht mit dem überein, was bei der Definition im **Interface-Teil** oder bei der **FORWARD-Deklaration** angegeben wurde.

Compiler-Fehler 132: Critical disk error

(Diskettenfehler während der Compilierung)

Fehlerursache ist z.B. offene Laufwerksklappe, schadhafte Diskette

Compiler-Fehler 133: Cannot evaluate this expression

(Dieser Ausdruck läßt sich in diesem Kontext nicht auswerten)
Sie haben in einem Konstanten-Ausdruck eine Funktion der Laufzeitbibliothek verwendet, die der Compiler nicht für Konstanten zuläßt (z.B. Sin).

Compiler-Fehler 136: Invalid indirect reference

(Unzulässiger indirekter Bezug)
Sie haben versucht, auf einen Bezeichner zuzugreifen, dessen Adresse dem Compiler unbekannt ist. Ein weiterer Grund für diese Meldung sind Bezüge auf nicht definierte Variablen durch inline-Befehle.

Compiler-Fehler 137: Structured variables are not allowed here

(Stukturierte Variablen sind an dieser Stelle nicht erlaubt)
Mit dieser Meldung reagiert der Compiler auf Operatoren und Operationen, die sich nicht mit strukturierten Variablen durchführen lassen (z.B. der Versuch, zwei Records zu multiplizieren).

Compiler-Fehler 140: Invalid floating-point operation

(Gleitkomma-Rechenfehler)
Mit dieser Meldung reagiert der Compiler, wenn die Bewertung einer Konstanten mit einer Division durch Null oder einem Überlauf endet.

Compiler-Fehler 142: Procedure or function variable expected

(Prozedur- oder Funktions-Variable erwartet)
Der Compiler erwartet an dieser Stelle den Bezeichner einer Prozedur oder Funktion, z.B. bei einem Adreßoperator (@).

Compiler-Fehler 143: Invalid procedure or function reference

(Bezug auf Prozedur/Funktion in dieser Weise nicht zulässig)
Diese Meldung kann die folgenden Gründe haben:
Der Name einer Prozedur taucht in einem Ausdruck auf.

Der Compiler moniert die Zuweisung einer Routine an eine Prozedur-Variable: die Routine wurde entweder nicht mit {$F+}fdirectiv übersetzt oder ist als inlinei oder **interrupt** deklariert.

Compiler-Fehler 146: File access denied

(Zugriff auf Datei verweigert)

Compiler-Fehler 147: Object type expected

(Objekttyp erwartet)
Die angegebene Variable ist nicht Instanz eines Objekts

Compiler-Fehler 148: Local object types are not allowed

(Lokale Objekttypen sind nicht zulässig)
Objekttypen dürfen nur im äußersten Bezugsrahmen eines Programms oder einer Unit definiert werden.

Die Definition von Objekttypen innerhalb von Prozeduren oder Funktionen ist nicht zulässig.

Compiler-Fehler 149: VIRTUAL expected

(virtual erwartet)

Compiler-Fehler 150: Method identifier expected

(Bezeichner für Methode erwartet)

Compiler-Fehler 151: Virtual constructors are not allowed

(Virtuelle Konstruktoren sind nicht zugelassen)

Compiler-Fehler 152: Constructor identifier expected

(Bezeichner für einen Konstruktor erwartet)

Compiler-Fehler 153: Destructor identifier expected

(Bezeichner für einen Destruktor erwartet)

Compiler-Fehler 154: Fail only allowed within constructors

(Fail darf nur innerhalb von Konstruktoren verwendet werden.)

Compiler-Fehler 155: Invalid combination of opcode and operands

(Ungültige Kombination von Opcodes und Operanden)
Diese Kombination von Assembler-Opcode und Operand ist nicht zulässig. Dies kann folgende Gründe haben:

- Der Assembler-Opcode enthält entweder zu viele oder zu wenige Operanden, z.B. INC AX,BX oder MOV AX.

- Die Anzahl der Operanden ist zwar korrekt, ihre Typen oder ihre Anordnung ist aber nicht zulässig, z.B. DEC 1, MOV AX,CL oder MOV 1,AX

Compiler-Fehler 156: Memory reference expected

(Speicherreferenz erwartet)
An dieser Stelle wird eine Speicherzelle als Operand erwartet.

Vermutlich haben Sie vergessen, eckige Klammern um den Operanden zu setzen, z.B. MOV AX,BX+SI statt MOV AX,[BX+SI].

Compiler-Fehler 157: Cannot add or subtract relocatable symbols

(Kann verschiebbares Symbol nicht addieren oder subtrahieren)
Die einzigen Operationen, die auf ein relozierbares Symbol angewendet werden dürfen, sind Addition und Subtraktion einer Konstanten.

Variablen, Prozeduren, Funktionen und Labels sind verschiebbare Symbole. Wenn Var eine Variable und Const eine Konstante ist, sind auch MOV AX,Const+Const und MOV AX,Var+Const gültige Ausdrücke, der Ausdruck MOV AX,Var+Var hingegen nicht.

Compiler-Fehler 158: Invalid register combination

(Ungültige Kombination von Registern)
Die einzigen gültigen Registerkombinationen sind [BX], [BP], [SI], [DI], [BX+SI], [BX+DI], [BP+SI] und [BP+DI]. Alle anderen Kombinationen mit Indizierungen, z.B. [AX], [BP+BX] oder [SI+DX] sind nicht erlaubt.

Lokale Variablen (Variablen, die in Prozeduren und Funktionen deklariert sind) werden immer dem Stack zugeordnet, der Zugriff auf sie erfolgt über das BP-Register.

Compiler-Fehler 159: 286/287 instructions are not enabled

(286/287 Anweisungen sind nicht zugelassen)
Der Compiler-Schalter **{$G+}** erlaubt die Erzeugung von 287/287 Opcodes, das Programm kann aber auf Rechnern mit 8088- oder 8086-CPU nicht mehr ausgeführt werden.

Compiler-Fehler 160: Invalid symbol reference

(Ungültige Symbolreferenz)
Dieses Symbol darf nicht in einem Assembler-Operanden stehen. Mögliche Gründe hierfür sind:

- Sie haben versucht, eine Standardprozedur, eine Standardfunktion oder die besonderen Arrays Mem, MemW, MemL, Port oder PortW in einem Assembler-Operanden zu verwenden.

- Sie haben versucht, einen String, eine Gleitkomma- oder eine Mengenkonstante in einem Assembler-Operanden zu verwenden.

- Sie haben versucht, auf eine Inline-Routine zuzugreifen.

- Sie haben versucht, auf das Symbol @Result außerhalb einer Funktion zuzugreifen.

- Sie haben versucht, eine short JMP-Anweisung an einer anderen Stelle als eine durch ein Label definierte zu erzeugen.

Compiler-Fehler 161: Code generation error

(Fehler bei Code-Erzeugung)
Der vorausgegangene Anweisungsblock enthält eine LOOPNE, LOOPE, LOOP oder JCXZ Anweisung, deren Ziel-Label nicht erreichbar ist.

Compiler-Fehler 162: ASM expected

(asm erwartet)

Compiler-Fehler 163: Duplicate dynamic method index

(Doppelter Index für dynamische Methode)
Der Index der dynamischen Methode wurde schon von einer anderen Methode verwendet.

Compiler-Fehler 164: Duplicate resource indentifier

(Doppelte Ressourcen-ID)
Diese Ressourcen-Datei enthält eine Ressource, deren Name oder ID schon von einer anderen Ressource verwendet wird.

Compiler-Fehler 165: Duplicate or invalid export index

(Doppelter oder ungültiger Export-Index)
Die Ordinalzahl der Indexklausel ist nicht im Bereich 1..32767 oder wurde schon von einer anderen exportierten Routine verwendet.

Compiler-Fehler 166: Procedure or function identifier expected

(Prozedur- oder Funktion-Bezeichner erwartet)
Die Exportklausel gestattet nur, Prozeduren und Funktionen zu exportieren.

Compiler-Fehler 167: Cannot export this symbol

(Kann diesen Bezeichner nicht exportieren)
Eine Prozedur oder Funktion kann nur exportiert werden, wenn sie in der Export-Funktion deklariert ist.

Compiler-Fehler 168: Duplicate export name

(Doppelter Export-Name)
Der Name in der Namensklausel wurde schon von einer anderen exportierten Routine verwendet.

Kommandozeilen-Optionen

Der Kommandozeilen-Compiler von Turbo Pascal für Windows (**TPCW.EXE**) wird von der DOS-Kommandoebene aus aufgerufen und stellt sämtliche Funktionen des IDE-Compilers (**TPW.EXE**) zur Verfügung.

Der Aufruf von TPCW.EXE hat folgende Syntax:

```
TPCW   [Optionen]   <Dateiname>   [Optionen]
```

`[Optionen]` können eine oder mehrere Optionen sein, die durch Leerzeichen getrennt sind.

- Ein **+** (oder Leerzeichen) nach der Option schaltet sie ein
- Ein **-** nach der Option schaltet sie aus.

Compiler-Optionen

Option	Bedeutung
/$A	Daten ausrichten (Align Data)
/$B	Boolsche Auswertung (Boolean Evaluation)
/$D	Debugger-Informationen (Debug Information)
/$F	FAR-Aufrufe erzwingen (Force FAR Calls)
/$G	286-Code erzeugen (Generate 286 Instructions)
/$I	Ein-/Ausgabe prüfen (Input/Output Checking)
/$L	Lokale Bezeichner (Local Symbol Information)
/$N	80x87-Code für numerischen Koprozessor (80x87 Code - Numeric coprocessor)
/$R	Bereichsüberprüfung (Range Checking)
/$S	Stack-Überprüfung (Stack-Overflow Checking)
/$V	Überprüfung von Stringvariablen (String Var Checking)
/$W	Windows Stack Frame
/$X	Extended Syntax

Compilermodus-Optionen

Option	Bedeutung
/B	alles compilieren (Build All)
/F	Fehler finden (Find Error)
/L	Link-Puffer (Link Buffer)
/M	Compilieren und Linken (Make)
/Q	Quiet (kein IDE-Äquivalent)

Bedingte Definitionen Option

/D	Bedingte Definitionen (Conditional Defines)

Debug Optionen

Option	Bedeutung
/G	Map-Datei (.MAP File)
/V	Integrierte Debugger-Information (Debug Info in EXE Option)

Verzeichnis-Optionen

Option	Bedeutung
/E	EXE/TPU-Verzeichnis (EXE and TPU Directory)
/I	Include-Verzeichnisse (Include Directories)
/O	Objekt-Datei-Verzeichnisse (Object Files Directories)
/R	Ressourcen-Verzeichnisse (Resource Directories)
/T	Turbo-Verzeichnis (Turbo Directory)
/U	Unit-Verzeichnisse (Unit Directories)

Die meisten Kommandozeilen-Optionen haben Menüäqivalente. Der Hilfebildschirm **Kommandozeilen-Optionen und IDE-Äquivalente** enthält eine Zusammenfassung.

Kommandozeilen-Optionen und IDE-Äquivalente

Option	IDE Equivalent
/$A	Options|Compiler|**Align Data**
/$B	Options|Compiler|**Boolean Evaluation**
/$D	Options|Compiler|**Debug Information**
/$F	Options|Compiler|**Force Far Calls**
/$G	Options|Compiler|**286 Code**
/$I	Options|Compiler|**I/O Checking**
/$L	Options|Compiler|**Local Symbols**
/$M	Options|Compiler|**Memory Sizes**
/$N	Options|Compiler|**80x87 Code**
/$R	Options|Compiler|**Range Checking**
/$S	Options|Compiler|**Stack Checking**
/$V	Options|Compiler|**String Var Checking options**
/$W	Options|Compiler|**Windows Stack Frames**
/$X	Options|Compiler|**Extended Syntax**
/B	Compile|**Build**
/D	Options|Compiler|**Conditional Defines**
/E	Options|Directories|**EXE and TPU Directory**
/F	Search|**Find Error**
/G	Options|Linker|**Map File**
/I	Options|Compiler|**Include Directories**
/L	Options|Linker|**Link Buffer**
/M	Compile|**Make**
/O	Options|Directories|**Object Directories**
/Q	(kein)
/R	Options|Directories|**Resource Directories**
/T	(kein)
/U	Options|Directories|**Unit Directories**
/V	Options|Linker|**Debug Info in EXE**

/V Kommandozeilen-Option

Die Kommandozeilen-Option /V hängt Debugger-Informationen für den Turbo Debugger am Ende der .EXE-Datei an.

Die Option berührt den Code der .EXE-Datei nicht. Beim Laufen des Programms wird kein zusätzlicher Speicherplatz gebraucht.

Siehe auch

Options|Linker|**Debug Info in Exe**

/Q Kommandozeilen-Option

Die Kommandozeilen-Option /Q unterdrückt die Ausgabe von Dateinamen und Zeilennummern beim Compilieren.

Falls ein Fehler auftritt, wird er gemeldet.

/L Kommandozeilen-Option

Die Kommandozeilen-Option /L schaltet beim Binden mehrerer Module zu einer .EXE-Datei die Zwischenspeicherung im Hauptspeicher ab. Dateien werden dann von der Platte statt aus dem Arbeitsspeicher gelesen.

Bei sehr großen Programmen ist /L möglicherweise zum erfolgreichen Linken nötig.

Siehe auch

Options|Linker|**Link Buffer**

/G Kommandozeilen-Option

Die Kommandozeilen-Option /G weist TPCW an, eine .MAP-Datei zu erzeugen, die den Aufbau der .EXE-Datei zeigt.

Der Kommandozeilen-Option /G muß S, P oder L folgen, um den Informationsgehalt der .MAP-Datei zu bestimmen.

- Die Option /GS zeigt den Abschnitt Segments der .MAP-Datei an.
- Die Option /GP zeigt die Abschnitte Segments und Publics der .MAP-Datei an.
- Die Option /GL zeigt die Abschnitte Segments, Publics, und Line Numbers der .MAP-Datei an.

Siehe auch

Options|Linker|**Map File**

/R Kommandozeilen-Option

Die Option /R gibt TPCW an, wo nach .RES-Dateien gesucht werden soll, die mit dem Ressourcen-Editor erstellt wurden.

Siehe auch

Options|Directories|**Resource Directories**

/O Kommandozeilen-Option

Mit der Kommandozeilen-Option /O geben Sie eine Verzeichnisliste an, in der nach .OBJ-Dateien gesucht wird, die externe Assembler-Routinen enthalten.

Die Verzeichnisnamen werden durch Strichpunkte getrennt.

Siehe auch

Options|Directories|**Object Directories**

/U Kommandozeilen-Option

Mit der Kommandozeilen-Option /U werden zusätzliche Verzeichnisse angegeben, in denen nach Units gesucht wird.
Die Verzeichnisnamen werden durch Strichpunkte getrennt.

Gemäß Standardeinstellung sucht TPCW zuerst in der Datei TPW.TPL. Sind die Units dort nicht enthalten, setzt TPCW die Suche nach UNITNAME.TPU im aktuellen Verzeichnis fort.

Siehe auch

Options|Directories|**Unit Directories**

/I Kommandozeilen-Option

Mit der Kommandozeilen-Option /I geben Sie eine Liste von Verzeichnissen an, in denen nach Include-Dateien gesucht wird.
Die Verzeichnisnamen werden durch Strichpunkte getrennt.

Siehe auch

Options|Directories|**Include Directories**

/E Kommandozeilen-Option

Die Kommandozeilen-Option /E gibt das Verzeichnis an, in dem TPCW die erzeugten .EXE- und .TPU-Dateien speichern soll.

Siehe auch

Options|Directories|**EXE and TPU Directory**

/T Kommandozeilen-Option

Die Kommandozeilen-Option /T sagt TPCW, wo
- die Konfigurationsdatei (TPCW.CFG) und
- die residente Library-Datei (TPCW.TPL) zu finden sind.

Befinden sich diese Dateien im aktuellen Verzeichnis oder im Verzeichnis von TPCW.EXE, brauchen Sie diese Option nicht anzugeben.

Diese Option muß das erste Kommandozeilen-Argument sein.

/D Kommandozeilen-Option

Mit der Kommandozeilen-Option /D definieren Sie bedingte Symbole. Dies entspricht der Anweisung **DEFINE** am Programmbeginn.

Siehe auch

Options|Compiler|**Conditional Defines**

/B Kommandozeilen-Option

Die Kommandozeilen-Option /B compiliert alle Programmteile neu. Die Aktualität einzelner Module wird nicht berücksichtigt.

Dies entspricht **/M** ohne Bedingungen. (/M compiliert nur nicht aktuelle Dateien).

Siehe auch

Compile|**Build**

/M Kommandozeilen-Option

Die Kommandozeilen-Option /M erzeugt eine .EXE-Datei nach folgenden Regeln:

- Wurde die Quelldatei einer gegebenen Unit seit Erzeugen der .TPU (object code) Datei geändert, wird sie neu compiliert.
- Wurde der Interface-Abschnitt einer Unit, auf die sich in einer Uses-Anweisung bezogen wird, geändert, werden alle davon abhängigen Units neu compiliert.
- Ist eine gelinkte .OBJ-Datei neuer als die .TPU-Datei der Unit, wird die Unit neu compiliert.
- Enthält eine Unit eine Include-Datei, die neuer ist als die .TPU-Datei der Unit, wird die Unit neu compiliert.

Diese Option entspricht **/B** mit Bedingungen. (/B compiliert alle Dateien).

HINWEIS: Units in TPW.TPL sind von diesem Prozeß ausgeschlossen.

Siehe auch

Compile|**Make**

/F Kommandozeilen-Option

Die Kommandozeilen-Option /F findet einen Laufzeitfehler im Quellcode.

Wenn ein Programm mit einem Laufzeitfehler abbricht, werden Fehlercode und Adresse (segment:offset) angezeigt.

Um den Fehler im Programmcode zu lokalisieren, geben Sie diese Adresse mit der Option /F an:

```
/Fsegment:offset
```

HINWEIS: Programm und Units müssen mit eingeschalteter Kommandozeilen-Option **/$D** compiliert worden sein.

Siehe auch

Search|**Find Error**

Das Menü Compile (Compilieren)

Sie verwenden zum Compilieren oder Linken des Programms im aktuellen Fenster das Menü Compile (Compilieren).

Um die Compilierbefehle Compile, Make und Build (Compilieren, Projekt aktualisieren, Alle Projektdateien compilieren) verwenden zu können, muß in einem aktiven Editierfenster eine Datei offen sein oder schon eine Datei angegeben sein.
Dafür stehen Ihnen folgende Befehle zur Verfügung:

Compile (Coompilieren)
Make (Projekt aktualisieren)
Build (Alle Projektdateien compilieren)
Primary File (Hauptdatei)
Clear Primary File (Eintrag für Hauptdatei löschen)
Information (Informationen)

Compile|Compile (Compilieren/Compilieren)

Der Befehl Compile|Compile (Compilieren/Compilieren) compiliert die Datei im aktiven Edit-Fenster.

Das Informationsfenster **Compile Status (Information)** zeigt den Ablauf der Compilierung und deren Ergebnisse an. Wählen Sie OK, wenn die Compilierung erfolgreich durchgeführt wurde.

Bei Fehlern wird der Cursor automatisch auf den Text der ersten Meldung im Edit-Fenster positioniert.

Zum Korrigieren entfernen Sie zuvor die Meldung durch Anklicken mit der Maus, oder drücken Sie irgendeine Taste.

Das Informationsfenster Compile Status (Information über Compilierung)

Das Informationsfenster Compile Status (Information über Compilierung) zeigt den Ablauf der Compilierung und deren Ergebnisse an.

Compile|Make (Compilieren/Projekt aktualisieren)

Über Make EXE File (Projekt aktualisieren) wird der Project-Manager aufgerufen, der eine ausführbare .EXE-Datei erzeugt. Der Name der EXE-Datei wird aus einer der nachstehend angegebenen Möglichkeiten abgeleitet:

- Wenn **Primary File (Hauptdatei)** eine Hauptdatei anzeigt, dann wird diese Datei compiliert, sonst die letzte Datei, die in den Editor geladen worden ist.
- Turbo Pascal prüft alle Module, von denen die Datei, die compiliert werden soll, abhängt.
- Wenn die Quell-Datei einer Unit NACH dem Entstehen der .TPU-Datei (Objektcode) geändert worden ist, wird die Unit neu compiliert.
- Wenn das Interface einer Unit geändert worden ist, werden alle anderen Units, die sich darauf beziehen, neu compiliert.
- Wenn eine .OBJ-Datei (externe Routinen) in eine Unit eingebunden wird, und diese .OBJ-Datei jünger als die .TPU-Datei der Unit ist, wird die Unit neu compiliert.

- Wenn eine Include-Datei in eine Unit eingebunden wird und diese Include-Datei jünger als die .TPU-Datei der Unit ist, wird die Unit neu compiliert.

- Wird der Programmcode einer Unit nicht gefunden, wird diese Unit nicht neu compiliert, sondern im alten Zustand verwendet.

Make funktioniert wie Compile|**Build (Alle Projektdateien compilieren)**, außer mit der oben angeführten Bedingung. (Build compiliert alle Dateien, auch wenn sie alten Datums sind).

Compile|Build (Alle Projektdateien compilieren)

Der Befehl Build (Alle Projektdateien compilieren) stellt ungeachtet deren Aktualität alle Teile Ihres Programms wieder her.

Diese Option führt dieselben Schritte wie Compile|**Make (Projekt aktualisieren)** durch, außer mit der oben angeführten Bedingung.

Der Befehl Compile|Build (Alle Projektdateien compilieren) compiliert alle Dateien, die in der Hauptdatei (Primary File) genannt sind, neu.

Einen durch Ctrl+Break oder falsche Tastatureingaben abgebrochenen Build-Befehl können Sie wieder aufnehmen, indem Sie Compile|Make wählen.

Compile|Primary File (Compilieren/Hauptdatei)

Der Befehl Compile|Primary File ruft das Dialogfenster **Primary File (Hauptdatei)** auf, in dem Sie bestimmen, welche .PAS-Datei compiliert werden soll, wenn einer der Befehle Compile|**Make(Projekt aktualisieren)** oder Compile|**Build (Alle Projektdateien compilieren)** aufgerufen wird.

Benutzen Sie den Befehl Primary File (Hauptdatei), wenn Ihr Programm mehrere Unit- (.TPU) oder Include-Dateien benutzt.

Unabhängig davon, welche Datei Sie gerade editieren, arbeiten Make oder Build immer mit der Hauptdatei.

Wenn Sie eine andere Datei als Hauptdatei spezifizieren, aber die Datei im aktiven Editorfenster compilieren möchten, dann wählen Sie **Compile (Compilieren)**.

Das Dialogfenster Primary File (Hauptdatei)

Im Dialogfenster Primary File (Hauptdatei) geben Sie die .PAS-Datei an, die compiliert werden soll, wenn Sie den Befehl Compile|**Make (Projekt aktualisieren)** oder Compile|**Build (Alle Projektdateien compilieren)**.

Das Eingabefeld File Name (Dateiname)

In dieses Eingabefeld tragen Sie den Namen der .PAS-Datei ein, die compiliert werden soll. Sie können hier auch eine Suchmaske für Dateinamen angeben, die im Listenfenster Dateien angezeigt werden sollen.

Das Listenfenster Files (Dateien)

Das Dateilistenfenster listet alle Dateien im aktuellen Verzeichnis auf, die der Eingabemaske entsprechen und alle über- und untergeordneten Verzeichnisse.

Das Listenfenster Directories (Verzeichnisse)

Prüfen Sie den Inhalt verschiedener Verzeichnisse durch Wahl eines Verzeichnisses in diesem Listenfenster.

Sie können auch Tastenkürzel (die unterstrichenen Buchstaben im Menü) verwenden, um den gewünschten Bereich des Dialogfensters zu aktivieren.

Das Eingabefeld File Name (Dateiname) im Dialogfenster Primary File (Hauptdatei)

In dieses Eingabefeld geben Sie den Namen der .PAS-Datei ein, die compiliert werden soll, oder einen Dateinamen, der Jokerzeichen enthält. Auf diese Art ausgewählte Dateien werden in der Dateiliste aufgeführt.

Dieses Eingabefeld hat eine zugehörige **Eingabeaufzeichnungsliste**.

Das Listenfenster File name (Dateiname) im Dialogfenster Primary File (Hauptdatei)

Das Dateilistenfenster listet alle Dateien im aktuellen Verzeichnis auf, die der Eingabemaske **File name (Dateiname)** entsprechen und allenüber- und untergeordneten Verzeichnisse.

Heben Sie den Namen der gewünschten Hauptdatei (Primary File) hervor (klicken Sie den Namen an oder positionieren Sie den Cursor auf den Namen), und wählen Sie dann OK.

Listenfenster Directories (Verzeichnisse) im Dialogfenster Primary File (Hauptdatei)

Dieses Listenfenster listet die verfügbaren Verzeichnisse und Laufwerke auf.

Alt+D öffnet dieses Fenster. Das erste Verzeichnis ist markiert.

Durch Doppelklicken auf einen Verzeichnisnamen setzen Sie dieses als aktuelles Verzeichnis ein.

Wenn Sie die Tastatur verwenden, wählen Sie mit den Pfeiltasten das gewünschte Verzeichnis oder Laufwerk und bestätigen mit OK oder Enter.

Anklicken des Symbols [...] wechselt zum übergeordneten Verzeichnis des aktuellen Verzeichnisses.

Clear Primary File (Hauptdatei löschen)

Der Befehl Compile|Clear Primary File (Compilieren/Hauptdatei löschen) löscht die aktuelle Hauptdatei, die mit dem Befehl Compile|**Primary File (Hauptdatei)** genannt wurde.

Wurde keine Hauptdatei (primary file) angegeben, wirkt dieser Befehl nicht.

Compile|Information

Der Befehl Compile|Information zeigt das Fenster **Compile Information (Informationen über Compilierung)**, das den Namen der Hauptdatei (falls vorhanden) und den Namen der zuletzt compilierten Datei anzeigt.

Das Dialogfenster Compile|Information (Compilieren/Informationen über Compilierung)

Dieses IInformationsfenster zeigt:

- den Namen der Hauptdatei (Primary file), sofern vorhanden
- den Namen der zuletzt compilierten Datei
- Größe des compilierten Quellcodes
- Größe des generierten Codes
- Datengröße
- Stack-Größe
- Größe des lokalen Heap

Turbo Pascal Systemmenü

Das Systemmenü von Turbo Pascal für Windows befindet sich am linken Ende der Titelleiste und es wird durch Anklicken oder durch Betätigen von Alt+Leertaste angezeigt.

Die Befehle dieses Menüs betreffen den Turbo-Pascal-Desktop.

Jedes **Editorfenster** und Dialogfenster besitzt ein derartiges Systemmenü.

Das Turbo Pascal Systemmenü enthält folgende Befehle:
Restore (Wiederherstellen)
Move (Verschieben)
Size (Größe ändern)
Minimize (Symbol)
Maximize (Vollbild)
Close (Schließen)
Switch to (Wechseln zu)

Turbo Pascal Control|Restore (System/Wiederherstellen)

Der Befehl Restore (Wiederherstellen) versetzt den Turbo -Pascal-Desktop in seine vorherige Größe.

Dieser Befehl ist nur wirksam, wenn der Turbo-Pascal-Desktop vorher **vergrößert** oder auf Symbolgröße **verkleinert** wurde.

Turbo Pascal Control|Move (System/Verschieben)

Der Befehl Move (Verschieben) ermöglicht Ihnen, den Turbo-Pascal-Desktop mit der Tastatur zu verschieben.

Dazu verwenden Sie die Pfeiltasten; Enter beendet den Vorgang.

Dieser Befehl ist unwirksam, wenn der Turbo-Pascal-Desktop auf Vollbildgröße **vergrößert** wurde.

Sie können den Desktop auch verlagern, indem Sie seine Titelleiste mit der Maus ziehen.

Turbo Pascal Control|Size (System/Größe ändern)

Sie könnn die Größe des Desktops mit der Tastatur und dem Befehl Size (Größe ändern) verändern.

Mit den Pfeiltasten können Sie die Fensterränder bewegen; Enter beendet den Vorgang.

Der Befehl ist nur wirksam, wenn der Turbo-Pascal-Desktop nicht auf Vollbild **vergrößert** ist.

Turbo Pascal Control|Minimize (System/Symbol)

Der Befehl Minimize (Symbol) verkleinert den Turbo-Pascal-Desktop in das Turbo Pascal Symbol.

Er ist nur wirksam, wenn der Turbo-Pascal-Desktop nicht bereits verkleinert ist.

Turbo Pascal Control|Maximize (Symbol/Vollbild)

Wenn Sie den Befehl Maximize (Vollbild) wählen, wird der Turbo-Pascal-Bildschirm vergrößert, bis er den gesamten Bildschirm ausfüllt.

Er ist nur wirksam, wenn der Turbo-Pascal-Desktop nicht bereits vergrößert wurde.

Turbo Pascal Control|Close (System/Schließen)

Der Befehl Close (Schließen) schließt den Turbo-Pascal-Desktop und entfernt Turbo Pascal aus dem Arbeitsspeicher.

Im CUA-Modus drücken Sie dazu Alt+F4; im Alternate-Modus Alt+X.

Sie können auch das Schließfeld in der oberen linken Ecke zum Schließen des Fensters anklicken.

Wurden Änderungen an einer nicht gespeicherten Datei vorgenommen, wird das Dialogfenster **Turbo Pascal**geöffnet, in welchem Sie gefragt werden, ob Sie die Datei abspeichern wollen.

Das Dialogfenster Turbo Pascal

Im Dialogfenster Turbo Pascal werden Sie gefragt, ob Sie die Datei vor dem Verlassen des Programms speichern wollen.

Durch Eingabe von Yes bzw. Ja speichern Sie die Datei ab, bei Eingabe von No bzw. Nein wird das Programm sofort verlassen, wobei die Datei nicht gespeichert wird.

Mit Cancel bzw. Abbrechen brechen Sie die Operation ab; der Cursor befindet sich danach wieder im aktiven Fenster.

Turbo Pascal Control|Switch To (System/Wechseln zu)

Der Befehl Switch To (Wechseln zu) zeigt das Dialogfenster **Task List (Task-Liste)** an, in welchem von einer Anwendung zu einer anderen gewechselt werden kann.

Das Dialogfenster Task List (Task-Liste)

In diesem Dialogfenster können Sie zwischen Anwendungen wechseln und die Fenster der Anwendungen neu ordnen.

Task-Liste
Alle offenen Anwendungen werden in diesem Dialogfenster in einer Liste angezeigt.

Task-Liste

Das Listenfenster Task-Liste führt alle offenen Anwendungen auf.

Die gewünschte Anwendung kann durch Doppelklick gewählt werden.

Mit der Tastatur können Sie die Anwendung über die Pfeiltasten markieren und mit Enter wählen.

Die **Aktionsschalter** schalten zur gewünschten Anwendung und ordnen die Fenster.

Aktionsschalter

Aktionsschalter schalten zur gewünschten Anwendung und ordnen die Fenster.

Klicken Sie den gewünschten Aktionsschalter an. Mit der Tastatur führt Tab zum richtigen Schalter; Enter betätigt ihn.

Der Schalter Switch To (Wechseln zu) schaltet zu einer anderen Anwendung aus der Liste.

Der Schalter End Task (Task beenden) schließt die gewählte Anwendung.

Der Schalter Cascade (Übereinander) ordnet die vorhandenen Fenster überlappend an.

Der Schalter Tile (Nebeneinander) ordnet die Fenster nebeneinander an.

Der Schalter Arrange Icons (Symbole anordnen) verteilt die Symbole gleichmäßig am unteren Rand des Bildschirms.

Das Systemmenü im Editorfenster

Jedes Editorfenster besitzt ein derartiges Systemmenü, wenn das Fenster aktiv ist.

Die Befehle dieses Menüs ähneln denen des **Turbo Pascal Systemmenüs**.

Das Systemmenü im Editorfenster hat folgende Befehle:
Restore (Wiederherstellen)
Move (Verschieben)
Size (Größe ändern)
Minimize (Symbol)
Maximize (Vollbild)
Close (Schließen)
Next (Nächstes)

Editorfenster, Control|Restore (System/Wiederherstellen)

Der Befehl Restore (Wiederherstellen) versetzt das Editorfenster in seine vorherige Größe.

Dieser Befehl ist nur wirksam, wenn das Editorfenster vorher **vergrößert** oder **verkleinert** wurde.

Editorfenster, Control|Move (System/Verschieben)

Der Befehl Move (Verschieben) ermöglicht Ihnen, das Editorfenster mit der Tastatur zu verschieben.

Sie verwenden dazu die Pfeiltasten; Enter beendet den Vorgang.

Dieser Befehl ist unwirksam, wenn das Editorfenster **vergrößert** ist.

Sie können das Fenster auch verlagern, indem Sie seine Titelleiste mit der Maus ziehen.

Editorfenster, Control|Size (System/Größe ändern)

Sie können die Größe des Bildschirmfensters mit der Tastatur und dem Befehl Size (Größe ändern) verändern.

Mit den Pfeiltasten können Sie die Fensterränder bewegen; Enter beendet den Vorgang.

Der Befehl ist nur wirksam, wenn das Editorfenster nicht **vergrößert** ist.

Wenn ein Fenster eine Feld zur Größenveränderung enthält, können Sie mit der Maus die Größe ändern.

Editorfenster, Control|Minimize (System/Symbol)

Der Befehl Minimize (Symbol) verwandelt das Editorfenster in das Turbo Pascal Symbol.

Er ist nur wirksam, wenn das Editorfenster nicht schon minimiert wurde.

Editorfenster, Control|Maximize (System/Vollbild)

Wenn Sie den Befehl Maximize (Vollbild) geben, wird das Editorfenster vergrößert bis es den gesamten Bildschirm füllt.

Er ist nur wirksam, wenn das Editorfenster nicht bereits vergrößert wurde.

Editorfenster, Control|Close (System/Schließen)

Der Befehl Close (Schließen) schließt das Editorfenster.

Sie können auch das Schließfeld in der oberen linken Ecke zum Schließen des Fensters anklicken.

Wurden Änderungen an einer Datei nicht gespeichert, wird das Dialogfenster **Turbo Pascal** eingeblendet, in welchem Sie gefragt werden, ob Sie die Datei speichern wollen.

Editorfenster, Control|Next (System/Nächstes)

Der Befehl Next (Nächstes) aktiviert das nächste offene Fenster oder Symbol.

Das Menü Edit (Bearbeiten)

Mit den Befehlen aus dem Menü Edit (Bearbeiten) können Sie Text in Editorfenstern ausschneiden oder kopieren und an anderer Stelle wieder einfügen. Sie können auch das Zwischenablage-Fenster öffnen und seinen Inhalt editieren. Das Menü enthält die folgenden Befehle:

Undo (Rückgängig)
Redo (Widerrufen)
Cut (Ausschneiden)
Copy (Kopieren)
Paste (Löschen)
Clear (Löschen)

Edit|Undo (Bearbeiten/Rückgängig)

Der Befehl Undo (Rückgängig) macht die letzten Eingaben oder Cursorbewegungen rückgängig.

Gelöschte Zeichen werden eingefügt, eingefügte gelöscht, überschriebene Zeichen restauriert und der Cursor auf die alte Position zurückgesetzt.

Wenn Sie eine Blockoperation zurücknehmen, erscheint Ihre Datei wie vor der Ausführung des Blockbefehls.

Wenn Sie den Befehl wiederholt geben, nimmt Turbo Pascal nacheinander die Änderungen an der Datei zurück, die in der aktuellen Arbeitssitzung vorgenommen wurden.

Undo ändert keine Einstellung von Optionen, die mehr als ein Fenster betreffen.

Die Option Group Undo (Befehlsgruppe widerrufen) im Dialogfenster Options| **Preferences (Optionen/Vorgaben)** bestimmt, wie sich die Befehle Undo (Rückgängig) und **Redo (Widerrufen)** verhalten.

Edit|Redo (Bearbeiten/Widerrufen)

Redo (Widerrufen) kehrt die Wirkung des letzten **Undo (Rückgängig)** Befehls um.

Dieser Befehl wirkt nur unmittelbar nach einem Undo (Rückgängig) oder Redo (Widerrufen).

Eine Folge von Redo-Befehlen kehrt die Wirkung einer Folge von Undo-Befehlen um.

Edit|Cut (Bearbeiten/Ausschneiden)

Der Befehl Cut (Ausschneiden) entfernt den markierten Text aus dem aktiven Editorfenster und kopiert ihn in die Zwischenablage. Mit Edit|**Paste (Bearbeiten/Einfügen)** können Sie diesen Text in ein anderes Fenster (oder an eine andere Stelle im selben Fenster) kopieren.

Der Text bleibt in der Zwischenablage, so daß man ihn beliebig oft kopieren kann.

Edit|Copy (Bearbeiten/Kopieren)

Copy (Kopieren) kopiert den markierten Text aus dem aktiven Editorfenster in die Zwischenablage, ohne ihn aus dem Fenster zu löschen. Mit **Paste (Einfügen)** können Sie diesen Text in ein anderes Fenster (oder an eine andere Stelle im selben Fenster) kopieren.

Sie können auch Text aus einem Hilfefenster kopieren. Drücken Sie Shift und eine Pfeiltaste, oder benutzen Sie die Maus mit gedrückter linker Taste, um den Text zu markieren. Wählen Sie anschließend Copy, um ihn in die Zwischenablage zu kopieren.

Edit|Paste (Bearbeiten/Einfügen)

Paste (Einfügen) fügt den in der Zwischenablage markierten Text an der Cursorposition im aktiven Edit-Fenster ein.

Edit|Clear (Bearbeiten/Löschen)

Clear (Löschen) löscht den markierten Text, ohne ihn in die Zwischenablage zu kopieren.

Dieser Text ist nicht mehr rekonstruierbar und steht für Einfügeoperationen nicht zur Verfügung.
Benutzen Sie hierfür **Cut (Ausschneiden)** oder **Copy (Kopieren)**.

Sie können zwar den gelöschten Text nicht mehr kopieren, aber den Löschbefehl mit **Undo (Rückgängig)** rückgängig machen.

Clear (Löschen) ist dann nützlich, wenn Sie Text löschen, aber den in der Zwischenablage befindlichen Text nicht überschreiben wollen.

κ+Das Menü File (Datei)

Dieses Menü steuert die Dateiverwaltung von Turbo Pascal. Zu jedem Menübefehl ist ein eigener Hilfebildschirm vorhanden:

New (Neu)
Open (Öffnen)
Save (Speichern)
Save As (Speichern unter)
Save All (Alles speichern)
Print (Drucken)
Printer Setup (Druckerinstallation)
Exit (Beenden)
Liste geschlossener Dateien

File|New (Datei/Neu)

New (Neu) eröffnet ein neues Editor-Fenster mit dem voreingestellten Namen NONAMExx.PAS (xx steht für eine Zahl zwischen 00 und 99) und macht es zum aktiven Fenster.

Die NONAME-Dateien werden als temporäre Editorpuffer benutzt; sobald Sie so eine Datei speichern wollen, werden Sie aufgefordert, einen anderen Namen anzugeben.

File|Open (Datei/Öffnen)

Der Befehl Open (Öffnen) blendet das Dialogfenster **File Open (Datei öffnen)** ein, mit dessen Hilfe Sie eine Programmdatei in ein Editor-Fenster laden können.

Im Alternate-Modus drücken Sie F3 zum Öffnen einer Datei.

Das Dialogfenster File Open (Datei öffnen)

Nach Auswahl des Befehls Open (Öffnen) wird das Dialogfenster Open a File (Datei öffnen) eingeblendet, mit dessen Hilfe Sie eine Programmdatei in ein Editor-Fenster laden können. Das Fenster enthält ein Eingabefeld, eine Dateinamensliste und die Aktionsschalter Open (OK) , Cancel (Abbruch) und Help (Hilfe).

Das Eingabefeld **File Name (Dateioname)**
Hier geben Sie den Namen der Datei an, die geladen werden soll. Jokerzeichen können als Filter für die Dateinamenliste verwendet werden.

Das Fenster Files (Dateien)
Das Fenster Files (Dateien) zeigt alle Dateien, die zur Maske im Eingabefeld passen, und zusätzliche alle entsprechenden Dateien des übergeordneten Verzeichnisses und der Unterverzeichnisse.

Das Fenster Directories (Verzeichnisse)
Sie geben hier einen Verzeichnisnamen ein, um den Inhalt verschiedener Verzeichnisse sehen zu können.

Sie können auch mit Tastenkürzeln (unterstrichenen Buchstaben) den gewünschten Begriff des Dialogfensters auswählen.

Das Eingabefeld File Name (Dateiname)

Hier geben Sie den Namen der Datei, die geladen werden soll, ein. Jokerzeichen als Filter für die Dateinamenliste sind zulässig.

Zu diesem Eingabefeld gehört eine **Eingabeaufzeichnungsliste**.

Sie können eine Datei auf folgende Weise öffnen:

- Tippen Sie einen Dateinamen ein (mit Pfadnamen, falls nötig) und wählen Sie OK oder Enter.
- Tippen Sie den Dateinamen mit Jokerzeichen ein, um eine Auswahl der Dateiliste zu sehen, und wählen Sie OK oder Enter.
- Drücken Sie die Tasten Alt+Pfeil ab oder klicken Sie den Pfeil im Eingabefenster an, um eine Datei aus der Eingabeaufzeichnungsliste zu wählen.
- Drücken Sie Pfeil auf, wenn der Cursor im Eingabefeld steht, um frühere Eingaben zu sehen.
- Drücken Sie Pfeil ab, um die Liste rückwärts zu druchsuchen.

Der Vorgabedateiname *.PAS zeigt alle Dateien mit der Dateinamenserweiterung PAS im Dialogfenster **Files List (Dateien)** an.

Wenn Sie ein anderes Muster mit Jokerzeichen angeben, speichert Turbo Pascal es, bis Sie das nächste Mal innerhalb derselben Arbeitssitzung das Dialogfenster **File Open (Datei öffnen)** aufrufen.

Das Fenster Files (Dateien)

Im Listenfenster Files (Dateien) werden alle Dateinamen aufgeführt, die zur Maske in Save File As (Datei speichern unter) passen. Die Dateien stammen aus dem aktuellen Verzeichnis sowie aus dem übergeordneten und den untergeordneten Verzeichnissen.

Durch Doppelklicken auf den Dateinamen wird die Datei geöffnet.

Wenn Sie die Tastatur verwenden, bewegen Sie den Cursor mit Tab bis zum Listenfenster und markieren mit den Pfeiltasten die Datei. Mit Enter öffnen Sie die Datei.

Das Fenster Directories (Verzeichnisse)

Im Listenfenster Directories (Verzeichnisse) werden die Namen der verfügbaren Laufwerke und Verzeichnisse aufgelistet.

Alt+D markiert das erste Verzeichnis im Listenfenster Directories.

Durch einen Doppelklick auf einen Verzeichnisnamen der Liste wechseln Sie in dieses Verzeichnis.

Mit der Tastatur wählen Sie das gewünschte Verzeichnis mit den Pfeiltasten und bestätigen die Auswahl mit OK oder Enter.

Wenn Sie das Symbol [...] sehen und mit Doppelklick wählen, wechseln Sie in das dem aktuellen übergeordnete Verzeichnis.

File|Save (Datei/Speichern)

Der Befehl Save (Speichern) speichert die Datei des aktiven Editor-Fensters auf die Festplatte.

Falls es sich um einen voreingestellten Namen (NONAMExx.PAS) handelt, werden Sie automatisch aufgefordert, im Dialogfenster **Save File As (Datei speichern unter)** den Namen zu ändern. Dabei können Sie auch die Angaben für das Verzeichnis und das Laufwerk ändern.

Wenn Sie einen vorhandenen Dateinamen verwenden, fragt Turbo Pascal, ob die Datei überschrieben werden soll.

Falls Sie alle geänderten Dateien speichern wollen, wählen Sie File|**Save All (Alles speichern)**.

Im Alternate-Modus drücken Sie zum Speichern einer Datei die Taste F2.

Das Dialogfenster File Save As (Datei speichern unter)

Der Befehl Save as (Speichern unter) öffnet das Dialogfenster Save File As (Datei speichern unter), in dem Sie im Eingabefeld **File Name (Dateiname)** den neuen Dateinamen (nach Bedarf mit Pfad) eingeben. Einen neuen Pfad können Sie aus dem Listenfenster **Directories (Verzeichnisse)** wählen.

Wenn Sie einen vorhandenen Dateinamen verwenden, fragt Turbo Pascal, ob Sie die Datei überschreiben wollen.

File|Save As (Datei/Speichern unter)

Der Befehl Save as (Speichern unter) öffnet das Dialogfenster **Save File As (Datei speichern unter)**, mit dessen Hilfe Sie die Datei aus dem aktiven Editor-Fenster unter einem anderen Namen und in einem anderen Verzeichnis oder Laufwerk speichern können.

Geben Sie den neuen Namen ein, optional Verzeichnis und/oder Laufwerk, und klicken oder wählen Sie OK. Die Namen aller Editor-Fenster mit dieser Datei werden automatisch umbenannt.

Wenn Sie einen vorhandenen Dateinamen verwenden, fragt Turbo Pascal, ob die Datei überschrieben werden soll.

File|Save All (Datei/Alles speichern)

Der Befehl Save All (Alles speichern) speichert alle Dateien, die in offenen Editor-Fenstern geladen sind.

File|Print (Datei/Drucken)

Der Befehl Print (Drucken) druckt den Inhalt des aktiven Editor-Fensters aus. Beim Ausdruck werden Tabulatoren durch die passende Anzahl von Leerzeichen ersetzt. Dieser Befehl ist deaktiviert, wenn das aktive Fenster nicht gedruckt werden kann.

File|Printer Setup (Datei/Druckerinstallation)

Der Befehl Printer Setup (Druckerinstallation) öffnet das Dialogfenster **Select Printer (Drucker auswählen)**, in welchem der Drucker für Ausgaben aus Turbo Pascal festgelegt wird.

Wenn Sie an der Vorgabeeinstellung nichts ändern wollen, brauchen Sie den Befehl Printer Setup (Druckerinstallation) nicht zu verwenden

Das Dialogfenster Select Printer (Drucker auswählen)

Im Dialogfenster Select Printer (Drucker auswählen) können Sie einen der aufgelisteten Drucker für Ausgaben aus Turbo Pascal wählen.

Wenn Sie die normale Konfiguration des Druckers ändern wollen, wählen Sie **Set Up (Drucker)**.

Manche Druckertreiber haben ihr eigenes Hilfesystem, das durch einen Aktionsschalter Help (Hilfe) angezeigt wird.

Wenn es keine derartige Hilfe gibt, können Sie im Microsoft Windows Anwenderhandbuch, Kapitel "Systemsteuerung" nachlesen.

Printer and Port (Drucker und Schnittstelle)
Das Listenfenster Printer and Port (Drucker und Schnittstelle) listet die Druckertreiber für Windows auf.

Das Dialogfenster Set Up (Drucker)
Das Dialogfenster Set Up (Drucker) enthält Optionen, die von den Möglichkeiten des Druckers abhängen.

Das Fenster Printer and Port (Drucker und Schnittstellen)

Das Listenfenster Printer and Port (Drucker und Schnittstelle) listet die in Windows installierten Druckertreiber auf.

Wenn Sie Alt+Pfeil ab drücken, wird eine Liste der Druckertreiber.

Dialogfenster Set Up (Drucker)

Das Dialogfenster Set Up (Drucker) enthält Druckeroptionen für den Drucker, den Sie im Dialogfenster **Select Printer** gewählt haben.

File|Exit (Datei/Beenden)

Der Befehl Exit (Beenden) beendet Turbo Pascal, entfernt es vollständig aus dem Speicher. Sollten Sie in einem Editor-Fenster Änderungen vorgenommen haben, ohne die Datei zu speichern, werden Sie gefragt, ob die Datei gespeichert werden soll.

Im CUA-Modus dient Alt+F4, im Alternate-Modus Alt+X zum Verlassen des Programms.

File|Liste geschlossener Dateien

Wenn Sie eine oder mehrere Dateien während einer Arbeitssitzung geöffnet und anschließend wieder geschlossen (mit Ctrl+F4) haben, werden bis zu fünf dieser geschlossenen Dateien am unteren Ende im Menü File (Datei) angezeigt. Sie können diese Dateien schnell wieder öffnen.

Wählen Sie den Dateinamen, um eine Datei erneut zu öffnen.

Dialogfenster Open (Öffnen)

Im Dialogfenster Open (Öffnen) geben Sie den Namen der zu öffnenden Datei an. Sie schreiben dazu den Namen in das Eingabefeld oder wählen einen Dateinamen aus der Dateinamensliste.

Das Eingabefeld File Name (Dateiname)

In dieses Eingabefeld tragen Sie den Namen der Datei ein, die geöffnet werden soll.

Das Listenfenster Files (Dateien)

In diesem Fenster werden die Namen der Dateien im aktuellen Verzeichnis angezeigt, die dem im Eingabefeld File Name (Dateinamen) angegebenen Muster entsprechen. Zudem werden hier die Namen der Unterverzeichnisse und des übergeordneten Verzeichnisses angezeigt.

Das Listenfenster Directories (Verzeichnisse)

Dieses Fenster enthält eine Verzeichnis- und Laufwerksliste. Sie können durch Auswahl eines Verzeichnisnamens in ein anderes Verzeichnis bzw. Laufwerk wechseln und sich dessen Inhalt ansehen.

Sie können auch mit Hilfe von Tastenkürzeln (Befehlsbuchstaben werden unterstrichen dargestellt) bestimmte Elemente des Dialogfensters auswählen.

Beispielweise wählen Sie mit Alt+D (bzw. V) den ersten Eintrag in der Verzeichnisliste aus.

Dialogfenster Save As (Speichern unter)

Im Dialogfenster Save As (Speichern unter) geben Sie den neuen Namen für die Datei an, die Sie unter einem anderen Namen abspeichern wollen.

Das Eingabefeld File Name (Dateiname)

Hier geben Sie den neuen Namen der Datei ein.

Dieses Eingabefeld ist mit einer **Eingabeaufzeichnungsliste** gekoppelt.

Das Listenfenster Directories (Verzeichnisse)

Sie können aus dieser Verzeichnisliste einen neuen Zugriffspfad (Verzeichnis bzw. Laufwerk) wählen. Mit Alt+D steuern Sie den Cursor in die Verzeichnisliste.

κDas Menü Help (Hilfe) (Alt+H)

Über das Menü Help (Hilfe) erhalten Sie Zugriff auf die Hilfefenster der Online-Hilfe. Es gibt zu fast allen Menüoptionen der integrierten Entwicklungsumgebung von Turbo Pascal Hilfeinformationen.
(Beachten Sie bitte auch die einzeilige Zusatzinformation, die immer nach Auswahl eines Befehls in der Status-Zeile erscheint.)
Nachstehend sind die verfügbaren Befehle des Menüs aufgeführt:

Index
Topic Search (Suche über Schlüsselwort)
Using Help (Hilfe über Hilfe)
Compiler Directives (Compiler-Befehle)
ObjectWindows
Procedures and Functions (Prozeduren und Funktionen)
Reserved Words (Reservierte Wörter)
Standard Units (Standard Units)
Turbo Pascal Language (Sprachspezifikation)
Windows API
About Turbo Pascal (Info über Turbo Pascal)

Help|Index (Hilfe/Index) (Shift+F1)

Der Befehl Index öffnet ein Dialogfenster mit allen verfügbaren Hilfe-Schlüsselwörtern (d.h. der Wörter, die im Hilfebildschirm hervorgehoben sind, und mit denen Sie schnell bestimmte Bildschirme anwählen können).

Help|Topic Search (Hilfe/Suche über Schlüsselwort) (Ctrl+F1)

Der Befehl Topic Search (Suche über Schlüsselwort) zeigt sprachbezogene Hilfe zum Wort an der aktuellen Cursorposition an.

Sie können die **rechte Maustaste** mit dem Dialogfenster Options|Preferences (Optionen/Vrogaben) so einstellen, daß damit ein sprachbezogener Hilfetext abgerufen wird.

Help|Using Help (Hilfe/Hilfe über Hilfe)

Der Befehl Help|Using Help (Hilfe/Hilfe über Hilfe) zeigt Ihnen Informationen zur Anwendung des Hilfesystems von Turbo Pascal.

About Turbo Pascal (Über Turbo Pascal)

Der Befehl Help|About Turbo Pascal (Hilfe/Über Turbo Pascal) zeigt ein Dialogfenster, in dem Informationen über Copyright und Version zu finden sind.

Mit Esc, OK oder Cancel schließen Sie das Dialogfenster.

Aktionsschalter OK

Mit Enter oder Anklicken des Aktionsschalters OK wird das Hilfefenster geschlossen; die Turbo Pascal-Umgebung wird wieder aktiv.

Help|Compiler Directives (Hilfe/Compiler-Befehle)

Bei Aufruf dieser Option sehen Sie eine Liste mit **Compiler-Befehlen.**

Compiler-Befehle steuern einige der Funktionen von Turbo Pascal. Manche bestimmen, wie der Compiler arbeitet, andere enthalten Parameter, die sich darauf auswirken, wie das Programm compiliert wird, oder wie die bedingte Compilierung abläuft.

Compiler-Befehle sind Kommentare mit einer besonderen Syntax, die überall im Programm stehen können.

Siehe auch

Index der Compiler-Befehle

Help|ObjectWindows

Bei dieser Menüoption sehen Sie einen Index zu ObjectWindows-Typen.

ObjectWindows ist eine objektorientierte Bibliothek aus Pascal-Klassen, die das Programmieren für Windows vereinfacht.

Siehe auch

Index der ObjectWindows-Objekte (alphabetisch)
ObjectWindows Objekthierarchy

Help|Procedures and Functions (Hilfe/Prozeduren und Funktionen)

Wenn Sie diese Option wählen, sehen Sie eine Liste aller Bibliotheksprozeduren und Funktionen von Turbo Pascal, die Sie in Ihre Programme einbinden können.

Siehe auch

Index derTurbo Pascal Prozeduren und Funktionen

Help|Reserved Words (Hilfe/Reservierte Wörter)

Diese Option zeigt eine Liste aller **reservierten Wörter** von Turbo Pascal.

Reservierte Wörter haben für den Compiler eine besondere Bedeutung und sollten nicht als Namen für Variablen, Prozeduren, Funktionen, Objekte etc. verwendet werden.

Help|Standard Units (Hilfe/Standard Units)

Diese Option zeigt eine Liste der **Standard-Units** von Turbo Pascal.

Eine Standard-Unit ist eine Sammlung vordefinierter Konstanten, Datentypen, Variablen, Prozeduren und Funktionen. Sie geben Ihrem Programm an, nur die Units zu verwenden, die es benötigt.

Siehe auch

Turbo Pascals Standard-Units

Help|Turbo Pascal Language (Hilfe/Sprachspezifikation)

Unter dieser Option sehen Sie eine Liste der Elemente, aus denen sich der Sprachumfang von Turbo Pascal zusammensetzt.

Siehe auch

Index des Sprachumfangs von Turbo Pascal

Help|Windows API (Hilfe/Windows API)

Diese Option zeigt Ihnen eine Liste der Konstanten, Stile, Meldungen, Typen, Funktionen und Prozeduren, welche die API (Windows Application Programming Interface, Windows-Programmierschnittstelle) bilden.

Turbo Pascal deklariert alle Stile, Konstanten und Typen für Windows in der Unit WinTypes. Die Unit WinProcs enthält alle Prozeduren und Funktionen der API.

Siehe auch

Index zu Turbo Pascals MS-Windows API

Das Menü Options (Optionen)

Das Menü Options (Optionen) erlaubt es Ihnen, viele der Standardeinstellungen von Turbo Pascal zu verändern. Die meisten Befehle in diesem Menü öffnen ein Dialogfenster.

Compiler (Compiler)
Linker (Linker)
Directories (Verzeichnisse)
Preferences (Vorgaben)
Open (Laden)
Save (Speichern)
Save As (Speichern unter)

Options|Compiler (Optionen/Compiler)

Der Befehl Options¦Compiler ruft das Dialogfenster **Compiler Options F(Compiler-Optionen)** auf, in dem Sie Einstellungen machen können, die die Codeerzeugung betreffen.

Das Dialogfenster Compiler Options (Compiler-Optionen)

Options (Optionen)

Diese Optionen legen fest, wie der Compiler bei der Codegenerierung arbeiten soll.

Range Checking (Bereichsüberprüfung)
Stack Checking (Stack-Prüfung)
I/O Checking (I/O-Prüfung)
Force Far Calls (Far-Aufrufe erzeugen)
Debug Information (Debug-Information)
Local Symbols (Lokale Symbole)
286 Code (286-Code)
80x87 Code (80x87-Code)
Windows Stack Frame (Windows Stack Frame)
Extended Syntax (Erweiterte Syntax)

Align Data (Datenausrichtung)

Diese Optionen (**Word** oder **Byte**) bestimmen, an welchen Adressen Variablen abgelegt werden.

String Var Checking (String-Prüfung)

Die Schaltfelder **Strict** (**Streng**) und **Relaxed** (**Locker**) setzen Optionen für die Fehlerprüfung bei Stringparametern.

Boolean Evaluation (Boolesche Auswertung)

Die Schaltfelder **Short Circuit** (**Kurzschluß**) und **Complete** (**Vollständig**) setzen Optionen für die Auswertung von Begriffen in Booleschen Ausdrücken.

Memory Sizes (Speichergrößen)

Die Eingabefelder **Stack Size** (**Stack**) und **Heap Size** (**Heap**) legen die Größe des Stacksegments und des lokalen Heap für lokale Fensterzuweisungen fest.

Conditional Defines (Definitionen für bedingte Compilierung)

In diesem Eingabefeld können Sie Symbole für eine bedingte Compilierung des Quelltextes festlegen.

Die Option Range Checking (Bereichsüberprüfung)

Dieser Schalter legt fest, ob der Compiler bei der Indizierung von Strings und Arrays sowie bei Zuweisungen auf skalare Typen und Unterbereichen eine Bereichsüberprüfung durchführt.

Scheitert die Überprüfung, stoppt das Programm mit einem Laufzeitfehler.
Wenn die Option nicht eingeschaltet ist, wird kein derartiger Code erzeugt.
Die Option Range Checking (Bereichsüberprüfung) entspricht **$R**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option Stack Checking (Stack-Prüfung)

Wenn diese Option eingestellt ist, erzeugt der Compiler zusätzlichen Code vor der Belegung des Stacks durch lokale Variablen.

Scheitert die Überprüfung, stoppt das Programm mit einem Laufzeitfehler.

Die Option Stack Checking (Stack-Prüfung) ist äquivalent zu **$S**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option I/O Checking (I/O-Prüfung)

Dieser Schalter legt fest, ob Ein- und Ausgaben zur Laufzeit des Programms automatisch auf Fehler überprüft werden.
Scheitert die Überprüfung, stoppt das Programm mit einem Laufzeitfehler.

Wenn die Option ausgeschaltet ist, wird kein derartiger Code generiert. Sie können jedoch I/O-Fehler mit der Systemfunktion **IOResult** überprüfen.

Die Option I/O Checking (I/O-Prüfung) entspricht **$I**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option Force Far Calls (Far-Aufrufe erzeugen)

Wenn der Schalter Force Far Calls (Far-Aufrufe erzeugen) gewählt wurde, werden die Adressen von Prozeduren und Funktionen als FAR-Zeiger abgespeichert (Segment und Offset).

Wenn die Option nicht eingestellt ist, verwendet der Compiler das Aufrufmodell NEAR (nur der Offset der Adresse der Routine wird übergeben).

Force Far Calls entspricht **$F**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option Debug Information

Dieser Schalter legt fest, daß der Compiler zusätzliche Informationen zur Fehlersuche erzeugt.

Diese Informationen bestehen aus einer Tabelle mit Zeilennummern für jede Routine, über die die Verbindung zwischen den Adressen der einzelnen Befehle und dem Quelltext hergestellt werden kann.

Normalerweise verwendet man diese Information zusammen mit der Option **local symbols (Lokale Symbole)**.

Sie müssen zusätzlich die Option Optionen/Linker/**Debug Info in Exe** einschalten, weil der Compiler sonst diese Einstellung ignoriert und Sie Ihr Programm nicht debuggen können.

Die Option Debug Information entspricht dem Compiler-Befehl **$D**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option Local Symbols (Lokale Symbole)

Wenn dieser Schalter eingestellt ist, dann erzeugt der Compiler zusätzliche Informationen über lokale Symbole in einem Modul.
Die Informationen über lokale Symbole bestehen aus:
- den Namen aller lokalen Variablen und Konstanten des Moduls (also auch der im Implementationsteil) und
- den Symbolen innerhalb der Prozeduren und Funktionen des Moduls.

Wenn ein Programm oder eine Unit mit diesen Informationen compiliert wurde, dann
- können Sie im integrierten Debugger auch den Inhalt der lokalen Variablen des Moduls untersuchen und ändern;
- können Sie Aufrufe der Prozeduren und Funktionen des Moduls im Fenster Call stack untersuchen.

Das Ausschalten dieser Option gibt mehr Speicher zum Compilieren frei. Sie haben dann keine lokalen Symbole für den Debugger.

Hinweis: Wenn die Optionen Options|Compiler|**Debug Information** und Options|Linker|**Debug Info in EXE** ausgeschaltet sind, dann ist diese Option ohne Wirkung.

Die Option Local Symbols (Lokale Symbole) entspricht dem Compiler-Befehl **$L**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option 286 Code

Wenn diese Option ausgeschaltet ist, generiert Turbo Pascal Code, der auch auf allen Prozessoren der 80x86-Familie lauffähig ist.

Wenn Sie diese Option ankreuzen, dann generiert Turbo Pascal spezifischen Code für die 80286-Prozessoren, das resultierende Programm ist dann jedoch nicht mit einem 8088 oder 8086 Prozessor lauffähig.

Hinweis: Programme, die mit 80286 Code-Erzeugung compiliert wurden, prüfen zur Laufzeit NICHT, ob auch ein 80286 vorhanden ist.

Die Option 286 Code entspricht dem Compiler-Befehl **$G**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option 80x87 Code

Dieser Schalter legt fest, wie Operationen mit Gleitkommazahlen erfolgen.

Wenn der Schalter eingeschaltet ist, generiert Turbo Pascal inline Code für 8087 oder 80287 Coprozessoren.

Lassen Sie die Option ausgeschaltet, wenn Turbo Pascal keinen 80x87-Code erstellen soll, dür den ein 80x87-Coprozessor erforderllich ist. (Voreinstellung = Ausgeschaltet)

Sie können alle Real-Datentypen mit dieser Option verwenden, wie
- Single
- Double
- Extended
- Comp

Siehe auch

Compiler Options dialog box

Die Option Windows Stack Frame

Veranlaßt den Compiler, für Programme, die im Real-Mode von Windows 3.0 laufen sollen, speziellen Prolog- und Epilog-Code zu erstellen.

Schalten Sie diese Option aus, wenn Ihre Programme nur im Protected-Mode laufen sollen.

Windows Stack Frame entspricht dem Compiler-Befehl **$W**.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Die Option Extended Syntax (Erweiterte Syntax)

Wenn diese Option eingeschaltet ist, wird die erweiterte Syntax von Turbo Pascal verwendet, es können also beispielsweise Funktionsaufrufe wie Anweisungen (also wie Prozeduren) verwendet werden.

Wenn die Option ausgeschaltet ist, ist diese Art der Aufrufe nicht möglich.

Die Option Extended Syntax entspricht dem Compiler-Befehl **$X**.

Siehe auch

Dialogfenster Compiler Options (Comopiler-Optionen)

Align Data (Datenausrichtung)

Die Auswahlfelder Align Data richten Daten, die keine Zeichen sind, an geraden oder ungeraden Adressen aus. Das Ausrichten von Daten auf Wortgrenzen erhöht die Zugriffsgeschwindigkeit auf diese Daten bei den 80x86 Prozessoren.

Align Data entspricht **$A**.

Word

Im Modus Word Align Data werden Variablen nur an geraden Adressen abgelegt. (gilt nicht für Char und Byte)

Byte

Im Modus Byte findet keine Ausrichtung statt - Variablen werden so dicht wie möglich gepackt.

Siehe auch

Compiler Options dialog box

String Var Checking (String-Prüfung)

Die String-Prüfungs-Optionen Strict (Streng) und Relaxed (Locker) lassen Sie zwischen strikter und gelockerter Prüfung von String-Parametern wählen.

String Var Checking entspricht dem Compiler-Befehl **$V**.

Strict (Streng)

Der Compiler setzt voraus, daß formale und aktuelle Parameter einen identischen Typ haben und weist jede Verletzung als Fehler aus.

Relaxed (Locker)

Jeder String kann als VAR-Parameter übergeben werden, selbst dann, wenn seine Länge nicht mit der des formalen Parameters übereinstimmt.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Boolean Evaluation (Boolesche Auswertung)

Mit diesen Schaltfeldern wählen Sie zwischen dem Kurzschlußverfahren und der vollständigen Auswertung von booleschen Ausdrücken.
Die Option entspricht **$B**.

Short Circuit (Kurzschluß)

Boolesche Ausdrücke werden nur soweit ausgewertet, bis sich am Gesamtergebnis nichts mehr ändern kann.

Zum Beispiel würde im Ausdruck

```
if False and MyFunc...
```

die Funktion MyFunc nie aufgerufen.

Complete (Vollständig)

Sämtliche Teile eines zusammengesetzten booleschen Ausdrucks werden ausgewertet, selbst wenn das Gesamtergebnis bereits feststeht.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Memory Sizes (Speichergrößen)

Diese Option besteht aus zwei Eingabefeldern, in denen Sie den Speicherbedarf Ihres Programmes angeben können.

Alle drei Speichereinstellungen können auch auch über den Compiler-Befehl **$M** im Quelltext angegeben werden.

Eingabefeld Stack Size

Hier können Sie die gewünschte Größe des Stacksegments in Bytes angeben.

Eingabefeld Heap

In dieser Eingabezeile geben Sie die Größe des lokalen Heaps in Bytes ein.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Das Eingabefeld Stack Size

Hier können Sie die gewünschte Größe des Stacksegments in Bytes angeben.

- Vorgabe = 8192 bytes
- Maximal = 65536 bytes

Durch Anklicken des Symbols Pfeil ab oder Betätigung der Tasten Alt+Pfeil ab lassen Sie die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Größen anzeigen.

Eingabefeld Heap Size

In diesem Eingabefeld geben Sie die Größe des Heaps in Bytes für lokale Fensterzuweisungen ein.

- Vorgabe = 0 Bytes

Wenn Sie versuchen, Ihr Programm mit zu kleinem Heap zu starten, bricht es mit einem Laufzeitfehler ab.

Anklicken des Symbols Pfeil ab oder die Betätigung der Tasten Alt+Pfeil ab zeigt die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Größen.

Das Eingabefeld Conditional Defines (Definitionen für bedingte Compilierung

In diesem Eingabefeld können Sie Symbole angeben, die in den Anweisungen zur bedingten Compilierung verwendet werden.

Mehrere Symbole können Sie jeweils durch ein Semikolon voneinander trennen:

```
TestCode;DebugCode
```

Anklicken des Symbols Pfeil ab oder Betätigung der Tasten Alt+Pfeil ab zeigt die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Symbolen.

Siehe auch

Dialogfenster Compiler Options (Compiler-Optionen)

Options|Linker (Optionen/Linker)

Mit diesem Befehl rufen Sie das Dialogfenster **Linker Options (Linker-Optionen)** auf, in dem Sie Einstellung am Turbo Pascal Linker vornehmen können.

Das Dialogfenster Linker Options (Linker-Optionen)

Das Dialogfenster Linker besteht aus zwei Gruppen von Auswahlfeldern sowie den bereits bekannten Schaltern OK, Cancel (Abbruch) und Help (Hilfe).

Map File (Map-Datei)

Diese Reihe von Auswahlfeldern erlaubt Ihnen anzugeben, welche Art von Map-Datei der Linker erzeugen soll, zur Auswahl stehen die Optionen **Off (Aus)**, **Segments (Segmente)**, **Publics** und **Detailed (Detailliert)**.

Link Buffer File (Link-Puffer)

Hier können Sie festlegen, wo der Linker seinen Zwischenspeicher anlegen soll, zur Auswahl stehen **Memory (Speicher)** oder **Disk (Diskette/Platte)**.

Debug Info in EXE

Diese Option integriert Debugger-Informationen in Ihr Programm.

Die Optionen Map File (Map-Datei)

Mit diesen Auswahlfeldern können Sie angeben, welche Art von Map-Datei der Linker erzeugen soll. Sofern Sie die Option nicht auf Off (Aus) eingestellt haben, wird die Map-Datei in dem Verzeichnis abgelegt, das im Dialogfenster Options|**Directories (Verzeichnisse)** benannt wurde.

- Die Map-Datei -Optionen liefern nur dann Debugger-Informationen, wenn das Modul mit der Option Options|Compiler|**Debug Information** compiliert wurde.

- Ebenso werden nur dann Informationen über lokale Variablen generiert, wenn das Modul mit der Option **Local Symbols (Lokale Symbole)** compiliet wurde.

Off (Aus)

Ist Off (Aus) eingeschaltet, wird keine Map-Datei erzeugt.

Segments (Segmente)

Die Map-Datei enthält eine Liste der Segmente, die Startadresse des Programms und alle Fehlermeldungen, die beim Linkvorgang erzeugt wurden. Dies entspricht dem Kommandozeilenparameter **/GS**.

Publics

Bei dieser Option wird zusätzlich zu den Informationen, die auch Segments erzeugt, noch eine alphabetisch sortierte Liste der PUBLIC- Symbole ausgegeben. Dies entspricht dem Kommandozeilenparameter **/GP**.

Detailed (Detailliert)

Bei eingestellter Option Detailed (Detailliert) erzeugt der Linker zusätzlich zu den Informationen, die bei Publics generiert werden, eine ausführliche Segmenttabelle.Dies entspricht dem Kommandozeilenparameter **/GD**.

Diese Segmenttabelle enthält die Adresse, die Länge in Bytes, den Segmentnamen und das Modul.

Siehe auch

Dialogfenster Linker

Die Optionen Link Buffer File (Link-Puffer)

Diese Option legt fest, ob Turbo Pascal zur Zwischenspeicherung von Linker-Daten den Hauptspeicher oder eine Disketten-Datei verwendet. Dies entspricht dem Kommandozeilenparameter **/L**.

Die Linker-Datei werden auf Diskette gehalten, der Hauptspeicher ist frei, aber der Linker braucht etwas länger.

Memory (Speicher)

Die Linker-Daten werden im Speicher gehalten, der Link-Vorgang ist schnell, aber bei großen Programm kann es passieren, daß der Speicher nicht ausreicht.

Disk (Diskette/Platte)

Die Linker-Datei werden auf der Festplatte gehalten, der Hauptspeicher ist frei, aber der Linker braucht etwas länger.

Siehe auch

Dialogfenster Linker

Die Option Debug Information in EXE

Dieser Schalter legt fest, daß der Compiler zusätzliche Informationen zur Fehlersuche erzeugt. Ihr Programm wird größer. Dies entspricht dem Kommandozeilenparameter **/V**.

Diese Option sollte immer eingeschaltet sein, wenn Sie ein Programm mit dem Turbo Debugger für Windows überprüfen.

Siehe auch

Dialogfenster Linker

Options|Directories (Optionen/Verzeichnisse)

Dieser Befehl öffnet das Dialogfenster **Directories (Verzeichnisse)**, das vier Eingabefelder enthält.

Das Dialogfenster Directories (Verzeichnisse)

In diesem Dialogfenster können Sie die Verzeichnisse angeben, in denen Turbo Pascal Dateien sucht bzw. ablegt.

EXE and TPU Directories (EXE/TPU-Verzeichnisse)

In dem hier angegebenen Verzeichnis werden die erzeugten TPU- und EXE-Dateien gespeichert.

Include Directories (Include-Verzeichnisse)

Dieses Eingabefeld enthält den Suchweg für die Include-Dateien.

Unit Directories (Unit-Verzeichnisse)

In diesem Verzeichnis werden die Units gesucht bzw. abgelegt.

Object Directories (Object-Verzeichnisse)

Dieses Eingabefeld enthält den Suchweg bzw. die Suchwege für OBJ-Dateien.

Resource Directories (Ressourcen-Verzeichnisse)

Hier wird das Verzeichnis für Ressourcen-Dateien angegeben.

Siehe auch

Eingabe von Verzeichnisnamen

Eingabe von Verzeichnisnamen

Bitte beachten Sie die folgenden Richtlinien bei der Eingabe von Suchwegen im Dialogfenster Options¦**Directories (Verzeichnisse)**.

- Falls Sie mehrere Verzeichnisse angeben (dürfen), so müssen sie durch einen Strichpunkt getrennt sein.
- Die Suchwege dürfen maximal 127 Zeichen lang sein.
- Leerzeichen vor und nach dem Strichpunkt sind zwar erlaubt, aber nicht notwendig.
- Es sind sowohl relative wie auch absolute Pfadangaben gestattet.

Beispiel:

```
C:;C:;A:TPW
C:\;C:\TPW;A:\TPW
```

Das Eingabefeld EXE and TPU Directory (EXE/TPU-Verzeichnis)

Dieses Eingabefeld enthält den Suchweg zu dem Directory, in dem Turbo Pascal erzeugte EXE- und TPU-Dateien speichert. Wenn hier nichts angegeben ist, wird das zum Zeitpunkt der Compilierung gesetzte Verzeichnis benutzt. Dies entspricht dem Kommandozeilenparameter **/E**.

Sollte eine **Map-Datei** erzeugt werden, dann wird sie ebenfalls in diesem Verzeichnis abgespeichert.

Anklicken des Symbols Pfeil ab oder die Betätigung der Tasten Alt+Pfeil ab zeigen die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Namen.

Siehe auch

Eingabe von Verzeichnisnamen
Dialogfenster Directories

Das Eingabefeld Include Directory (Include-Verzeichnisse)

Dieses Eingabefeld enthält den Suchweg bzw. die Suchwege für Include-Dateien, die über einen Compiler-Befehl aufgenommen werden.

Dies entspricht dem Kommandozeilenparameter **/I**.

Anklicken des Symbols Pfeil ab oder die Betätigung der Tasten Alt+Pfeil ab zeigen die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Namen.

Siehe auch

Eingabe von Verzeichnisnamen
Dialogfenster Directories (Verzeichnisse)

Das Eingabefeld Unit Directory (Unit-Verzeichnisse)

Dieses Eingabefeld enthält den Suchweg bzw. die Suchwege für Units. Dies entspricht dem Kommandozeilenparameter **/U**.

Anklicken des Symbols Pfeil ab oder die Betätigung derTasten Alt+Pfeil ab zeigen die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Namen.

Siehe auch

Eingabe von Verzeichnisnamen
Dialogfenster Directories (Verzeichnisse)

Das Eingabefeld Object Directory (Object-Verzeichnisse)

Dieses Eingabefeld enthält den Suchweg bzw. die Suchwege für OBJ-Dateien (Assembler-Routinen).

Dies entspricht dem Kommandozeilenparameter **/O**.

Anklicken des Symbols Pfeil ab oder die Betätigung der Tasten Alt+Pfeil ab zeigen die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Namlen.

Siehe auch

Eingabe von Verzeichnisnamen
Dialogfenster Directories (Verzeichnisse)

Das Eingabefeld Resource Directory (Ressourcen-Verzeichnisse)

Hier wird der Suchweg für Ressourcen-Dateien angegeben. Dies entspricht dem Kommandozeilenparameter **/R**.

Anklicken des Symbols Pfeil ab oder die Betätigung der Tasten Alt+Pfeil ab zeigen die **Eingabeaufzeichnungsliste** mit den bisher eingegebenen Namen.

Siehe auch

Eingabe von Verzeichnisnamen
Dialogfenster Directories (Verzeichnisse)

Options|Preferences (Optionen/Vorgaben)

Der Menübefehl Preferences (Vorgaben) öffnet das Dialogfenster **Preferences (Vorgaben)**.

Mit den Einstellungen in diesem Dialogfenster können Sie Turbo Pascal an Ihre Erfordernisse anpassen.

Das Dialogfenster Preferences (Vorgaben)

Mit den Einstellungen in diesem Dialogfenster können Sie Turbo Pascal an Ihre Erfordernisse anpassen.

Editor Options (Editor-Optionen)

Hier finden Sie Optionen, die die Textverarbeitung in Editorfenstern bestimmen.

Create Backup File (Backup-Datei anlegen)
Auto Indent Mode (Automatischer Zeileneinzug)
Use Tab Character (Einzug mit Tab)
Optimal Fill (Füllen mit Tabs)
Backspace Unindents (Backspace löscht Einzug)
Cursor through Tabs (Cursor-Modus durch Tabs)
Group Undo (Befehlsgruppe widerrufen)
Block Overwrite (Block überschreiben)

Auto Save Options (Automatisches Speichern)

Diese Optionen legen fest, wann und wie oft Informationen über Ihre offenen Dateien, Einstellungen und Optionen gesichert werden.

Editor Files (Dateien im Editor)
Desktop (Desktop)
Configuration (Konfiguration)

Font (Schrift)

In diesem Listenfenster können Sie die Schriften und Schriftgrößen auswählen, die auf dem Bildschirm erscheinen.

Tab Size (Tab-Weite)

Hier wird festgelegt, wie viele Spalten weit der Cursor mit der Tabulatortaste bewegt wird.

Right Mouse Button (Rechte Maustaste)

Die Optionen **Nothing (Keine Wirkung)** oder **Topic Search (Schlüsselwort)** für die rechte Maustaste bestimmen, ob mit ihr nach bestimmten Hilfetexten gesucht werden kann.

Command Set (Befehlssatz)

Die Befehlsoptionen (**CUA** (Common User Access) oder **Alternate (Alternativ)** bestimmen, welche Befehle der Editor akzeptiert.

Die Option Create Backup Files (Backup-Dateien anlegen)

Wann immer Sie eine Datei mit File|**Save (Datei/Speichern)** abspeichern, sichert die Entwicklungsumgebung die vorletzte Version, indem die Datei in .BAK umbenannt wird.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Auto Indent Mode (Automatischer Zeileneinzug)

Wenn Sie im Editor Enter drücken, wird der Cursor automatisch unter das erste Zeichen der vorhergehenden Zeile gesetzt.

Dies erleichtert die Gliederung von Quelltexten in Einrückungsebenen.

Sie können die Einstellung **Optimal Fill (Füllen mit Tabs)** zusammen mit dem Einrücken-Modus verwenden.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Use Tab Character (Einzug mit Tab)

Turbo Pascal fügt das Zeichen Tab (ASCII 9) in den Text ein, wenn Sie die Tabulator-Taste drücken.

Ist diese Option ausgeschaltet, fügt Turbo Pascal stattdessen eine entsprechende Anzahl Leerzeichen ein.

Die Einstellung **Tab Size (Tab-Weite)** in den Editor-Optionen legt fest, wieviele Leerzeichen einem Tabulator entsprechen.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Optimal Fill (Füllen mit Tabs)

Jede einzurückende Zeile wird mit einem Minimum an Zeichen gefüllt, d.h. Tabs und Leerzeichen werden verwendet. Die so entstehenden Zeilen enthalten weniger Zeichen, als wenn diese Option nicht angewählt ist.

Ein Beispiel: Wenn **Tab Size (Tab-Weite)** auf 3 gesetzt ist, und Sie diese Zeilen eingeben:

```
begin
   if A <= B then
      DoSomething
```

sehen die entstehenden Zeilen so aus (Leerzeichen werden als "b" dargestellt und Tabs als "T"):

Ohne Optimal Fill	**Mit Optimal Fill**
`begin`	`begin`
`...if A <= B then`	`^if A <= B then`
`..DoSomething`	`^^DoSomething`

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Backspace Unindents (Backspace löscht Einzug)

Wenn diese Option eingeschaltet ist (die Voreinstellung), können Sie eine eingerückte Zeile durch Drücken von BackSpace wieder um eine Stufe "ausrücken".

Ist sie ausgeschaltet, wird der Cursor durch BackSpace nur um ein Zeichen nach links bewegt.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Cursor through Tabs (Cursor-Modus durch Tabs)

[X] Der Cursor bewegt sich jeweils um ein Zeichen, selbst wenn sich Tabs im Text befinden.
[ ] Der Cursor springt um mehrere Zeichen, wenn er auf ein Tab im Text trifft.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Group Undo Option (Befehlsgruppe widerrufen)

Ist diese Option eingeschaltet, wird beim Drücken von Alt+Backspace oder Wahl von Edit|**Undo (Bearbeiten/Rückgängig)** die letzte Gruppe der Editierbefehle rückgängig gemacht.

Ist die Option ausgeschaltet, wird nur die letzte Änderung aufgehoben.

Einfügen, Löschen, Überschreiben und Cursorbewegungen können gruppiert werden. Der Turbo Pascal Editor interpretiert Enter als Einfügung, die von einer Cursorbewegung gefolgt wird. Mit Enter endet eine Gruppe von Editierbefehlen, weil die Art der Änderung wechselt (Einfügen zu Cursorbewegung).

Diese Option wirkt sich auf Edit|**Redo (Bearbeiten/Widerrufen)** aus.

Beispiel

Ist Group Undo eingeschaltet, und Sie tippen OOPS und wählen Undo, wird das gesamte Wort gelöscht. Wählen Sie danach Redo, erscheint die ganze Gruppe OOPS wieder auf dem Bildschirm.

Ist Group Undo ausgeschaltet, wird nur das S mit Undo gelöscht. Um das Wort zu löschen, müssen Sie viermal Undo wählen. Mit Redo taucht ebenfalls nur ein Buchstabe auf.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die OptionBlock Overwrite (Block überschreiben)

Wenn diese Option eingeschaltet und ein Textblock markiert ist, überschreibt ein Buchstabe den gesamten Block.

Ist die Option ausgeschaltet, wird der Buchstabe nach dem markierten Text eingefügt.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Auto Save (Automatisches Speichern)

Diese Optionen legen fest, wann und wie oft Informationen über Ihre offenen Dateien, Einstellungen und Optionen gespeichert werden.

Editor Files (Editor-Dateien)

Turbo Pascal speichert den Inhalt des aktuellen Editorfensters ab, wenn Sie einen der folgenden Befehle ausführen:
Run|**Run (Ausführen/Ausführen)** oder Run|**Debugger(Ausführen/Debugger)**.

Desktop

Die aktuellen Fenstereinstellungen, die Eingabeaufzeichnungslisten etc. werden in einer Datei TURBO.DSK gesichert, sobald Sie Turbo Pascal verlassen, und wiederhergestellt, wenn Sie es starten.

Configuration (Konfiguration)

Wenn diese Option eingeschaltet ist, speichert Turbo Pascal den Namen der Hauptdatei (Primary file) und alle Einstellungen aus dem Menü Options (Optionen), wenn Sie das Programm verlassen.
Bei Beginn der nächsten Arbeitssitzung sind die gesicherten Einstellungen in Kraft.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Editor Files (Editor-Dateien)

Turbo Pascal sichert den Inhalt des aktuellen Edit-Fensters, wenn Sie einen der folgenden Befehle geben:
Run|**Run (Ausführen/Ausführen)** oder Run|**Debugger (Ausführen/Debugger)**.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die OptionDesktop

Die aktuellen Fenstereinstellungen, die Eingabeaufzeichnungslisten etc. werden in einer Datei TURBO.DSK gesichert, sobald Sie Turbo Pascal verlassen, und wiederhergestellt, wenn Sie es starten.

Die aktuellen Fenstereinstellungen werden auch dann gespeichert, wenn Sie zu einer neuen Konfigurationsdatei umschalten, und stehen nachher wieder zur Verfügung.

Die Einstellungen stehen in der Datei **TPW.INI**.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Configuration (Konfiguration)

Wenn die Option eingeschaltet ist, speichert Turbo Pascal den Namen der Hauptdatei (Primary file) und alle Einstellungen aus dem Menü Options (Optionen), wenn Sie das Programm verlassen.

Beim Start einer neuen Sitzung gelten alle gespeicherten Optionen.

Configuration weist die IDE an, die Datei **.CFG** im Verzeichnis der Hauptdatei oder in demselben Verzeichnis wie die Datei des aktiven Editorfensters zu speichern.

Manuell können Sie eine Konfigurationsdatei mit Options|**Open (Optionen/Laden)** erzeugen.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Das Listenfenster Fonts (Schrift)

Größe und Aussehen von Text können Sie durch die Wahl einer anderen Schrift im Listenfenster Font (Schrift) ändern.

Windows liefert einige Bildschirmschriften, und bei der Installation Ihres Druckers wurden auch alle Schriften, die der Drucker verwendet, installiert.

Sie können dazu weitere Schriften für Drucker oder Bildschirm erwerben.

Turbo Pascal verwendet nur monospace-Schriften wie Courier.

Die neu gewählte Schrift wird erst durch Eingabe von OK wirksam.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Das Eingabefeld Tab Size (Tab-Weite

Hier können Sie die Tabulatorschrittweite festlegen. Mögliche Werte liegen im Bereich 2 bis 16, die Standardvorgabe ist 8.

Bei einer neuen Tabulatorgröße zeigt Turbo Pascal alle Tabulatoren der aktuellen Datei in der neuen Größe.

Mit Options|**Save (Optionen/Speichern)** können Sie die neue Einstellung speichern.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Right Mouse Button (Rechte Maustaste)

Als Vorgabe verwendet Turbo Pascal die rechte Maustaste für die sprachbezogene Hilfefunktion.

Topic Search (Schlüsselwort)

Wenn die Option Topic Search (Schlüsselwort) eingeschaltet ist, erscheint beim Drücken der rechten Maustaste ein Hilfebildschirm, in dem sprachbezogene Hilfetexte zu dem Begriff geboten wird, auf dem der Cursor gerade steht.

Nothing (Keine Wirkung)

Wollen Sie diese Hilfe nicht, anzeigen lassen, wählen Sie Nothing (Keine Wirkung).

Siehe auch

Dialogfenster Preferences (Vorgaben)

Die Option Command Set (Befehlssatz)

Hier wird festgelegt, ob der Editor Common User Access (CUA) oder Alternate-Befehle verwendet.

CUA

Common User Access-Befehle werden von Windows-Editoren benutzt.

Alternate (Alternativ)

Alternativ-Befehle dienen für Editoren in anderen Borland Sprachprodukten.

Siehe auch

Dialogfenster Preferences (Vorgaben)

Options|Open (Optionen/Laden)

Dieser Befehl ruft das Dialogfenster **Open Configuration File (Konfigurationsdatei laden)** auf, in dem abweichende Konfigurationen gewählt werden können.

Das Dialogfenster Open Configuration File (Konfigurationsdatei laden)

In diesem Dialogfenster können Sie neue Einstellungen für Konfigurationsdateien treffen.

File Name (Dateiname)

Hier geben Sie einen Dateinamen der gewünschten Konfigurationsdatei ein.

Vorgabe ist TPW.CFG.

Files (Dateien)

Das Listenfenster Files (Dateien) zeigt alle Dateien, die zur Maske im Eingabefeld passen, und zusätzlich alle entsprechenden Dateien des übergeordneten Verzeichnisses und der Unterverzeichnisse.

Directories (Verzeichnisse)

Aus dieser Liste können Sie ein anderes Verzeichnis wählen. Mit Alt + D steuern Sie den Cursor in dieses Listenfenster.

Options|Save (Optionen/Speichern)

Dieser Befehl ruft das Dialogfenster Save Options (Konfiguration speichern) auf, mit dem Sie die Desktop-Einstellungen und alle Einstellungen sichern können, die Sie im Menü Options (Optionen) vorgenommen haben.

Mit dem Befehl Options|**Save As (Optionen/Speichern unter)** können Sie die Einstellungen in einer anderen Konfigurationsdatei abspeichern.

- Alle Optionen und die Befehlstabelle des Editors werden in **TURBO.CFG**, der Vorgabedatei, gespeichert.

- Eingabeaufzeichnungslisten, der Status des Desktops und die Positionen von Breakpoints werden in **TURBO.INI**, der Vorgabe-Desktop-Datei, gespeichert.

Turbo Pascal sucht nach diesen Dateien zuerst im aktuellen Verzeichnis und dann in dem Verzeichnis, in dem sich TURBO.EXE befindet.

Options|Save As (Optionen/Speichern unter)

Dieser Befehl öffnet das Dialogfenster **Configuration Save As (Konfiguration speichern unter)**, in dem der Name Ihrer Konfigurationsdatei angegeben wird.

Wenn Sie eine neue Konfigurationsdatei anlegen wollen, führen Sie die gewünschten Einstellungen aus und wählen den Befehl Options|Save As (Optionen/Speichern unter). Geben Sie den neuen Dateinamen ein und wählen Sie OK. Beim nächsten Aufruf von Turbo Pascal gelten die Einstellungen der neuen Konfiguationsdatei.

Das Dialogfenster Configuration Save As (Konfiguration speichern unter)

In diesem Dialogfenster wird der Name der Konfigurationsdatei angegeben.

File Name (Dateiname)

In diesem Eingabefeld geben Sie den Namen der Konfigurationsdatei an Turbo Pascal fügt die Dateinamenserweiterung .CFG an. Beispielsweise wird MEIN dabei zu MEIN.CFG.

Zu diesem Eingabefeld gehört auch eine **Eingabeaufzeichnungsliste**.

Wenn die im Dialogfenster Preferences (Vorgaben) bei Auto Save (Automatisches Speichern) die Option **Configuration (Konfiguration)** eingeschaltet ist, werden Änderungen beim Verlassen des Programms gespeichert.

Directories (Pfad)

Mit Hilfe dieses Listenfensters können Sie Dateien in anderen Verzeichnissen öffnen. Alt+D steuert den Cursor in dieses Listenfenster.

Das Menü Run (Ausführen)

Von diesem Menü aus können Sie Ihr Programm compilieren, linken und im Debug-Modus starten. Wählen Sie einen der folgenden Befehle:

Run (Ausführen)
Debugger (Debugger)
Parameters (Parameter)

Wenn Sie den Debugger einsetzen wollen, sollten Sie vor dem Compilieren und Linken Ihres Programms das Markierungsfeld Options|Linker|**Debug Info in EXE** markieren.

Die Option Debug Info in EXE fügt die nötigen Informationen in Ihre ausführbare Datei ein.

Run|Run (Ausführen/Ausführen)

Dieser Befehl startet Ihr Programm und zwar mit den Parametern, die Sie mit Run¦**Parameters** angegeben haben.

Falls der Quelltext seit der letzten Compilierung geändert wurde, führt Turbo Pascal den Befehl Make (Projekt aktualisieren) aus.

Run|Debugger (Ausführen/Debugger)

Dieser Befehl startet den Turbo Debugger für Windows.

Turbo Pascal teilt dem Turbo Debugger mit, welches Programm getestet werden soll.

Zum Debuggen muß Ihr Programm mit eingestellter Option Options|Linker|**Debug Info in EXE** compiliert und gelinkt worden sein. Die Option Debug Info in EXE fügt die nötigen Informationen in Ihre ausführbare Datei ein.

Um die symbolischen Debug-Möglichkeiten des Turbo Debugger ganz nutzen zu können, sollten Sie auch die Optionen Options|Compiler|**Debug Information** und Options|Linker|**Local Symbols (Lokale Symbole)** vor dem Compilieren einschalten.

Run|Parameters (Ausführen/Parameter)

Mit diesem Befehl können Sie Parameter an das Programm übergeben, genauso als ob Sie es aus dem Menü File|Run (Datei/Ausführen) des Programm-Managers gestartet hätten.

Nach Eingabe des Befehls wird das Dialogfenster **Parameters (Parameter)** angezeigt, in welchem Sie die beim Start des Programms gewünschte Parameterliste eingeben können.

Das Dialogfenster Parameters (Parameter)

Hier geben Sie Kommandozeilenparameter für Programme ein.

Command Line Parameters (Kommandozeilenparameter)
In diesem Eingabefeld geben Sie die neuen Parameter ein, die Turbo Pascal an Ihr Programm weitergeben soll.

Das Eingabefeld Command-Line Parameters (Neue Kommandozeilenparameter)

Geben Sie hier die Parameter ein, die Turbo Pascal an Ihr laufendes Programm weitergeben soll, nicht aber den Programmnamen selbst.

Die Parameter sind nur beim Programmstart von Bedeutung.

Sie können auf Parameter, die Sie schon früher eingegeben haben, über die **Eingabeaufzeichnungsliste** zugreifen.

Eine Liste der Kommandozeilenparameter finden Sie im Hilfebildschirm **Command-Line Options (Kommandozeilenoptionen)**.

Das Menü Search (Suchen)

Im Menü Search (Suchen) finden Sie Funktionen, mit denen Sie Texte, Deklarationen von Funktionen und Fehlermeldungen in der Datei suchen können. Folgende Befehle enthält das Menü:

Find (Suchen nach)
Replace (Ersetzen)
Search Again (Weitersuchen)
Go to Line Number (Gehe zu Zeile)
Show Last Compile Error (Letzter Compilerfehler)
Find Error (Laufzeitfehler suchen)

Search|Find (Suchen/Suchen nach)

Dieser Befehl zeigt das Dialogfenster **Find Text (Suchen nach Text)** an. Geben Sie den Begriff, nach dem Sie suchen möchten, ein. Weiterhin können Sie mehrere Optionen angeben, die die Suche beeinflussen.

Das Dialogfenster Find Text (Suchen nach Text)

In diesem Dialogfenster geben Sie den Text ein, nach dem gesucht werden soll.

Text to Find (Suchen nach)
In dieses Eingabefeld tragen Sie den Begriff ein, nach dem Sie suchen möchten, betätigen Sie OK zum Starten der Suche oder Cancel (Abbruch) zum Abbruch derselben.

Wenn Sie die Taste Pfeil ab drücken, bekommen Sie die **Eingabeaufzeichnungsliste** für das Eingabefeld.

Optionen für Find Text (Suchen nach Text)
Diese Auswahlfelder bestimmen, nach welchen Strings Turbo Pascal sucht:

Case Sensitive (Groß-/Kleinschreibung)
Whole Words Only (Nur ganze Wörter)
Regular Expression (Reguläre Ausdrücke)

Direction (Richtung)
In diesem Wahlfeld legen Sie die Richtung fest, in die Turbo Pascal sucht. Zur Auswahl stehen Forward (Vorwärts) und Backward (Rückwärts).
Die Voreinstellung lautet **Forward (Vorwärts)** ab der aktuellen Cursorposition.

Scope (Bereich)
Mit diesem Schalter stellen Sie ein, welcher Bereich der Datei durchsucht werden soll. Voreinstellung ist **global**, d.h. die ganze Datei.

Origin (Beginn)
In diesem Wahlfeld legen Sie fest, an welcher Stelle die Suche beginnen soll.

Das Eingabefeld Text to Find (Suchen nach)

Geben Sie den Such-String in das Eingabefeld ein. Starten Sie die Suche mit OK oder löschen Sie die Eingabe mit Cancel (Abbruch).

Wenn Sie einen String eingeben wollen, den Sie vorher schon einmal gesucht haben, drücken Sie die Taste Pfeil ab, um sich die **Eingabeaufzeichnungsliste** anzeigen zu lassen, und wählen Sie den String aus dieser Liste.

Wenn das gesuchte Wort im Editorfenster sichtbar ist, können Sie den Cursor darauf setzen. Turbo Pascal verwendet das Wort, auf dem der Cursor steht als Vorgabe.

Die Option Case Sensitive (Groß-/Kleinschreibung)

Markieren Sie diese Option, wenn Turbo Pascal zwischen Groß- und Kleinbuchstaben unterscheiden soll.

Vorgabegemäß ist diese diese Option ausgeschaltet.

Die Option Whole Words Only (Nur ganze Wörter)

Wählen Sie diese Option, wenn Sie nur nach ganzen Wörtern suchen (d.h. der String muß auf beiden Seiten von Interpunktionszeichen oder Whitespace eingeschlossen sein). Wenn diese Option ausgeschaltet ist, wird der Suchstring auch als Teilsausdruck innerhalb längerer Wörter gesucht.

Die Option Regular Expression (Reguläre Ausdrücke)

Diese Option markieren Sie, wenn Turbo Pascal **GREP-ähnliche Jokerzeichen** im Such-String erkennen soll.

GREP-ähnliche Jokerzeichen

^ Ein Zirkumflex am Anfang eines Strings gibt den Anfang einer Zeile an.
$ Ein Dollarzeichen am Ende eines Ausdrucks gibt das Zeilenende an.
. Ein Punkt steht für jedes beliebige Zeichen.
* Ein Stern nach einem Zeichen steht für eine beliebige Anzahl von Vorkommen (einschl. Null) dieses Zeichens, z.B. steht bo* für bot, b, boo und beer b.r b, o oder t.
+ Ein Pluszeichen nach einem Zeichen steht für eine beliebige Anzahl von Vorkommen (größer Null) dieses Zeichens, z.B steht bo+ für bot und boo, aber nicht für b.
[ ] Zeichen in eckigen Klammern stehen für jedes Zeichen innerhalb der Klammern, aber für kein anderes.
[^] Ein Zirkumflex am Anfang eines Strings, der in eckigen Klammern steht, bedeutet **nicht**, [^bot] findet also alle Zeichen außer b, o und t.
[-] Ein Bindestrich in eckigen Klammern kennzeichnet einen Zeichenbereich.
\ Ein Backslash vor einem Ersetzungszeichen läßt Turbo Pascal dieses Zeichen buchstäblich und nicht als Ersetzungszeichen ansehen. Beispielsweise steht \^ für ^ und nicht für den Zeilenanfang.
к Direction Option

$KDirection (Richtung)

Diese Optionen gibt die Suchrichtung an, in der Turbo Pascal suchen soll, wobei bei der in der Einstellung **Origin (Beginn)** gegebenen Position begonnen wird.

Forward (Vorwärts)

Der Schalter **Forward** bewirkt, daß die Suche ab Cursorposition bis zum Dateiende erfolgt. **Forward (Vorwärts)** ist die Vorgabeeinstellung.

Backward (Rückwärts)

Der Schalter **Backward (Rückwärts)** bewirkt, daß die Suche ab der atuellen Cursorposition rückwärts bis zum Dateianfang erfolgt.

Scope (Bereich)

Diese Option bestimmt, welcher Bereich der Datei durchsucht werden soll.

Global (Gesamter Text)

Die Option **Global (Gesamter Text)** durchsucht die gesamte Datei in der in **Direction (Richtung)** eingestellten Richtung. Global wird als Vorgabe verwendet.

Selected Text (Markierter Text)

Die Option **Selected Text (Markierter Text)** läßt nur die Suche innerhalb des markierten Textblocks zu.

Sie markieren mit der Maus oder mit den **Blockbefehlen** einen Textblock.

Origin (Beginn)

Die Optionen für Origin (Beginn) bestimmen den Anfangspunkt der Suche.

From Cursor (Ab Cursor)

Diese Option entspricht der Voreinstellung. Die Suche wird hier bei der aktuellen Cursorposition begonnen.
Es wird dann entweder vorwärts oder rückwärts gesucht, abhängig von der Einstellung der Option **Direction (Richtung)**.

Entire Scope (Textanfang)

Entire Scope sucht entweder im selektierten Block oder in der gesamten Datei (unabhängig von der Cursorposition). Der Bereich, der durchsucht werden soll, wird mit der Option **Scope (Bereich)** definiert.

Search|Replace (Suchen/Ersetzen)

Dieser Befehl öffnet das Dialogfenster **Replace Text (Text ersetzen)**, in das Sie den Text, der gesucht, und den Text, der den gefundenen Begriff ersetzen soll, eingeben können.

Das Dialogfenster Replace Text (Text ersetzen)

In diesem Dialogfenster geben Sie an, welcher Texteintrag gesucht und durch was dieser ersetzt werden soll.
Die meisten Elemente dieses Dialogfensters entsprechen denen im **Find Text (Text suchen)**-Dialogfenster.

Text to Find (Suchen nach)

In dieses Eingabefeld tragen Sie den Such-String ein. Starten Sie die Suche mit OK oder Replace All (Alles ersetzen). Cancel (Abbruch) bricht den Vorgang ab.

Drücken Sie die Pfeiltaste Ab, um eine **Eingabeaufzeichnugsliste** mit allen Begriffen zu erhalten, die Sie vormals angegeben hatten.

New Text (Ersetzen durch)

In dieses Eingabefeld tragen Sie den String ein, durch den der gesuchte String ersetzt werden soll.

Sie können auch hier auf eine Eingabeaufzeichnungsliste zugreifen und früher verwendete Strings auswählen.

Optionen für Find Text (Suchen nach)

Diese Auswahlfelder bestimmen, nach welchen Strings Turbo Pascal sucht, und ob automatisch ersetzt wird:

- **Case sensitive (Groß-/Kleinschreibung)**
- **Whole words only (Nur ganze Worte)**
- **Regular expression (Reguläre Ausdrücke)**
- **Prompt to Replace (Mit Bestätigung)**

Direction (Richtung)

Über diese Optionen geben Sie an, in welche Richtung gesucht werden soll. Den Beginn der Suche legen Sie mit der Einstellung für Origin (Beginn) fest.
Gemäß Voreinstellung wird die Suche nach vorwärts (**Forward**) gerichtet ausgeführt..

Scope (Bereich)

Mit diesem Schalter stellen Sie ein, welcher Bereich der Datei durchsucht werden soll. Voreinstellung ist **Global**, d.h. die gesamte Datei.

Origin (Beginn)

Die Option Origin (Beginn) legt fest, wo in der Datei die Suche beginnen soll.

Replace All (Alles ersetzen)

Bei Auswahl von Replace All (Alles ersetzen) ersetzt Turbo Pascal jedes Vorkommen des Suchstrings, die Operation wird gemäß der Einstellungen für Direction (Richtung), Scope (Bereich) und Origin (Beginn) ausgeführt.

Das Eingabefeld New Text (Ersetzen durch)

In diese Eingabezeile geben Sie den Text ein, der den **Suchstring** ersetzen soll. (Es steht Ihnen auch eine **Eingabeaufzeichnugsliste** mit früheren Ersatz-Strings zur Verfügung)

Die Option Prompt on Replace (Mit Bestätigung)

Markieren Sie diese Option, wenn Sie wollen, daß Sie bei jeder Fundstelle des Such-Strings gefragt werden, ob ein Austausch vorgenommen werden soll.

Wenn diese Option ausgeschaltet ist, ersetzt Turbo Pascal automatisch den Such-String.

Replace All (Alles Ersetzen)

Nach der Eingabe des Suchstrings und des Ersatzstrings können Sie mit den Schaltern OK oder Replace All (Alles ersetzen) die Suche beginnen oder mit Cancel (Abbrechen) abbrechen.

Wird der Suchbegriff gefunden, erscheint eine Abfrage, ob der Begriff ersetzt werden soll - wenn Sie Replace All (Alles ersetzen) gewählt haben, erscheint die Abfrage bei jeder Fundstelle gefragt.

Vorher eingegebene Suchbegriffe finden Sie mit der Pfeiltaste Ab in einer **Eingabeaufzeichnugsliste**.

Search|Search Again (Suchen/Weitersuchen)

Dieser Befehl wiederholt den letzten **Find (Suchen nach)oder Replace (Ersetzen)**-Befehl. Alle Einstellungen, die Sie in den Dialogfenstern Find (Suchen nach Text) und Replace (Text ersetzen) vorgenommen haben, werden bei der Ausführung des Befehls verwendet.

Im CUA-Modus drücken Sie F3, im Alternate-Modus Ctrl+L für Search Again (Weitersuchen).

Search|Go to Line Number (Suchen/Gehe zu Zeile)

Der Befehl öffnet das Dialogfenster **Go To Line Number (Gehe zu Zeile)**, in das Sie die Zeilennummer, die gesucht werden soll, eingeben können.

Turbo Pascal zeigt die aktuelle Zeilen- und Spaltennummer in der unteren linken Ecke jedes Editorfensters an.

Das Dialogfenster Go to Line Number (Gehe zu Zeile)

Das Dialogfenster besteht aus einem Eingabefeld mit einer Eingabeaufzeichnungsliste sowie den Standardschaltern OK, Cancel (Abbruch) und Help (Hilfe)

Benutzen Sie dieses Dialogfenster, um den Cursor in eine bestimmte Zeilennnumer in Ihrer Datei zu steuern.

Enter New Line Number (Neue Zeilennummer)

Geben Sie die gewünschte Zeilennummer ein, oder wählen Sie eine früher eingegebene aus der **Eingabeaufzeichnungsliste**. Drücken Sie anschließend Enter oder wählen Sie OK.

Das Eingabefeld Enter New Line Number (Neue Zeilennummer)

Geben Sie die gewünschte Zeilennummer ein, oder wählen Sie eine früher eingegebene aus der **Eingabeaufzeichnungsliste**.

Drücken Sie anschließend Enter oder wählen Sie OK.

Die aktuelle Zeilen- und Spaltennummer wird in der unteren linken Ecke des Turbo-Pascal-Desktop angezeigt.

Search|Show Last Compile Error (Suchen/Letzter Compilerfehler)

Dieser Befehl findet die Zeile, in der der letzte Fehler beim Compilieren auftrat. Der Cursor wird in die betreffende Zeile gesetzt.

Falls der Fehler nicht in der Datei im aktiven Fenster ist, aktiviert Turbo Pascal das Fenster mit dem letzten Compilierungsfehler und öffnet die betreffende geschlossene Datei, wenn dies nötig ist.

Auf der Statuszeile werden Fehlernummer und -meldung angezeigt.

Search|Find Error (Suchen/Laufzeitfehler suchen)

Dieser Befehl führt zum Dialogfenster **Find Error (Fehler suchen)**, über das Turbo Pascal Laufzeitfehler lokalisiert.

Das Dialogfenster Find Error (Fehler suchen)

In diesem Dialogfenster geben Sie die Adresse des zuletzt aufgetretenen Laufzeitfehlers an.

Error Address (Adresse des Fehlers)

In dieses Eingabefeld tragen Sie die Adresse des zuletzt aufgetretenen Laufzeitfehlers ein.

Wenn Sie OK wählen, compiliert der Compiler das Programm neu und hält an der angegebenen Adresse an. Die Zeile, die den Laufzeitfehler verursacht hat, wird dann hervorgehoben angezeigt.

Das Eingabefeld Error Address (Adresse des Fehlers)

Hier geben Sie die gesuchte Zeilennummer des Fehlers ein, oder Sie wählen eine früher eingegebene aus der Eingabeaufzeichnungsliste.

Sie können eine bestimmte Adresse eingeben und danach suchen.

Das Menü Window (Fenster)

Im diesem Menü finden Sie Befehle zum Verwalten der Fenster.

Die meisten Fenster verfügen über Standardfenster-Elemente wie Bildlaufleisten, ein Schließfeld und Zoomfelder zum Verkleindern und Vergrößern.

Sie öffnen Fenster mit den Befehlen File|**Open (Datei/Öffnen)** oder File|**New (Datei/Neu)**.

Am unteren Ende des Menüs wird eine Liste mit allen geöffneten Fenstern angezeigt. Gibt es mehr als ein offenes Fenster, können Sie zu einem anderen Fenster wechseln. Dieses Fenster wird dann zum aktuellen Fenster.

Es gibt folgende Fensterbefehle:

Tile (Nebeneinander)
Cascade (Überlappend)
Arrange Icons (Symbole anordnen)
Close All (Alle schließen)

Window|Tile (Fenster/Nebeneinander)

Benutzen Sie diesen Befehl, um alle offenen Fenster nebeneinander anzeigen zu lassen.

Window|Cascade (Fenster/Hintereinander)

Mit diesem Befehl stapeln Sie alle geöffneten Fenster - jeweils um eine Zeile versetzt - übereinander.

Window|Arrange Icons (Fenster/Symbole anordnen)

Mit diesem Befehl ordnen Sie die Symbole so auf dem Turbo-Pascal-Desktop an, daß diese beginnend an der unteren linken Bildschirmecke in gleichem Abstand nebeneinander verteilt werden.

Offene Dateien müssen auf Symbolgröße **verkleinert** sein, sonst ist dieser Befehl unwirksam.

Window|Close All (Fenster/Alle schließen)

Der Befehl Close All (Alle schließen) schließt alle offenen Fenster des Desktop.

Wurde seit dem letzten Speichern Text geändert, werden Sie in einem Dialogfenster gefragt, ob die Datei vor dem Schließen des Fensters gespeichert werden soll.

Liste offener windows (Window menu)

Am unteren Ende des Menüs wird eine Liste offener Dateien angezeigt.

Falls mehr als ein Fenster geöffnet ist, können Sie in ein anderes Fenster wechseln, indem Sie es durch Doppelklicken oder Eingabe der entsprechenden Fensternummer auswählen.

Eingabefeld

In ein Eingabefeld können Sie Text eintippen oder ein Wort, einen Ausdruck, einen Datei- oder Verzeichnis-Namen, wobei Sie auch die in DOS üblichen Jokerzeichen (* und ?) benutzen können.

In einem Eingabefeld funktionieren alle regulären Editiertasten. Falls die Eingabe nicht vollständig in die Zeile paßt, wird der Inhalt automatischgerollt.

Falls am rechten Rand des Eingabefeldes ein Pfeilsymbol angezeigt wird, ist eine **Eingabeaufzeichnungsliste** zu diesem Feld verfügbar.

Dateinamen eingeben und Dateilisten anzeigen lassen

Falls Sie einen vollständigen Namen eingeben und Return drücken, öffnet Turbo Pascal die Datei (bzw. legt eine neue an, falls sie noch nicht existiert). Falls Sie aber nur einen Verzeichnisnamen eingeben oder einen Dateinamen mit DOS-Jokerzeichen, so erhalten Sie eine entsprechend gefilterte Dateiauswahlliste.

Dateiliste

Die Dateiliste zeigt folgende Einträge:

- alle Dateinamen im aktuellen Verzeichnis, die zu den Angaben im Eingabefeld passen.
- das übergeordnete Verzeichnis.
- alle Unterverzeichnisse.

Sie aktivieren die Dateiliste, indem Sie sie anklicken oder solange Tab drücken, bis der Name des Listenfensters hervorgehoben angezeigt wird.

Das Fenster wird aktualisiert und zeigt die Dateien des Verzeichnisses, das Sie ausgewählt haben.

Mit den Pfeiltasten Auf und Ab bewegen Sie den Cursor durch die Liste. Wenn die gewünschte Datei hervorgehoben ist, öffnen Sie sie, indem Sie Return drücken oder OK anklicken.

Sie können jede beliebige Datei ebenso durch einen Doppelklick öffnen. Eventuell müssen Sie das Fenster rollen, um alle Dateinamen sehen zu können.

Ein anderer Weg, den Cursor durch die Dateiliste zu bewegen: Tippen Sie den/die ersten Buchstaben des Dateinamens ein.

Oberhalb des Fensters sehen Sie einen Informationsbereich, der den Pfadnamen und Dateinamen der ausgewählten Datei zeigt. Keine dieser Informationen ist wählbar.

Dieser Bereich wird beim Durchsuchen der Liste jeweils aktualisiert.

Editorfenster

Im Editorfenster geben Sie Ihren Turbo-Pascal-Code ein und editieren ihn. Zudem können Sie folgende Aktionen ausführen:

- Programme **compilieren**
- Programme **ausführen**
- Programme von Diskette/Platte einlesen
- Programme speichern

Sie können so viele Editorfenster öffnen, wie Sie wünschen. Turbo Pascal numeriert aber nur die ersten neun geöffneten Fenster (alle Arten von Fenstern zählen mit).

Editorfenster öffnen

Sie öffnen ein Editorfenster, indem Sie den Befehl File|**Open (Datei/Öffnen)** geben. Sie können die gleiche Datei in mehreren Fenstern bearbeiten.

Weitere Informationen

Wählen Sie aus der folgenden Liste:

Die Fenster von Turbo Pascal verwenden
Der Editor
Dialogfenster Find (Text suchen)
DialogfensterReplace (Text ersetzen)
Das Menü Edit (Bearbeiten)
File Save (Datei Speichern)
File Save As (Datei Speichern unter)

Die Fenster von Turbo Pascal

Die meisten Arbeiten in Turbo Pascal werden in und mit Fenstern durchgeführt. Ein Fenster ist ein Teil des Bildschirms, den Sie verschieben, vergrößern, auf ein Symbol verkleinern, teilen, ablegen, öffnen und schließen können.
Ein Fenster hat folgende Elemente:

SystemmenüfeldTitelbalken
Bildlaufleisten

In Turbo Pascal können Sie bis zu 32 Fenster öffnen (d.h. soviele Ihr Speicher erlaubt). Eines davon, nämlich das, in dem Sie gerade arbeiten, ist das aktive Fenster.

Alle Arbeiten, die Sie durchführen, beziehen sich auf das aktive Fenster. (Wenn Sie dieselbe Datei in verschiedenen Fenstern bearbeiten, sehen Sie Ihre Tastendrucke in allen Fenstern.)

Überlappen sich die Fenster, so befindet sich das aktive Fenster immer im Vordergrund.

Die aktuelle Zeilen- und Spaltennummer wird jeweils am unteren Rand eines Editorfensters angezeigt.

TApplication (WObjects unit)

TApplication bildet das Fundament aller ObjectWindows-Programme.

Alle ObjectWindows-Programme leiten einen Objekttyp von TApplication ab, primär, um ein Fenster eines selbst definierten Objekttyps zu erstellen.

Felder

HAccTable
KBHandlerWnd
MainWindow
Name
Status

Methoden

Init
Done
CanClose
Error
ExecDialog
InitApplication
InitInstance
InitMainWindow
MakeWindow
MessageLoop
ProcessAppMsg
ProcessDlgMsg
ProcessAccels
ProcessMDIAccels
Run
SetKBHandler
ValidWindow

TApplication.HAccTable (Feld)

Syntax

```
HAccTable: THandle;
```
(Read/Write)

Beschreibung

HAccTable hat ein Handle zur Tabelle der Windows-Tastenkürzel-Ressource für das Programm.

Siehe auch

TApplication

TApplication.KBHandlerWnd (Feld)

Syntax

```
KBHandlerWnd: PWindowsObject;
```
(Read only)

Beschreibung

KBHandlerWnd zeigt auf das aktive Fenster, wenn dessen Tastatur-Handler auch aktiv ist. So können die Elemente des Fensters Tasteneingaben wie ein Dialogfenster bearbeiten.

KBHandlerWnd ist **nil**, wenn der Tastatur-Handler nicht aktiv ist.

Siehe auch

TApplication

TApplication.MainWindow (Feld)

Syntax

```
MainWindow: PWindowsObject;
```
(Read/Write)

Beschreibung

MainWindow zeigt auf das überlappende Hauptfenster des Programms, das von **InitMainWindow method** instantiiert werden sollte.

Siehe auch

TApplication

TApplication.Name (Feld)

Beschreibung

Name enthält den Namen des Programms, der auch intern im Programm verwendet werden kann.

Siehe auch

TApplication

TApplication.Status (Feld)

Syntax

```
Status: Integer;
```

Beschreibung

Status gibt den Status des laufenden Programms an. Dieses läuft erfolgreich, wenn Status größer oder gleich Null ist.

Fehlerwerte sind z.B. **em_InvalidMainWindow** bei einem ungültigen Hauptfenster.

Siehe auch

TApplication

TApplication.Init (Methode)

Syntax

```
constructor Init(AName: PChar);
```

Beschreibung

Wird manchmal überschrieben. Erstellt das Programmobjekt.

- Ruft **TObject.Init** auf
- Setzt die globale Variable Application auf @Self
- Setzt das Feld **Name** auf den String, auf welchen AName zeigt
- Setzt **HAccTable field** und das **Statusfeld** auf 0
- Initialisiert das **Hauptfenster** und **KBHandlerWnd** auf nil.

Ist dies die erste ablaufende Instanz des Programms, ruft Init **InitApplication** auf. Ist InitApplication erfolgreich, wird **InitInstance** aufgerufen.

Siehe auch

TApplication
TObject.Init

TApplication.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Wird manchmal überschrieben. Löscht die Objekte des Programms durch das Löschen des **Programmfensters** und ruft **TObject.Done** auf, um das Programm zu beenden.

Siehe auch

TApplication

TApplication.CanClose (Methode)

Syntax

```
function CanClose: Boolean; virtual;
```

Beschreibung

Wird selten überschrieben. Gibt True zurück, wenn das Programm beendet werden kann.

Als Vorgabe wird die Methode CanClose des Hauptfensters aufgerufen und deren Rückgabewert zurückgegeben.

Das Verhalten beim Beenden des Programms kann in der Methode CanClose des Hauptfensters überschrieben werden.

Siehe auch

TWindowsObject.CanClose
TWindowsObject.WMDestroy
TApplication

TApplication.MakeWindow (Methode)

Syntax

```
function MakeWindow(AWindowsObject: PWindowsObject): PWindowsObject;
virtual;
```

Beschreibung

Wird nie überschrieben. Versucht, ein Fenster oder ein Element ohne Modus in Zusammenhang mit dem Objekt zu erzeugen, das von AWindowsObject übergeben wurde, nachdem die Verwendung des Sicherheitsbereichs geprüft wurde.

Ist zu wenig Speicher vorhanden oder das Objekt konnte nicht erstellt werden, entfernt MakeWindow dieses und gibt **nil** zurück.

Ist es erfolgreich, gibt es AWindowsObject zurück.

Siehe auch

TWindow.Create
TApplication

TApplication.Error (Methode)

Syntax

```
procedure Error(ErrorCode: Integer); virtual;
```

Beschreibung

Wird manchmal überschrieben. Error bearbeitet Fehler, die vom Fehlerwert in ErrorCode definiert wurden. Solche Fehler können vom Programm oder jedem Fenster- oder Dialogobjekt stammen.

ErrorCode kann einer der folgenden oder ein selbst definierter Fehler sein:

em_OutOfMemory
em_InvalidClient
em_InvalidChild
em_InvalidMainWindow
em_InvalidWindow

Error stellt den Fehlercode in einem Meldungsfenster dar und fragt nach, ob das Programm fortgeführt werden soll. Im negativen Fall stoppt das Programm.

Siehe auch

em_XXXX-Konstanten
TApplication

TApplication.ExecDialog (Methode)

Syntax

```
function ExecDialog(ADialog: PWindowsObject): Integer; virtual;
```

Beschreibung

Wird nie überschrieben. Führt ein modales Dialogobjekt von ADialog aus, nachdem die Verwendung des Stack-Sicherheitsbereichs geprüft wurde.

Wenn zu wenig Arbeitsspeicher vorhanden ist oder der Dialog nicht ausgeführt werden kann, entfernt ExecDialog das Objekt und gibt einen negativen Fehlerstatus zurück.

Siehe auch

TDialog.Execute
TApplication

TApplication.InitApplication (Methode)

Syntax

```
procedure InitApplication; virtual;
```

Beschreibung

Wird manchmal überschrieben. Initialisiert das Nötige für den Start des Programms.

TApplication.InitApplication hat keinerlei Wirkung. Ein Programm kann InitApplication überschreiben, um sich damit zu initialisieren.

Siehe auch

TApplication

TApplication.InitInstance (Methode)

Syntax

```
procedure InitInstance; virtual;
```

Beschreibung

Wird manchmal überschrieben. Erledigt die Initialisierungen für alle ausführenden Instanzen des Programms.

TApplication.InitInstance ruft **InitMainWindow** auf, erzeugt das Hauptfensterelement mit **MakeWindow** und zeigt es mit **TWindowsObjectShow**.

Wenn Sie diese Methode überschreiben, sollten Sie explizit TApplication.InitInstance aufrufen.

Siehe auch

TApplication

TApplication.InitMainWindow (Methode)

Syntax

```
procedure InitMainWindow; virtual;
```

Beschreibung

Wird immer überschrieben. Überschreiben Sie InitMainWindow, um ein Hauptfensterobjekt zu erstellen, und speichern Sie es in **MainWindow**.

Typische Verwendung:

```
procedure MyApplication.InitMainWindow;
begin
   MainWindow := New(PMyWindow, Init('Window Caption'));
end;
```

Als Vorgabe erzeugt InitMainWindow ein Fenster ohne Titel des Typs **TWindow**.

Siehe auch

TApplication

TApplication.MessageLoop (Methode)

Syntax

```
procedure MessageLoop; virtual;
```

Beschreibung

Wird nie überschrieben. Die Botschaftsbearbeitungs-Schleife des Programms, die so lange wie das Programm läuft.

TApplication.MessageLoop ruft **ProcessAppMsg** auf, um besondere Botschaften für Dialoge ohne Modus, Tastenkürzel und MDI-Tastenkürzel zu verwalten.

Siehe auch

TApplication

TApplication.ProcessAppMsg (Methode)

Syntax

```
function ProcessAppMsg(var Message: TMsg): Boolean; virtual;
```

Beschreibung

Wird manchmal überschrieben. Prüft auf Botschaften von nicht-modalen Dialogfenstern, Tastenkürzeln und MDI-Tastenkürzeln.

Ruft **ProcessDlgMsg**, **ProcessMDIAccels** und **ProcessAccels** auf. Gibt True zurück, wenn besondere Botschaften auftraten.

Treten derartige Botschaften in Ihrem Programm nicht auf, läuft es schneller, wenn Sie diese Methode überschreiben und sie einfach False zurückgeben lassen.

Siehe auch

TApplication

TApplication.ProcessDlgMsg (Methode)

Syntax

```
function ProcessDlgMsg(var Message: TMsg): Boolean; virtual;
```

Beschreibung

Wird manchmal überschrieben. Erledigt die besondere Botschaftsbearbeitung für nicht-modale Dialogfenster, wenn Tastatureingaben auf Steuerelemente gemacht werden.

Enthält Ihr Programm keine derartigen Fenster, so läuft es schneller, wenn Sie diese Methode überschreiben und sie einfach False zurückgeben lassen.

Siehe auch

TApplication

TApplication.ProcessAccels (Methode)

Syntax

```
function ProcessAccels(var Message: TMsg): Boolean; virtual;
```

Beschreibung

Wird manchmal überschrieben. Verwaltet besondere Tastenkürzel-Botschaften.

Falls Sie in Ihrem Programm keine Tastenkürzel verwenden, läuft es schneller, wenn Sie diese Methode überschreiben und sie einfach False zurückgeben lassen.

Siehe auch

TApplication

TApplication.ProcessMDIAccels (Methode)

Syntax

```
function ProcessMDIAccels(var Message: TMsg): Boolean; virtual;
```

Beschreibung

Verwaltet besondere Tastenkürzel-Botschaften für MDI-Programme.

Treten derartige Botschaften in Ihrem Programm nicht auf, läuft es schneller, wenn Sie diese Methode überschreiben und sie einfach False zurückgeben lassen.

Siehe auch

TApplication

TApplication.Run (Methode)

Syntax

```
procedure Run; virtual;
```

Beschreibung

Wird selten überschrieben. Startet das Programm durch Aufruf von **MessageLoop** (in manchen Fällen), **InitInstance** und **MessageLoop**.

Siehe auch

TApplication

TApplication.SetKBHandler (Methode)

Syntax

```
procedure SetKBHandler(AWindowsObject: PWindowsObject);
```

Beschreibung

Wird nie überschrieben. Setzt **KBHandlerWnd** auf AWindowsObject.

Siehe auch

TApplication

TApplication.ValidWindow (Methode)

Syntax

```
function
ValidWindow(AWindowsObject: PWindowsObject):PWindowsObject;
```

Beschreibung

Bestimmt, ob `AWindowsObject` ein gültiges Objekt ist. Wenn ja, gibt ValidWindow einen Zeiger darauf zurück, ansonsten nil.

Siehe auch

TApplication
TWindowsObject.Status

TBufStream (Unit WObjects)

TBufStream implementiert eine gepufferte Version von **TDosStream**. Die zusätzlichen Felder geben Größe und Position des Puffers an, sowie die aktuelle und vorige Cursorposition im Puffer. TBufStream überschreibt die acht Methoden von TDosStream und definiert die abstrakte Methode **TStreamFlush**. Der Konstruktor TBufStream erzeugt und öffnet eine benannte Datei durch Aufruf von **TDosStream.Init**, danach den Puffer mit GetMem.

TBufStream ist deutlich effizienter als TDosStream, wenn viele kleine Datenmengen auf den Stream übertragen werden, wie z.B. beim Laden und Speichern von Objekten mit **TStream.Get** und **TStream.Put**.

Felder

- **Buffer**
- **BufSize**
- **BufPtr**
- **BufEnd**

Methoden

- **Init**
- **Done**
- **Flush**
- **GetPos**
- **GetSize**
- **Read**
- **Seek**
- **Truncate**
- **Write**

TBufStream.Buffer (Feld)

Syntax

```
Buffer: Pointer;
```
(Read only)

Beschreibung

Zeiger auf den Beginn des Stream-Puffers.

Siehe auch

TBufStream

TBufStream.BufSize (Feld)

Syntax

```
BufSize: Word;
```
(Read only)

Beschreibung

Die Größe des Puffers in Bytes.

Siehe auch

TBufStream

TBufStream.BufPtr (Feld)

Syntax

```
BufPtr: Word;
```
(Read only)

Beschreibung

Das Offset des Zeigers Buffer,: die aktuelle Position innerhalb des Puffers.

Siehe auch

TBufStream

TBufStream.BufEnd (Feld)

Syntax

```
BufEnd: Word;
```
(Read only)

Beschreibung

Wenn der Puffer nicht voll ist, steht in BufEnd das Offset des Zeigers Buffer auf das letzte belegte Byte im Puffer.

Siehe auch

TBufStream

TBufStream.Init (Methode)

Syntax

```
constructor Init(FileName: FNameStr; Mode, Size: Word);
```

Beschreibung

Erzeugt und öffnet die benannte Datei mit dem Zugriffsmodus Mode durch Aufruf von **TDosStream.Init**. Erzeugt einen Puffer von Size Bytes mit einem Aufruf von GetMem. Die Felder Handle, Buffer und BufSize werden passend initialisiert. Typische Puffergrößen gehen von 512 Bytes bis 2,048 Bytes.

Siehe auch

TBufStream
TDosStream.Handle

TBufStream.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Schließt und entfernt den Datei-Stream, speichert den Inhalt seines Puffers und entfernt ihn.

Siehe auch

Flush
TBufStream

TBufStream.Flush (Methode)

Syntax

```
procedure Flush; virtual;
```

Beschreibung

Speichert den Puffer des aufrufenden Stream, wenn der Stream stOK ist.

Siehe auch

Done
TBufStream

TBufStream.GetPos (Methode)

Syntax

```
function GetPos: Longint; virtual;
```

Beschreibung

Gibt den Wert der aktuellen Position des aufrufenden Stream zurück (nicht zu verwechseln mit BufPtr, der aktuellen Position innerhalb des Puffers).

Siehe auch

Seek
TBufStream

TBufStream.GetSize (Methode)

Syntax

```
function GetSize: Longint; virtual;
```

Beschreibung

Speichert den Puffer und gibt die Gesamtgröße des aufrufenden Stream in Bytes zurück.

Siehe auch

TBufStream

TBufStream.Read (Methode)

Syntax

```
procedure Read(var Buf; Count: Word); virtual;
```

Beschreibung

Wenn stOK true ist, werden ab der aktuellen Position des aufrufenden Streams Count Bytes gelesen und in den Puffer Buf geschrieben.

Beachten Sie, daß Buf nicht der Stream-Puffer, sondern ein externer Puffer ist, der die Daten aufnimmt, die vom Stream gelesen werden.

Siehe auch

stXXXX-Konstanten
TBufStream
Write

TBufStream.Seek (Methode)

Syntax

```
procedure Seek(Pos: Longint); virtual;
```

Beschreibung

Speichert den Puffer und setzt die aktuelle Position Pos Bytes vom Beginn des aufrufenden Stream. Der Beginn ist die Position 0.

Siehe auch

GetPos
TBufStream

TBufStream.Truncate (Methode)

Syntax

```
procedure Truncate; virtual;
```

Beschreibung

Speichert den Puffer und löscht alle Daten auf dem aufrufenden Stream von der aktuellen Position bis zum Ende. Die aktuelle Position wird auf das neue Ende des Stream gesetzt.

Siehe auch

GetPos
Seek
TBufStream

TBufStream.Write (Methode)

Syntax

```
procedure Write(var Buf; Count: Word); virtual;
```

Beschreibung

Wenn der Stream stOK ist, werden Count Bytes aus dem Puffer Buf ab der aktuellen Position in den Stream geschrieben.

Beachten Sie, daß Buf nicht der Stream-Puffer, sondern ein externer Puffer ist, der die Daten aufnimmt, die in den Stream geschrieben werden. Wenn Write aufgerufen wird, zeigt Buf auf die Variable, deren Wert geschrieben wird.

Siehe auch

Read
stXXXX-Konstanten
TBufStream

TButton (Unit WObjects)

TButton ist ein Schnittstellenobjekt, das einen entsprechenden Aktionsschalter in Windows darstellt. TButton wird normalerweise nicht in Dialogfeldern (**TDialog**) oder -fenstern (**TDlgWindow**) verwendet, aber als selbständiger Aktionsschalter in einem untergeordneten Fenster des Client-Bereichs eines anderen Fensters.

Es gibt zwei Typen von Aktionsschaltern: Mit dünnem Rand als normaler Aktionsschalter, mit dickem Rand als Vorgabe-Aktionsschalter des Fensters, wovon es nur einen geben kann.

Methoden

Init
InitResource
GetClassName

TButton.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnId: Integer; AText: PChar;
X,Y,W,H: Integer; IsDefault: Boolean);
```

Beschreibung

Erzeugt ein Listenfenster-Objekt mit
- der Identifikationsnummer (AnId) des übergeordneten Fensters (AParent),
- dem Text (AText),
- der Position (X, Y), relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters,
- der Breite (W) und Höhe (H).

Ruft **TControl.Init** auf und fügt zum Feld Attr.Style bs_DefPushButton hinzu, wenn IsDefault True ist, ansonsten bs_PushButton.

Siehe auch

bs_ Button styles
TButton

TButton.InitResource (Methode)

Syntax

```
constructor InitResource(AParent: PWindowsObject, ResourceID: Word);
```

Beschreibung

Assoziiert den Aktionsschalter durch Erstellen eines ObjectWindows-Objekts, damit er einem Aktionsschalter-Element einer Ressourcendefinition entspricht. Ruft **TControl.InitResource** und **TWindowsObject.DisableTransfer** auf, um den Aktionsschalter von dem Transfermechanismus auszuschließen, da keine Daten zu übertragen sind.

Siehe auch

TButton

TButton.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt den Namen der Fensterklasse von TButton zurück, 'Button'.

Siehe auch

TButton

TCheckBox (Unit WObjects)

TCheckBox ist ein Schnittstellenobjekt, das einen entsprechenden Aktionsschalter in Windows darstellt. TCheckBox wird normalerweise nicht in Dialogfenstern (**TDialog** oder **TDlgWindow**) verwendet, sondern als selbständiges Wahlfeld in einem untergeordneten Fenster eines anderen Fensters.

Wahlfelder können gewählt oder nicht gewählt sein. Die Methoden von TCheckBox verwalten den Status des Wahlfelds. Ein Wahlfeld kann Teil einer Gruppe sein (**TGroupBox**), welche verschiedene Elemente visuell und funktional gruppiert.

Felder

Group
InitResource

Methoden

Init
BNClicked
Check
GetCheck
Load
SetCheck
Store
Toggle
Transfer
Uncheck

TCheckBox.Group (Feld)

Syntax

```
Group: PGroupBox;
```
(Read only)

Beschreibung

Group zeigt auf das Element **TGroupBox**, welches das Wahlfeld mit anderen Wahlfeldern und Schaltfeldern verbindet (**TRadioButton**). Wenn das Wahlfeld nicht Teil einer Gruppe ist, ist Group = **nil**.

Siehe auch

TCheckBox

TCheckBox.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnID: Integer; ATitle: PChar;
X,Y,W,H: Integer; AGroup: PGroupBox);
```

Beschreibung

Erzeugt ein Wahlfeld-Objekt mit
- der Identifikationsnummer (AnId) des übergeordneten Fensters (AParent),
-dem Titel (ATitle),
-der Position (X, Y), relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters,
- der Breite (W)
- und Höhe (H) und
- einem Gruppenfenster (AGroup).

Init setzt das Feld Attr.Style des Wahlfelds auf ws_Child oder ws_Visible oder ws_TabStop oder **bs_AutoCheckBox**.

Siehe auch

TCheckBox
ws_Window Styles

TCheckBox.InitResource (Methode)

Syntax

```
constructor InitResource(AParent: PWindowsObject; ResourceID: Word);
```

Beschreibung

Assoziiert ein TCheckbox-Objekt mit der Ressource in `ResourceID`. Schaltet dann den Transfermechanismus durch Aufruf von EnableTransfer ein.

Siehe auch

TCheckBox

TCheckBox.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Konstruiert und lädt ein Wahlfeld vom Stream S, indem es erst TButton.Load aufruft und dann das zusätzliche Feldes (Group) von TCheckBox einliest.

Siehe auch

TCheckBox
TWindow.Load

TCheckBox.BNClicked (Methode)

Syntax

```
procedure BNClicked(var Msg: TMessage); virtual nf_First + bn_Clicked;
```

Beschreibung

Reagiert automatisch auf Anklicken des Wahlfelds und ändert dessen Status.
Ist die Gruppe des Wahlfelds nicht **nil**, informiert BNClicked **TGroupBox** durch Aufruf der Methode SelectionChanged.

Siehe auch

TCheckBox
TGroupBox.SelectionChanged

TCheckBox.Check (Methode)

Syntax

```
procedure Check; virtual;
```

Beschreibung

Setzt das Wahlfeld auf den gewählten Status durch Aufruf von SetCheck.

Siehe auch

SetCheck
TCheckBox

TCheckBox.GetCheck (Methode)

Syntax

```
function GetCheck: Word; virtual;
```

Beschreibung

Gibt bf_Unchecked (zero) zurück, wenn das Wahlfeld nicht gewählt ist, bf_Checked (1) im Wahlfall oder bf_Grayed (2), wenn es grau = nicht aktiv ist.

Siehe auch

bf_XXXX Konstanten
TCheckBox

TCheckBox.SetCheck (Methode)

Syntax

```
procedure SetCheck(CheckFlag: Word); virtual;
```

Beschreibung

Setzt das Wahlfeld auf den Status von CheckFlag.

Ist CheckFlag = 0, wird der Status zu "nicht gewählt", bei 1 zu gewählt, bei 2 zu grau (nicht aktiv). SetCheck informiert die Gruppe des Wahlfelds, daß der Status geändert wurde.

Siehe auch

TCheckBox
TGroupBox.SelectionChanged

TCheckBox.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert das Wahlfeld in den Stream S durch Aufruf von TControl.Store und anschließendes Schreiben des zusätzlichen Feldes (Group), das von TCheckBox eingeführt wird.

Siehe auch

TCheckBox
TWindow.Store

TCheckBox.Toggle (Methode)

Syntax

```
procedure Toggle; virtual;
```

Beschreibung

Schaltet den Status des Wahlfelds durch Aufruf von Check oder Uncheck. Gibt es drei Stati, werden alle nacheinander geschaltet: Checked (gewählt), unchecked (nicht gewählt) und grayed (nicht aktiv).

Siehe auch

Check
TCheckBox
Uncheck

TCheckBox.Transfer (Methode)

Syntax

```
function Transfer(DataPtr: Pointer; TransferFlag: Word): Word; virtual;
```

Beschreibung

Überträgt den Booleschen Status des Wahlfelds (True oder False) zu oder von der Speicheradresse, auf die DataPtr zeigt.

Hat TransferFlag den Wert tf_GetData, wird der Status geschrieben, hat es den Wert tf_SetData, wird der Status gelesen und das Wahlfeld entsprechend gesetzt.

Transfer gibt die Anzahl der übertragenen Bytes zurück. Mit tf_SizeData als Parameter liefert Transfer den Umfang der Transferdaten zurück.

Siehe auch

TCheckBox
tf_XXXX-Konstanten

TCheckBox.Uncheck (Methode)

Syntax

```
procedure Uncheck; virtual;
```

Beschreibung

Setzt das Wahlfeld auf den Status "nicht gewählt", indem es SetCheck aufruft.

Siehe auch

SetCheck
TCheckBox

TCollection (Unit WObjects)

TCollection bietet eine allgemeine Grundlage zur Implementierung einer Kollektion von Daten und Objekten. Das Konzept, das TCollection zugrundeliegt, ist wesentlich flexibler als das von Arrays, Sets oder Listen. TCollection paßt sich zur Laufzeit dynamisch an die Größe der ihm übergebenen Daten an und bildet die Grundlage für spezialisiertere Objekttypen, wie **TSortedCollection** und **TStrCollection**. Das Objekt TCollection verfügt über Methoden, die es erlauben, Einträge hinzufügen oder zu löschen. Darüber hinaus stehen Iteratormethoden bereit, die eine vom Anwender zu definierende Funktion auf jeden Eintrag in der Kollektion anwenden.

Felder

- **Count**
- **Delta**
- **Items**
- **Limit**

Methoden

- **Init**
- **Done**
- **AtDelete**
- **AtFree**
- **AtInsert**
- **AtPut**
- **At**
- **DeleteAll**
- **Delete**
- **Error**
- **FirstThat**
- **ForEach**
- **FreeAll**
- **FreeItem**
- **Free**
- **GetItem**
- **IndexOf**
- **Insert**
- **LastThat**
- **Load**
- **Pack**
- **PutItem**
- **SetLimit**
- **Store**

TCollection.Items (Feld)

Syntax

```
Items: PItemList;
```
(Read only)

Beschreibung

Ein Zeiger auf ein Array von Elementzeigern.

Siehe auch

TCollection
TItemList-Typ

TCollection.Count (Feld)

Syntax

```
Count: Integer;    (Read only)
```

Beschreibung

Aktuelle Anzahl der Elemente der Kollektion , bis zu MaxCollectionSize.

Siehe auch

TCollection

TCollection.Limit (Feld)

Syntax

```
Limit: Integer;
```
(Read only)

Beschreibung

Die aktuell zugewiesene Länge der Item-Liste in Elementen.

Siehe auch

Delta
Init
TCollection

TCollection.Delta (Feld)

Syntax

```
Delta: Integer;
```
(Read only)

Beschreibung

Die Anzahl der Elemente, um welche die Items Liste vergrößert wird, wenn sie voll ist. Ist Delta Null, kann die Kollektion nicht größer als Limit werden.

Das Vergrößern der Kollektion kostet einiges an Zeit. Damit dies nur selten anfällt, sollten Sie Limit so hoch ansetzen, daß alle Elemente in die Kollektion passen, und Delta nicht zu klein wählen.

Siehe auch

Init
Limit
TCollection

TCollection.Init (Methode)

Syntax

```
constructor Init(ALimit, ADelta: Integer);
```

Beschreibung

Erzeugt eine Kollektion mit Limit auf ALimit und Delta auf ADelta. Zu Beginn sind die Elemente auf ALimit begrenzt, doch kann die Kollektion in ADelta Schritten wachsen, bis der Speicher voll ist oder MaxCollectionSize erreicht wird.

Siehe auch

Delta
Limit
TCollection

TCollection.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Erzeugt und lädt eine Kollektion aus dem übergebenen Stream. Load ruft GetItem für jedes Element der Kollektion auf.

Siehe auch

GetItem
TCollection

TCollection.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Löscht und entfernt alle Elemente der Kollektion durch Aufruf von FreeAll und Setzen von Limit auf 0.

Siehe auch

FreeAll
Init
TCollection

TCollection.At (Methode)

Syntax

```
function At(Index: Integer): Pointer;
```

Beschreibung

Liefert einen Zeiger auf das Element, welches durch Index in der Kollektion indiziert wird. Mit dieser Methode können Sie eine Kollektion wie ein indiziertes Array behandeln. Ist Index kleiner Null oder größer/gleich Count, wird die Methode Error mit dem Argument coIndexError aufgerufen und der Wert **nil** zurückgegeben.

Siehe auch

IndexOf
TCollection

TCollection.AtDelete (Methode)

Syntax

```
procedure AtDelete(Index: Integer);
```

Beschreibung

Löscht das Element an der Position Index und schiebt nachfolgende Elemente eine Stelle nach vorne. Count wird um 1 dekrementiert, aber der Speicher Limit für die Kollektion wird nicht reduziert. Ist Index kleiner Null oder größer/gleich Count, wird die Methode Error mit dem Argument coIndexError aufgerufen.

Siehe auch

Delete
Free
FreeItem
TCollection

TCollection.AtFree (Methode)

Syntax

```
procedure AtFree(Index: Integer);
```

Beschreibung

Entfernt und löscht das Element an der Position Index.

Siehe auch

TCollection

TCollection.AtInsert (Methode)

Syntax

```
procedure AtInsert(Index: Integer; Item: Pointer);
```

Beschreibung

Fügt Item an der Position Index ein und schiebt nachfolgende Elemente um eine Stelle nach hinten.

Ist Index kleiner Null oder größer Count, wird die Methode Error mit dem Argument coIndexError aufgerufen und das neue Item nicht eingefügt.

Wenn Count gleich Limit ist, bevor AtInsert aufgerufen wird, vergrößert sich die Kollektion um Delta durch Aufruf von SetLimit.

Wenn SetLimit die Kollektion nicht vergrößern konnte, wird das neue Item nicht eingefügt und statt dessen die Methode Error mit dem Argument coOverflow aufgerufen.

Siehe auch

At
AtPut
TCollection

TCollection.AtPut (Methode)

Syntax

```
procedure AtPut(Index: Integer; Item: Pointer);
```

Beschreibung

Ersetzt das Element an der Position Index durch das Element aus Item. Ist Index kleiner Null oder größer/gleich Count, wird die Methode Error mit dem Argument coIndexError aufgerufen.

Siehe auch

At
AtInsert
TCollection

TCollection.Delete (Methode)

Syntax

```
procedure Delete(Item: Pointer);
```

Beschreibung

Löscht das Element in Item aus der Kollektion. Entspricht AtDelete(IndexOf(Item)).

Siehe auch

AtDelete
DeleteAll
TCollection

TCollection.DeleteAll (Methode)

Syntax

```
procedure DeleteAll;
```

Beschreibung

Löscht alle Elemente aus der Kollektion, indem Count auf Null gesetzt wird.

Siehe auch

AtDelete
Delete
TCollection

TCollection.Error (Methode)

Syntax

```
procedure Error(Code, Info: Integer); virtual;
```

Beschreibung

Wird aufgerufen, wenn ein Fehler der Kollektion auftritt. Als Vorgabe wird ein Laufzeitfehler (212 - Code) erzeugt.

Siehe auch

TCollection
coXXXX Konstanten

TCollection.FirstThat (Methode)

Syntax

```
function FirstThat(Test: Pointer): Pointer;
```

Beschreibung

FirstThat wendet eine Boolesche Funktion des Funktionszeigers Test auf jedes Element der Kollektion an, bis Test True zurückgibt. Das Ergebnis ist der erste Elementzeiger, für den die Funktion Test True zurückgab, oder **nil**, wenn sie für alle Elemente False zurückgab. Test muß auf eine lokale **far** Funktion zeigen, einen Pointer-Parameter verwenden und einen Booleschen Wert zurückgeben. Beispiel:

```
function Matches(Item: Pointer): Boolean; far;
```

Die Funktion Test kann nicht global sein.

Wenn List eine TCollection ist, sind die Anweisungen

```
P := List.FirstThat(@Matches);
```

und

```
I := 0;
while (I < List.Count) and not Matches(List.At(I)) do Inc(I);
if I < List.Count then P := List.At(I) else P := nil;
```

äquivalent.

Siehe auch

ForEach
LastThat
TCollection

TCollection.ForEach (Methode)

Syntax

```
procedure ForEach(Action: Pointer);
```

Beschreibung

ForEach wendet die Prozedur, auf die Action zeigt, auf jedes Element der Kollektion an. Action muß auf eine lokale **far** Prozedur zeigen und einen Parameter vom Typ Pointer verwenden. Beispiel:

```
function PrintItem(Item: Pointer); far;
```

Die Prozedur Action kann nicht global sein.

Wenn List eine TCollection ist, sind die Anweisungen

```
List.ForEach(@PrintItem);
```

und

```
for I := 0 to List.Count - 1 do PrintItem(List.At(I));
```

äquivalent.

Siehe auch

FirstThat
LastThat
TCollection

TCollection.Free (Methode)

Syntax

```
procedure Free(Item: Pointer);
```

Beschreibung

Löscht und entfernt ein gegebenes Item. Entspricht

```
Delete(Item);
FreeItem(Item);
```

Siehe auch

Delete
FreeItem
TCollection

TCollection.FreeAll (Methode)

Syntax

```
procedure FreeAll;
```

Beschreibung

Löscht und entfernt alle Elemente der Kollektion.

Siehe auch

DeleteAll
TCollection

TCollection.FreeItem (Methode)

Syntax

```
procedure FreeItem(Item: Pointer); virtual;
```

Beschreibung

Die Methode FreeItem muß das gegebene Item entfernen. Die Vorgabemethode FreeItem setzt voraus, daß Item ein Zeiger auf einen Nachkommen von **TObject** ist, und ruft entsprechend den Destruktor Done auf:

```
if Item <> nil then Dispose(PObject(Item), Done);
```

FreeItem wird von Free und FreeAll aufgerufen, sollte aber nie direkt aufgerufen werden.

Siehe auch

Free
FreeAll
FreeItem
TCollection

TCollection.GetItem (Methode)

Syntax

```
function TCollection.GetItem(var S: TStream): Pointer; virtual;
```

Beschreibung

Wird von Load für jedes Element der Kollektion aufgerufen. Diese Methode kann überschrieben werden, sollte aber nicht direkt aufgerufen werden. GetItem setzt voraus, daß die Elemente der Kollektion Nachkommen von **TObject** sind und ruft entsprechend **TStream.Get** auf, um die Elemente zu laden:

```
GetItem := S.Get;
```

Siehe auch

GetItem
Load
Store
TCollection

TCollection.IndexOf (Methode)

Syntax

```
function IndexOf(Item: Pointer): Integer; virtual;
```

Beschreibung

Liefert den Index auf das übergebene Item. Die umgekehrte Operation zu At. Ist Item nicht in der Kollektion, gibt IndexOf -1 zurück.

Siehe auch

At
TCollection

TCollection.Insert (Methode)

Syntax

```
procedure Insert(Item: Pointer); virtual;
```

Beschreibung

Fügt Item in die Kollektion ein und aktualisiert andere Indizes, falls nötig. Vorgabe ist Einfügen am Ende der Kollektion durch AtInsert(Count, Item);

Siehe auch

AtInsert
TCollection

TCollection.LastThat (Methode)

Syntax

```
function LastThat(Test: Pointer): Pointer;
```

Beschreibung

FirstThat wendet die Boolesche Funktion, auf die Test zeigt, auf jedes Element der Kollektion in umgekehrter Reihenfolge an, bis Test True zurückgibt. Das Ergebnis ist der Elementzeiger, für den Test True zurückgab, oder **nil**, wenn die Funktion Test für alle Elemente False zurückgibt. Test muß auf eine lokale **far** Funktion zeigen, einen Pointer Parameter verwenden und einen Boolean Wert zurückgeben. Beispiel:

```
function Matches(Item: Pointer): Boolean; far;
```

Die Funktion Test kann nicht global sein.

Wenn List eine TCollection ist, sind die Anweisungen

```
P := List.LastThat(@Matches);
```

und

```
I := List.Count - 1;
while (I >= 0) and not Matches(List.At(I)) do Dec(I);
if I >= 0 then P := List.At(I) else P := nil;
```

äquivalent.

Siehe auch

FirstThat
ForEach
TCollection

TCollection.Pack (Methode)

Syntax

```
procedure Pack;
```

Beschreibung

Löscht alle **nil** Zeiger der Kollektion.

Siehe auch

Delete
DeleteAll
TCollection

TCollection.PutItem (Methode)

Syntax

```
procedure PutItem(var S: TStream; Item: Pointer); virtual;
```

Beschreibung

Wird von Store für jedes Element der Kollektion aufgerufen. Diese Methode kann überschrieben werden, sollte aber nicht direkt aufgerufen werden. PutItem setzt voraus, daß die Elemente der Kollektion Nachkommen von **TObject** sind und ruft deshalb **TStream.Put** auf, um die Elemente zu speichern:

```
S.Put(Item);
```

Siehe auch

GetItem
PutItem
Store
TCollection

TCollection.SetLimit (Methode)

Syntax

```
procedure SetLimit(ALimit: Integer); virtual;
```

Beschreibung

Vergrößert oder verkleinert die Kollektion durch Ändern der Größe auf ALimit. Wenn ALimit kleiner als Count ist, wird es auf Count gesetzt; wenn ALimit größer als MaxCollectionSize ist, wird es zu MaxCollectionSize. Wenn sich ALimit vom aktuellen Limit unterscheidet, wird ein neues Item-Array von ALimit Elementen angelegt, das alte Item-Array in das neue Array kopiert und danach entfernt.

Siehe auch

Count
Limit
TCollection

TCollection.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert die Kollektion mit allen Elementen in den Stream S. Store ruft PutItem für jedes Element der Kollektion auf.

Siehe auch

PutItem
TCollection

TComboBox (WObjects Unit)

TComboBox ist ein Schnittstellenobjekt, das ein Kombinationsfenster in Windows repräsentiert. TComboBox Objekte finden üblicherweise nicht in Dialogfeldern (**TDialog**) oder -fenstern (**TDlgWindow**) Verwendung, sondern als selbständige Kombinationsfenster, als untergeordnetes Fenster im Client-Bereich eines anderen Fensters. Kombinationsfenster erben die meisten ihrer Funktionen von **TListBox**.

Es gibt folgende Kombinationsfenster:
- einfache,
- mit verdeckter lokaler Liste und
- mit Eingabebeschränkung auf die Liste.

Die Art des Kombinationsfensters wird mit Hilfe der Windows-Konstanten cbs_Simple, cbs_DropDown und cbs_List bestimmt. Diese Konstanten werden an den Konstruktor Init weitergegeben, der Windows instruiert, welche Fenster erstellt werden sollen.

Felder

TextLen

Methoden

Init
InitResource
Load
GetClassName
HideList
ShowList
Store
Transfer

TComboBox.TextLen (Feld)

Syntax

```
TextLen: Word;
```

Beschreibung

TextLen enthält die Länge des Zeichenpuffers im Editierteil des Kombinationsfensters, zugleich ist dies die Anzahl des Bytes, die von **Transfer** übertragen wurden.
TextLen wird von **Init** gesetzt.

Siehe auch

TComboBox

TComboBox.IsList (Feld)

Syntax

```
IsList: Boolean;
```
(Read/write)

Beschreibung

IsList ist True bei einem Kombinationsfenster mit verdeckter lokaler Liste, False bei einem einfachen Kombinationsfenster oder einem mit Eingabebeschränkung.

Siehe auch

TComboBox

TComboBox.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnID: Integer; X,Y,W,H: Integer;
CanDropList: Boolean; HasStaticControl: Boolean);
```

Beschreibung

Konstruiert durch Aufruf von **TListBox.Init** ein Kombinationsfenster mit
- vererbtem übergeordneten Fenster (AParent),
- Identifikationsnummern (AnId);
- Position (X, Y) relativ zum Client-Bereich des übergeordneten Fensters;
- mit der Breite (W);
- Höhe (H);

Init ruft **TControl.Init** auf. Wenn CanDropList False ist, handelt es sich um ein einfaches Kombinationsfenster, cbs_Simple wird ins Feld Attr.Style eingetragen. Wenn CanDropList True, aber HasStaticControl False ist, hat das Kombinationsfenster eine verdeckte lokale Liste, und cbs_DropDown wird ins Feld Attr.Style eingetragen. Sind beide Booleschen Parameter True, wird für die Liste mit Eingabebeschränkung cbs_DropDownList ins Feld Attr.Style eingetragen. Weiterhin wird cbs_Sort in Attr.Style eingetragen, damit die Liste sortiert wird, und **ws_VScroll**, damit die Liste eine vertikale Bildlaufleiste erhält.

Siehe auch

cbs_ Combo box styles
TComboBox

TComboBox.InitResource (Methode)

Syntax

```
constructor InitResource(AParent: PWindowsObject; ResourceID: Integer;
ATextLen: Word);
```

Beschreibung

Assoziiert das Kombinationsfenster-Objekt mit der Ressource aus ResourceID, mit einer maximalen Textlänge von TextLen -1.

Siehe auch

TComboBox

TComboBox.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Erzeugt und lädt ein Kombinationsfenster vom Stream Constructs S durch Aufruf von TListBox.Load und nachfolgendem Einlesen der weiteren Felder (IsDropDown, IsList) von TComboBox.

Siehe auch

TComboBox
TWindow.Load

TComboBox.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt den Namen der Fensterklasse von TComboBox, 'ComboBox', zurück.

Siehe auch

TComboBox

TComboBox.HideList (Methode)

Syntax

```
procedure HideList; virtual;
```

Beschreibung

Bewirkt das Verdecken der lokalen Liste des entsprechenden Kombinationsfensters.

Siehe auch

TComboBox

TComboBox.ShowList (Methode)

Syntax

```
procedure ShowList; virtual;
```

Beschreibung

Bewirkt das Anzeigen der lokalen Liste des entsprechenden Kombinationsfensters.

Siehe auch

TComboBox

TComboBox.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert das Kombinationsfenster auf den Stream S, indem es TListBox.Store aufruft und zusätzlich das Feld TextLen von TComboBox schreibt.

Siehe auch

TComboBox
TWindow.Store

TComboBox.Transfer (Methode)

Syntax

```
function Transfer(DataPtr: Pointer; TransferFlag: Word): Word; virtual;
```

Beschreibung

Überträgt Daten zu oder von dem Record, auf den DataPtr zeigt.
Der Record sollte zuerst ein Zeiger auf eine String-Kollektion sein, die die Einträge der Kombinationsfenster-Liste enthält. Dann wird ein Zeichenarray mit dem aktuell gewählten Eintrag übertragen.
- Ist TransferFlag gleich tf_GetData, werden die Daten des Kombinationsfensters zum DataPtr-Record übertragen. Transfer gibt die Anzahl der übertragenen Bytes zurück.
- Ist TransferFlag gleich tf_SetData, werden die Daten des Records zum Kombinationsfenster übertragen. Transfer gibt die Anzahl der übertragenen Bytes zurück.
- Ist TransferFlag gleich tf_SizeData, gibt Transfer die Anzahl der zu übertragenden Bytes zurück.

Siehe auch

TComboBox

TControl (Unit WObjects)

TControl ist ein abstrakter Objekttyp, der als Vorfahre alle Dialogelement-Objekttypen wie **TScrollBar** und **TButton** vereint.

Er ist auch Vorfahr von **TMDIClient**, einem speziellen Element für MDI-Programme.

Dialogelement-Objekte werden üblicherweise nicht in Dialogfeldern (**TDialog**) oder -fenstern (**TDlgWindow**) verwendet, sondern als selbständige Elemente im Client-Bereich eines anderen Fensters.

Methoden

Init
GetClassName
InitResource
Register
WMPaint

TControl.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnId: Integer; ATitle: PChar;
X,Y,W,H: Integer);
```

Beschreibung

Konstruiert ein Dialogelement-Objekt mit vererbtem
- übergeordneten Fenster (AParent),
- Identifikationsnummern (AnId);
- Text (ATitle);
- Position (X, Y), relativ zum Client-Bereich des übergeordneten Fensters; mit der
- Breite (W); Höhe (H).

Mit diesen Parametern wird das von **TWindow** geerbte Feld Attr versehen. Als Vorgabe wird Attr.Style auf ws_Child oder ws_Visible oder ws_Group oder ws_TabStop gesetzt, so daß alle Dialogelement-Objekte sichtbare untergeordnete Fenster sind.

Siehe auch

TControl
ws_Window styles

TControl.InitResource (Methode)

Syntax

```
constructor InitResource(AParent: PWindowsObject; ResourceID: Word);
```

Beschreibung

Assoziiert ein Dialogelement-Objekt mit einem Steuerelement aus einer Dialogfensterdefinition. Ruft **TWindow.InitResource** und EnableTransfer auf, um Transfermechanismen für abhängige Elemente einzurichten.

Siehe auch

TControl
TWindowsObject.EnableTransfer

TControl.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Abstrakte Methode, die von abgeleiteten Objekten immer überschrieben wird.

Siehe auch

TControl

TControl.Register (Methode)

Syntax

```
function Register: Boolean; virtual;
```

Beschreibung

Liefert True, um anzuzeigen, daß Nachkommen von TControl schon registrierte Fensterklassen verwenden.

Siehe auch

TControl

TControl.WMPaint (Methode)

Syntax

```
procedure WMPaint(var Msg: TMessage); virtual wm_First + wm_Paint;
```

Beschreibung

Ruft DefWndProc zum Restaurieren von Dialogelement-Objekten auf.

Siehe auch

TControl

TDialog (Unit WObjects)

TDialog definiert Objekte, die modalen und nichtmodalen Dialogfenstern für Windows-Programme entsprechen. Zu einem TDialog-Objekt gehört eine Dialogfenster-Ressource, die Erscheinung und Lage seiner Elemente bestimmt. Diese Ressource wird im Aufruf von Init benannt.

Dialogfenster können modale und nichtmodale Dialogelemente haben je nachdem, ob sie Execute oder Create aufrufen. Beachten Sie, daß ein modaler Dialog die Operation des übergeordneten Fensters unterbricht.

Felder

Attr
DialogProc
IsModal

Methoden

Init
Load
Done
Cancel
Create
DefWndProc
EndDlg
Execute
GetItemHandle
Ok
SendDlgItemMsg
Store
WMInitDialog
WMClose

TDialog.Attr (Feld)

Syntax

```
Attr: TDialogAttr;
```

Beschreibung

Attr enthält die Attribute des Dialogfensters. Attr ist so definiert:

```
TDialogAttr = record
   Name: PChar;
   Param: Longint;
end;
```

Das Feld Name enthält den Namen oder die Identifikationsnummer der Dialogfenster-Ressource. Das Feld Param enthält einen Parameter, der beim Erstellen des Dialogfensters an die Dialogprozedur weitergegeben wird.

Siehe auch

TDialog

TDialog.DialogProc (Feld)

Syntax

```
DialogProc: TFarProc;
```
(Read only)

Beschreibung

DialogProc zeigt auf die Prozedurinstanz-Adresse der Dialogfunktion.

Siehe auch

TDialog

TDialog.IsModal (Feld)

Syntax

```
IsModal: Boolean;
```
(Read only)

Beschreibung

IsModal ist True, wenn der Dialog modal, und False, wenn er nichtmodal ist.

Siehe auch

TDialog

TDialog.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AName: PChar);
```

Beschreibung

Erzeugt das Dialogobjekt durch Aufruf von **TWindowsObject.Init**, an welches das übergeordnete Fenster weitergegeben wird, AParent.

Init setzt das Feld Attr.Name auf den String aus AName. Der String kann ein Symbolbezeichner wie 'EMPLOYEEINFO' sein oder eine zum Typ PChar umgewandelte Integer-Identifikationsnummer, z.B. PChar(120).

Init ruft **TWindowsObject.DisableAutoCreate** auf, damit Dialoge nicht automatisch mit ihren Übergeordneten Fenstern erzeugt und dargestellt werden.

Siehe auch

TDialog

TDialog.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Erzeugt und lädt ein Dialogfenster aus dem Stream S durch Aufruf von **TWindowsObject.Load** und Lesen der weiteren Felder (Attr und IsModal) von TDialog.

Siehe auch

TDialog

TDialog.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Entfernt das Dialogfenster-Objekt mit Aufruf von **TWindowsObject.Done**.

Siehe auch

TDialog

TDialog.Cancel (Methode)

Syntax

```
procedure Cancel(var Msg: TMessage); virtual id_First + id_Cancel;
```

Beschreibung

Reagiert auf den Aktionsschalter Cancel durch Aufruf von EndDlg mit dem Wert id_Cancel.

Siehe auch

EndDlg
TDialog

TDialog.Create (Methode)

Syntax

```
function Create: Boolean; virtual;
```

Beschreibung

Erzeugt ein Dialogelement für ein Dialogobjekt ohne Modus. Create gibt im Erfolgsfall True zurück. Ansonsten ruft es Error mit dem Fehlercode em_InvalidWindow auf.

Siehe auch

Execute
TDialog

TDialog.DefWndProc (Methode)

Syntax

```
procedure DefWndProc(var Msg: TMessage); virtual;
```

Beschreibung

Setzt das Feld Result von Msg auf 0 und aktiviert damit die Standard-Botschaftsbearbeitung von Windows.

Siehe auch

TDialog

TDialog.EndDlg (Methode)

Syntax

```
procedure EndDlg(ARetValue: Integer); virtual;
```

Beschreibung

Löscht modale und nichtmodale Dialogfenster. ARetValue ist als Wert für modale Dialogfenster von Execute zurückgegeben worden.

Siehe auch

Create
Execute
TDialog

TDialog.Execute (Methode)

Syntax

```
function Execute: Integer; virtual;
```

Beschreibung

Erzeugt ein Dialogelement für ein modales Dialogobjekt. Diese Methode läuft während der gesamten Zeit, in der das Dialogfenster sichtbar ist, bis EndDlg aufgerufen wird.

Vor dem Ende der Methode wird HWindow auf 0 gesetzt. Execute gibt im Erfolgsfall den Integerwert von EndDlg zurück. Ansonsten ruft es Error mit dem Fehlercode em_InvalidWindow auf.

Siehe auch

Create
EndDlgTDialogEndDlg
TDialogTDialog

TDialog.GetItemHandle (Methode)

Syntax

```
function GetItemHandle(DlgItemID: Integer): HWnd;
```

Beschreibung

Gibt das Handle des Dialogelements mit der Identifikationsnummer aus DlgItemID zurück.

Siehe auch

TDialog

TDialog.Ok (Methode)

Syntax

```
procedure Ok(var Msg: TMessage); virtual id_First + id_OK;
```

Beschreibung

Reagiert auf den Aktionsschalter OK durch Aufruf von CanClose und EndDlg mit dem Wert id_OK. Ruft außerdem **TransferData** auf, um Transferdaten von den Elementen in den Transferpuffer zu schreiben.

Siehe auch

EndDlg
TDialog

TDialog.SendDlgItemMsg (Methode)

Syntax

```
function SendDlgItemMsg(DlgItemID: Integer; AMsg, WParam: Word; LParam: DWORD): Longint;
```

Beschreibung

Schickt die Windows-Botschaft AMsg an das Dialogelement mit der Identifikationsnummer DlgItemID. WParam und LParam sind Parameter der Meldung.

SendDlgItemMsg gibt den Wert zurück, den das Element zurückgab, oder 0, falls die Identifikationsnummer ungültig war.

Siehe auch

TDialog

TDialog.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert das Dialogfenster im Stream S, indem es **TWindowsObject.Store** aufruft und danach die weiteren Felder (Attr und IsModal) von TDialog schreibt.

Siehe auch

TDialog

TDialog.WMClose (Methode)

Syntax

```
procedure WMClose(var Msg: TMessage); virtual wm_First + wm_Close;
```

Beschreibung

Bei modalen Dialogfenstern ruft diese Mehode **EndDlg** zum Schließen des Fensters auf. Ist das Dialogfenster nicht modal, ruft sie die Methode **WMClose** auf, die sie von TWindowsObject geerbt hat.

Siehe auch

TDialog

TDialog.WMInitDialog (Methode)

Syntax

```
procedure WMInitDialog(var Msg: TMessage); virtual wm_First +
wm_InitDialog;
```

Beschreibung

WMInitDialog wird vor dem Anzeigen des Dialogfensters automatisch aufgerufen und ruft SetupWindow auf, um das Dialogfenster und seine Elemente zu initialisieren.

Siehe auch

TDialog
TWindowsObject.SetupWindow

TDlgWindow (Unit WObjects Unit)

Dialogfenster, die durch TDlgWindow definiert werden, vereinen Eigenschaften von normalen Dialogfenstern (TDialog) und Fenstern.

Ein Dialogfenster hat wie ein Dialogelement eine Dialog-Ressource, die Erscheinungsbild und Position der Elemente bestimmt. Wie beim Fenster gibt es eine Fensterklasse, die Symbole und die Form des Cursors angeben kann.

Verwenden Sie ausschließlich die Methode Create (diese erzeugt nichtmodale Dialogfenster). Verwenden Sie nicht die Methode Execute.

Methoden

- **Init**
- **Create**
- **GetWindowClass**

TDlgWindow.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AName: PChar);
```

Beschreibung

Erzeugt ein neues TDlgWindow-Objekt durch Aufruf von **TDialog.Init**.

Ruft auch EnableAutoCreate auf, damit es als untergeordnetes Fenster automatisch zusammen mit dem übergeordneten Fenster erzeugt und dargestellt wird.

Siehe auch

TDlgWindow
TWindowsObject.EnableAutoCreate

TDlgWindow.Create (Methode)

Syntax

```
function Create: Boolean; virtual;
```

Beschreibung

Registriert die Klasse des Dialogfensters und ruft **TDialog.Create** auf. Create liefert im Erfolgsfall True zurück.

Siehe auch

TDlgWindow
TWindowsObject.Register

TDlgWindow.GetWindowClass (Methode)

Syntax

```
procedure GetWindowClass(var AWndClass: TWndClass); virtual;
```

Beschreibung

Definiert die Vorgabe-Fensterklasse und gibt den Record in AWndClass zurück. Diese Fensterklasse hat kein Menü, Standard-Icon und -Cursor.

Definieren Sie GetWindowClass sowie GetClassName für Nachkommen von TDlgWindow um. Ihr neues GetWindowClass muß aber vor der Veränderung irgendwelcher TWndClass-Felder das alte (dieses) aufruften.

Siehe auch

TDlgWindowTDlgWindow

TDosStream **(Unit WObjects)**TDosStream ist ein spezieller Nachkomme von **TStream**, der ungepufferte DOS-Datei-Streams implementiert. Der Konstruktor ermöglicht, eine DOS-Datei zu erzeugen oder zu öffnen, indem der Dateiname und ein Zugriffsmodus genannt wird: stCreate, stOpenRead, stOpenWrite, oder stOpen. Das zusätzliche Feld von TDosStream ist Handle, das DOS-Datei-Handle, mit welchem auf offene Dateien zugegriffen wird. Die meisten Programme verwenden den gepufferten Nachkommen von TDosStream namens **TBufStream**. TDosStream überschreibt alle abstrakten Methoden von TStream außer **TStream.Flush**.

Felder

Handle

Methoden

Init
Done
GetPos
GetSize
Read
Seek
Truncate
Write

TDosStream.Handle (Feld)

Syntax

```
Handle: Word
```
(Read only)

Beschreibung

Handle ist das bekannte DOS-Datei-Handle, mit welchem auf offene Dateien zugegriffen wird.

Siehe auch

TDosStream

TDosStream.Init (Methode)

Syntax

```
constructor Init(FileName: FNameStr; Mode: Word);
```

Beschreibung

Erzeugt einen DOS-Datei-Stream mit gegebenem FileName und Zugriffsmodus. Bei Erfolg erhält das Feld Handle das DOS-Datei-Handle. Bei Mißerfolg wird Error mit dem Argument stInitError aufgerufen.

Das Argument Mode muß einen folgender Werte erhalten: stCreate, stOpenRead, stOpenWrite, oder stOpen.

Siehe auch

TDosStream
stXXXX-Konstanten

TDosStream.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Schließt und entfernt den DOS-Datei-Stream.

Siehe auch

Init
TDosStream

TDosStream.GetPos (Methode)

Syntax

```
function GetPos: Longint; virtual;
```

Beschreibung

Liefert die aktuelle Position des aufrufenden Stream.

Siehe auch

Seek
TDosStream

TDosStream.GetSize (Methode)

Syntax

```
function GetSize: Longint; virtual;
```

Beschreibung

Liefert die Gesamtgröße des aufrufenden Stream in Bytes.

Siehe auch

TDosStream

TDosStream.Read (Methode)

Syntax

```
procedure Read(var Buf; Count: Word); virtual;
```

Beschreibung

Liest Count Bytes in den Puffer Buf ab der aktuellen Position des aufrufenden Stream.

Siehe auch

Write
stXXXX-KonstantenX
TDosStream

TDosStream.Seek (Methode)

Syntax

```
procedure Seek(Pos: Longint); virtual;
```

Beschreibung

Setzt die aktuelle Position Pos Bytes vom Anfang des aufrufenden Stream.

Siehe auch

GetPos
GetSize
TDosStream

TDosStream.Truncate (Methode)

Syntax

```
procedure Truncate; virtual;
```

Beschreibung

Löscht alle Daten auf dem aufrufenden Stream ab der aktuellen Position bis zum Ende.

Siehe auch

GetPos
Seek
TDosStream

TDosStream.Write (Methode)

Syntax

```
procedure Write(var Buf; Count: Word); virtual;
```

Beschreibung

Schreibt Count Bytes vom Puffer Buf in den aufrufenden Stream, beginnend bei der aktuellen Position.

Siehe auch

Read
stXXXX-Konstanten
TDosStream

TEdit (WObjects Unit)

TEdit ist ein Schnittstellenobjekt, das ein entsprechendes Editierfeld repräsentiert.

TEdit-Objekte werden üblicherweise nicht in Dialogfenstern (**TDialog** oder -**TDlgWindow**) verwendet, sondern als selbständige Editierfelder, als untergeordnetes Fenster im Client-Bereich eines anderen Fensters.

Es gibt einzeilige und mehrzeilige Editierfelder. Mehrzeilige Editierfelder ermöglichen vertikale Bildlaufleisten und die Bearbeitung mehrerer Zeilen. Die meisten Methoden von TEdit verwalten den Text des Editierfelds. Daneben enthält TEdit Methoden zur Beantwortung von Befehlsbotschaften. Diese bearbeiten Menüauswahlen des übergeordneten Fensters wie Ausschneiden, Kopieren, Einfügen, Löschen und Rückgängig Machen.

Zwei wichtige vom Vorfahren **TStatic** geerbte Methoden sind GetText und SetText.

Methoden

Init
CanUndo
ClearModify
CMEditClear
CMEditCopy
CMEditCut
CMEditDelete
CMEditPaste
CMEditUndo
Copy
Cut
DeleteLine
DeleteSelection
DeleteSubText
GetClassName
GetLine
GetLineFromPos
GetLineIndex
GetLineLength
GetNumLines
GetSelection
GetSubText
Insert
IsModified
Paste
Scroll
Search
SetSelection
SetupWindow
Transfer
Undo

TEdit.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnId: Integer; ATitle: PChar;
X,Y,W,H: Integer; Multiline : Boolean);
```

Beschreibung

Erzeugt ein Editierfeld mit
- übergeordnetem Fenster (AParent) und füllt das Feld Attr mit
- Steuerelement-ID AnId),
- Titel (ATitle),
- Position (X, Y), relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters
- Breite (W) und
- Höhe (H).

Wenn Multiline True ist, ist das Editierfeld mehrzeilig und enthält horizontale und vertikale Bildlaufleisten. In diesem Fall enthält das Feld Attr.Style die Windows-Stilkonstanten es_Multiline, es_AutoVScroll, es_AutoHScroll, es_Left, ws_VScroll, und ws_HScroll. Ist Multiline False, so ist das Editierfeld einzeilig, umrandet (**ws_Border**) und linksbündig (es_Left).

Siehe auch

es_Edit control styles
TEdit

TEdit.CanUndo (Methode)

Syntax

```
function CanUndo: Boolean; virtual;
```

Beschreibung

Gibt True zurück, wenn der letzte Editiervorgang rückgängig gemacht werden kann.

Siehe auch

TEdit
Undo

TEdit.CMEditClear (Methode)

Syntax

```
procedure CMEditClear(var Msg: TMessage); virtual cm_First + cm_EditClear;
```

Beschreibung

Beantwortet die Auswahl einer Menüoption mit der Menü-ID cm_EditClear mit einem Aufruf der Methode Clear.

Siehe auch

TEdit
TStatic.Clear

TEdit.GetLineIndex (Methode)

Syntax

```
function GetLineIndex(LineNumber: Integer): Integer; virtual;
```

Beschreibung

Gibt von einem mehrzeiligen Editierfeld die Zeichenzahl vor der Zeile LineNumber zurück. Ein Zeilenumbruch zählt als zwei Zeichen. Existiert die angegebene Zeile nicht, gibt GetLineIndex die Gesamtzahl der Zeichen im Editierfeld zurück.

Siehe auch

TEdit

TEdit.Copy (Methode)

Syntax

```
procedure Copy; virtual;
```

Beschreibung

Kopiert den markierten Text in die Zwischenablage.

Siehe auch

CMEditCopy
TEdit

TEdit.CMEditCopy (Methode)

Syntax

```
procedure CMEditCopy(var Msg: TMessage); virtual cm_First + cm_EditCopy;
```

Beschreibung

Beantwortet die Auswahl einer Menüoption mit der Menü-ID cm_EditCopy mit einem Aufruf der Methode Copy.

Siehe auch

Copy
TEdit

TEdit.Cut (Methode)

Syntax

```
procedure Cut; virtual;
```

Beschreibung

Schneidet den markierten Text aus, d.h. kopiert ihn in die Zwischenablage und löscht ihn.

Siehe auch

CMEditCut
TEdit

TEdit.CMEditCut (Methode)

Syntax

```
procedure CMEditCut(var Msg: TMessage); virtual cm_First + cm_EditCut;
```

Beschreibung

Beantwortet die Auswahl einer Menüoption mit der Menü-ID cm_EditCut mit einem Aufruf der Methode Cut.

Siehe auch

Cut
TEdit

TEdit.CMEditDelete (Methode)

Syntax

```
procedure CMEditDelete(var Msg: TMessage); virtual cm_First +
cm_EditDelete;
```

Beschreibung

Beantwortet die Auswahl einer Menüoption mit der Menü-ID cm_EditDelete mit einem Aufruf der Methode DeleteSelection.

Siehe auch

DeleteSelection
TEdit

TEdit.DeleteLine (Methode)

Syntax

```
function DeleteLine(LineNumber: Integer): Boolean; virtual;
```

Beschreibung

Löscht den Text in der Zeile LineNumber eines mehrzeiligen Editierfelds. DeleteLine löscht den Zeilenumbruch nicht und berührt keine anderen Zeilen. Bei Erfolg wird True zurückgegeben.

Siehe auch

TEdit

TEdit.DeleteSelection (Methode)

Syntax

```
function DeleteSelection: Boolean; virtual;
```

Beschreibung

Löscht den selektierten Text und gibt False zurück, wenn kein Text markiert war.

Siehe auch

CMEditDelete
TEdit

TEdit.DeleteSubText (Methode)

Syntax

```
function DeleteSubText(StartPos, EndPos: Integer): Boolean; virtual;
```

Beschreibung

Löscht den Text zwischen StartPos und EndPos. Das erste Zeichen steht an Position Null. Bei Mehrzeileneditierfeldern gehen die Positionsnummern forlaufend über alle Zeilen, ein Zeilenumbruch zählt als zwei Zeichen. DeleteSubText liefert im Erfolgsfall True.

Siehe auch

TEdit

TEdit.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt 'Edit' zurück, die Bezeichnung der Fensterklasse von TEdit.

Siehe auch

TEdit

TEdit.GetLine (Methode)

Syntax

```
function GetLine(ATextString: PChar; StrSize, LineNumber: Integer):
Boolean; virtual;
```

Beschreibung

Ruft den Text eines mehrzeiligen Editierfelds der Zeile LineNumber ab und gibt ihn in ATextString aus. StrSize zeigt die Anzahl der Zeichen an. GetLine gibt False zurück, wenn der Text nicht abgerufen werden kann oder zu lang ist.

Siehe auch

GetNumLines
GetLineLength
TEdit
TStatic.GetText

TEdit.GetSubText (Methode)

Syntax

```
procedure GetSubText(ATextString: PChar; StartPos, EndPos: Integer);
virtual;
```

Beschreibung

Holt in ATextString den Text eines Editierfelds vom Index StartPos bis EndPos. Das erste Zeichen steht an Position Null. Bei Mehrzeileneditierfeldern laufen die Positionsnummern über alle Zeilen. Ein Zeilenumbruch zählt als zwei Zeichen.

Siehe auch

GetSelection
TEdit

TEdit.GetSelection (Methode)

Syntax

```
procedure GetSelection(var StartPos, EndPos: Integer); virtual;
```

Beschreibung

Holt Anfang und Ende des markierten Texts und gibt sie in den Argumenten StartPos und EndPos zurück. In diesem Bereich ist das erste Zeichen auf Position Null. Bei mehrzeiligen Editierfeldern läuft der Index über alle Zeilen. Ein Zeilenumbruch zählt als zwei Zeichen. Bei Verwendung von GetSelection zusammen mit GetSubText erhält man den aktuell markierten Text.

Siehe auch

GetSubText
TEdit

TEdit.IsModified (Methode)

Syntax

```
function IsModified: Boolean; virtual;
```

Beschreibung

Gibt True zurück, wenn der Text des Editierfelds geändert wurde.

Siehe auch

ClearModify
TEdit

TEdit.Insert (Methode)

Syntax

```
procedure Insert(ATextString: PChar); virtual;
```

Beschreibung

Fügt den in ATextString übergebenen Text an der Cursorposition ein und ersetzt gegebenenfalls den markierten Text. Insert ähnelt Paste, umgeht aber die Zwischenablage.

Siehe auch

Paste
TEdit

TEdit.GetLineFromPos (Methode)

Syntax

```
function GetLineFromPos(CharPos: Integer): Integer; virtual;
```

Beschreibung

Gibt von einem mehrzeiligen Editierfeld die Zeile zurück, in der das Zeichen an CharPos steht. Der Index des ersten Zeichens ist Null, die Indices laufen über alle Zeilen. Ein Zeilenumbruch zählt als zwei Zeichen.

Siehe auch

TEdit

TEdit.GetLineLength (Methode)

Syntax

```
function GetLineLength(LineNumber: Integer): Integer; virtual;
```

Beschreibung

Gibt bei einem mehrzeiligen Editierfeld die Zeichenzahl in der Zeile LineNumber zurück. GetGetLineLength sollte vor GetLine aufgerufen werden.

Siehe auch

GetLine
TEdit

TEdit.GetNumLines (Methode)

Syntax

```
function GetNumLines: Integer; virtual;
```

Beschreibung

Gibt bei einem mehrzeiligen Editierfeld die Anzahl der eingegebenen Zeilen zurück. GetNumLines sollte vor GetLine aufgerufen werden.

Siehe auch

GetLine
TEdit

TEdit.Paste (Methode)

Syntax

```
procedure Paste; virtual;
```

Beschreibung

Fügt Text aus der Zwischenablage an der Cursorposition in das Editierfeld ein.

Siehe auch

CMEditPaste
TEdit

TEdit.CMEditPaste (Methode)

Syntax

```
procedure CMEditPaste(var Msg: TMessage); virtual cm_First + cm_EditPaste;
```

Beschreibung

Beantwortet die Auswahl einer Menüoption mit der Menü-ID cm_EditPaste mit einem Aufruf der Methode Paste.

Siehe auch

Paste
TEdit

TEdit.ClearModify (Methode)

Syntax

```
procedure ClearModify; virtual;
```

Beschreibung

Setzt das Änderungs-Flag im Editierfeld zurück.

Siehe auch

IsModified
TEdit

TEdit.SetSelection (Methode)

Syntax

```
function SetSelection(StartPos, EndPos: Integer): Boolean; virtual;
```

Beschreibung

Selektiert den Text zwischen StartPos und EndPos, ohne das Zeichen an EndPos selbst. Das erste Zeichen steht an Position Null und der Index läuft über alle Zeilen. Ein Zeilenumbruch zählt als zwei Zeichen.

Siehe auch

TEdit

TEdit.Search (Methode)

Syntax

```
procedure Search(StartPos: Integer; AText: PChar; CaseSensitive: Boolean):
Integer;
```

Beschreibung

Search durchsucht den Text des Editierfelds ab der Position StartPos, bis es AText findet.

- Wird Text, der mit AText übereinstimmt, gefunden, so wird er selektiert und Search gibt die Position, wo er beginnt, zurück.
- Wenn er nicht gefunden wird gibt Search -1 zurück.

Um die Suche ab der aktuellen Position zu beginnen, übergibt man -1 in StartPos.

Siehe auch

TEdit

TEdit.Scroll (Methode)

Syntax

```
procedure Scroll(HorizontalUnit, VerticalUnit: Integer); virtual;
```

Beschreibung

Rollt den Text in einem mehrzeiligen Editierfeld um die Zeichenzahl HorizontalUnit und die Zeilenzahl VerticalUnit. Positive Werte verschieben nach rechts bzw. unten, negative nach links bzw. oben.

Siehe auch

TEdit

TEdit.Transfer (Methode)

Syntax

```
function Transfer(DataPtr: Pointer; TransferFlag: Word): Word; virtual;
```

Beschreibung

Überträgt TextLen Zeichen des aktuellen Textes im Editierfeld an oder von der Adresse in DataPtr. Wenn TransferFlag gleich tf_GetData ist, wird vom Speicher gelesen. Wenn TransferFlag gleich tf_SetData ist, wird der Text an die Speicheradresse geschrieben. Transfer gibt in TextLen die gelesene oder geschriebene Anzahl Bytes zurück. Wenn TransferFlag gleich tf_SizeData ist, gibt Transfer die Größe in Bytes der Transferdaten zurück.

Siehe auch

TEdit

TEdit.Undo (Methode)

Syntax

```
procedure Undo; virtual;
```

Beschreibung

Macht die letzte Änderung rückgängig.

Siehe auch

CanUndo
CMEditUndo
TEdit

TEdit.CMEditUndo (Methode)

Syntax

```
procedure CMEditUndo(var Msg: TMessage); virtual cm_First + cm_EditUndo;
```

Beschreibung

Beantwortet die Auswahl einer Menüoption mit der Menü-ID cm_EditUndo mit einem Aufruf der Methode Undo.

Siehe auch

TEdit
Undo

TEdit.SetupWindow (Methode)

Syntax

```
procedure SetupWindow; virtual;
```

Beschreibung

Begrenzt die Anzahl der Zeichen, die in das Editierelement eingegeben werden können, mit einem Aufruf von TStatic.SetupWindow. Ist das Feld TextLen nicht Null, wird der Wert TextLen -1 durch **em_LimitText**-Botschaft an Windows gesendet.

Siehe auch

TEdit

TEmsStream (Unit WObjects)

TEmsStream ist ein Nachkomme von **TStream** zur Implementierung von Streams im EMS-Speicher. Die zusätzlichen Felder enthalten ein EMS-Handle, einen Seitenzähler, die Stream-Größe und die aktuelle Position. TEmsStream überschreibt alle abstrakten Methoden von TStream.Init und definiert einen speziellen Konstruktor und Destruktor.

Beim Debuggen eines Programms, das mit EMS-Streams arbeitet, kann die IDE den von diesem Programm benutzten EMS-Speicher nicht mehr freigeben, falls das Programm vorzeitig abbricht oder falls der Destruktor Done für einen EMS-Stream nicht aufgerufen wurde.
Nur die Methode Done (oder ein Neustart) gibt den EMS-Speicher, der vom Stream belegt wird, frei.

Felder

- **Handle**
- **PageCount**
- **TEmsStreamSize**
- **Position**
- **Size**

Methoden

- **Init**
- **Done**
- **GetPos**
- **GetSize**
- **Read**
- **Seek**
- **Truncate**
- **Write**

TEmsStream.Handle (Feld)

Syntax

```
Handle: Word;
```
(Read only)

Beschreibung

Das EMS-Handle für den Stream.

Siehe auch

TEmsStream

TEmsStream.PageCount (Feld)

Syntax

```
PageCount: Word;
```
(Read only)

Beschreibung

Die Anzahl der dem Stream zugewiesenen Seiten mit 16 KByte pro Seite.

Siehe auch

TEmsStream

TEmsStream.Size (Feld)

Syntax

```
Size: Longint;
```
(Read only)

Beschreibung

Größe des Stream in Bytes.

Siehe auch

TEmsStream

TEmsStream.Position (Feld)

Syntax

```
Position: Longint;
```
(Read only)

Beschreibung

Aktuelle Position im Stream, die erste Position ist 0.

Siehe auch

TEmsStream

TEmsStream.Init (Methode)

Syntax

```
constructor Init(MinSize, MaxSize: Longint);
```

Beschreibung

Erzeugt einen EMS-Stream mit den gegebenen Minimal- und Maximalgrößen in Bytes. Ruft TStream.Init auf und setzt Handle, Size und PageCount. Ruft Error mit einem Argument stInitError auf, wenn die Initialisierung scheitert.

Siehe auch

Done
TEmsStream
TObject.Init

TEmsStream.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Gibt den EMS-Stream und die belegten EMS-Seiten frei.

Siehe auch

Init
TEmsStream

TEmsStream.GetPos (Methode)

Syntax

```
function GetPos: Longint; virtual;
```

Beschreibung

Liefert die aktuelle Position des aufrufenden Stream.

Siehe auch

Seek
TEmsStream

TEmsStream.GetSize (Methode)

Syntax

```
function GetSize: Longint; virtual;
```

Beschreibung

Gibt die Gesamtgröße des aufrufenden Stream zurück.

Siehe auch

TEmsStream

TEmsStream.Read (Methode)

Syntax

```
procedure Read(var Buf; Count: Word); virtual;
```

Beschreibung

Liest Count Bytes in den Puffer Buf, beginnend an der aktuellen Position des aufrufenden Stream.

Siehe auch

stXXXX-Konstanten
TEmsStream
Write

TEmsStream.Seek (Methode)

Syntax

```
procedure Seek(Pos: Longint); virtual;
```

Beschreibung

Setzt die aktuelle Position auf Pos Bytes vom Beginn des aufrufenden Stream.

Siehe auch

GetPos
GetSize
TEmsStream

TEmsStream.Truncate (Methode)

Syntax

```
procedure Truncate; virtual;
```

Beschreibung

Löscht alle Daten auf dem aufrufenden Stream ab der aktuellen Position bis zum Ende. Die aktuelle Position wird auf das neue Ende des Stream gesetzt.

Siehe auch

GetPos
Seek
TEmsStream

TEmsStream.Write (Methode)

Syntax

```
procedure Write(var Buf; Count: Word); virtual;
```

Beschreibung

Schreibt Count Bytes vom Puffer Buf ab der aktuellen Position in den aufrufenden Stream.

Siehe auch

GetPos
Seek
TEmsStream
TStreamRead

TGroupBox WObjects unit)

TGroupBox ist ein Schnittstellen-Objekt, das eine entsprechende Gruppe repräsentiert. TGroupBox Objekte werden normalerweise nicht in Dialogfenstern (**TDialog** oder **TDlgWindow**) verwendet, sondern dann, wenn man eine selbständige Gruppe im Client-Bereich eines anderen Fensters darstellen will.

Gruppenfenster haben auf dem Bildschirm lediglich die Funktion, Wahlfelder optisch zusammenzufassen. Im Hintergrund verwalten sie jedoch den Zustand der Wahlfelder. Z.B. kann die Wahl eines Feldes die Rücknahme der Markierung aller anderen Felder bewirken.

Felder

NotifyParent

Methoden

Init
GetClassName
InitResource
Load
SelectionChanged
Store

TGroupBox.NotifyParent (field)

Syntax

```
NotifyParent: Boolean;
```
(Read/write)

Beschreibung

Flag, das anzeigt, ob das übergeordnete Fenster benachrichtigt werden soll, wenn sich der Status eines Wahlfelds im Gruppenfenster geändert hat.

Siehe auch

TGroupBox

TGroupBox.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnID: Integer; AText: PChar;
X,Y,W,H: Integer);
```

Beschreibung

Erzeugt ein Gruppen-Objekt mit
- übergeordnetem Fenster (AParent),
- Steuerelement-ID (AnId),
- Text (AText),
- Position (X, Y), relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters,
- Breite (W) und
- Höhe (H).

Ruft **TControl.Init** auf und fügt den Winndows-Stil **bs_GroupBox** zum Feld Attr.Style der Gruppe und entfernt den Stil **ws_TabStop** daraus. NotifyParent wird auf True gesetzt; damit wird als Vorgabe das übergeordnete Fenster benachrichtigt, wenn sich der Status eines Wahlfelds ändert.

Siehe auch

TGroupBox

TGroupBox.InitResource (Methode)

Syntax

```
constructor InitResource(AParent: PWindowsObject; ResourceID: Word);
```

Beschreibung

Klassifiziert die Gruppe durch Erzeugen eines ObjectWindows-Objekts, das einem Gruppenfenster-Element einer Ressourcendefinition entsprechen soll.

Ruft **TControl.InitResource** und **TWindowsObject.DisableTransfer** auf, um Gruppenfenster vom Transfermechanismus auszuschließen, da sie keine zu übertragenden Daten enthalten.

Siehe auch

TGroupBox

TGroupBox.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Erzeugt und lädt eine Gruppe vom Stream S durch Aufruf von TControl.Load und anschließendes Einlesen des zusätzlichen Felds (NotifyParent) von TGroupBox.

Siehe auch

TGroupBox
TWindow.Load

TGroupBox.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt 'Button', den Namen der Fensterklasse von TGroupBox, zurück.

Siehe auch

TGroupBox

TGroupBox.SelectionChanged (Methode)

Syntax

```
procedure SelectionChanged(ControlId: Integer); virtual;
```

Beschreibung

Wenn NotifyParent True ist, benachrichtigt diese Methode das übergeordnete Fenster der Gruppe durch eine Botschaft auf der Basis der ID untergeordneter Fenster, daß eine ihrer Auswahlen geändert hat. Diese Methode kann überschrieben werden, damit die Gruppe auf ihre Auswahlen reagieren kann.

Siehe auch

TGroupBox

TGroupBox.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert die Gruppe im Stream S durch Aufruf von TControl.Store und anschließendes Schreiben des weiteren Felds NotifyParent von TGroupBox.

Siehe auch

TGroupBox
TWindow.Store

TListBox (Unit WObjects)

TListBox ist ein Schnittstellen-Objekt, welches eine Liste repräsentiert. TListBox-Objekte werden normalerweise nicht in Dialogfenstern (**TDialog** oder **TDlgWindow**) verwendet, sondern um eine selbständige Liste im Client-Bereich eines anderen Fensters darzustellen.
Die Methoden von TListBox dienen auch den Instanzen des Nachkommens **TComboBox**.

Methoden

Init
AddString
ClearList
DeleteString
GetClassName
GetCount
GetMsgID
GetSelIndex
GetSelString
GetStringLen
GetString
InsertString
SetSelIndex
SetSelString
Transfer

TListBox.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnId: Integer; X,Y,W,H: Integer);
```

Beschreibung

Initialisiert ein Listenfenster-Objekt für das übergebene übergeordnete Fenster (AParent) mit
- Steuerelement-ID (AnId),
- Position (X, Y), relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters,
- Breite (W) und
- Höhe (H).

Ruft **TControl.Init** auf und fügt zum Feld Attr.Style die Windows-Stilkonstante **lbs_Standard** hinzu, die das Fenster mit folgenden Attributen ausstattet:

- Rand (**ws_Border**)
- Vertikale Bildlaufleiste (ws_VScroll)
- Automatische alphabetische Listensortierung (lbs_Sort)
- Übergeordnetes Fenster wird bei Auswahl informiert (lbs_Notify)

Diese Stilattribute können in einer abgeleiteten Klasse oder im Konstruktor Init des übergeordneten Fensterobjekts überschrieben werden.

Siehe auch

TListBox

TListBox.AddString (Methode)

Syntax

```
function AddString(AString: PChar): Integer; virtual;
```

Beschreibung

Fügt AString als Listenelement in das Listenfenster ein und gibt den Index der Elementposition, beginnend bei Null, zurück, oder im Fall eines Fehlers einen negativen Wert. Die Liste wird automatisch sortiert, außer **lbs_Sort** wurde aus dem Feld Attr.Style des Objekts vor dessen Erstellung entfernt.

Siehe auch

TListBox

TListBox.ClearList (Methode)

Syntax

```
procedure ClearList; virtual;
```

Beschreibung

Entfernt alle Listenelemente aus der Liste.

Siehe auch

TListBox

TListBox.DeleteString (Methode)

Syntax

```
function DeleteString(Index: Integer): Integer; virtual;
```

Beschreibung

Entfernt das Listenelement an der Position Index (beginnend bei Null). DeleteString gibt die Anzahl der restlichen Listenelemente oder im Fehlerfall einen negativen Wert zurück.

Siehe auch

TListBox

TListBox.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt den Namen der Fensterklasse **TListBox**, 'ListBox', zurück.

Siehe auch

TListBox

TListBox.GetCount (Methode)

Syntax

```
function GetCount: Integer; virtual;
```

Beschreibung

Gibt die Anzahl der Listenelemente oder im Fehlerfall einen negativen Wert zurück.

Siehe auch

TListBox

TListBox.GetSelIndex (Methode)

Syntax

```
function GetSelIndex: Integer; virtual;
```

Beschreibung

Gibt die Indexposition (beginnend bei Null) des aktuellen Listenelements zurück, oder einen negativen Wert, wenn keines ausgewählt ist.

Siehe auch

TListBox

TListBox.GetMsgID (Methode)

Syntax

```
function GetMsgID(AMsg: TMsgName): virtual;
```

Beschreibung

Überträgt Listenbotschaften zur Verwendung mit **TComboBox**-Objekten.

Siehe auch

TListBox

TListBox.GetSelString (Methode)

Syntax

```
function GetSelString(AString: PChar; MaxChars: Integer): Integer; virtual;
```

Beschreibung

Holt das aktuelle Listenelement in AString, wenn es nicht länger als MaxChars ist. GetSelString gibt die Stringlänge und im Fehlerfall einen negativen Wert zurück.

Siehe auch

TListBox

TListBox.GetString (Methode)

Syntax

```
function GetString(AString: PChar; Index: Integer): Integer; virtual;
```

Beschreibung

Holt in AString das Listenelement an der Position Index (beginnend bei Null) und gibt die Stringlänge und im Fehlerfall einen negativen Wert zurück.

Siehe auch

TListBox

TListBox.GetStringLen (Methode)

Syntax

```
function GetStringLen(Index: Integer): Integer; virtual;
```

Beschreibung

Gibt die Stringlänge des Listenelements an der Indexposition Index und im Fehlerfall einen negativen Wert zurück.

Siehe auch

TListBox

TListBox.InsertString (Methode)

Syntax

```
function InsertString(AString: PChar; Index: Integer): Integer; virtual;
```

Beschreibung

Fügt AString als Listenelement in das Listenfenster an der Position Index ein und gibt den Index der Elementposition, beginnend bei Null, zurück, oder im Fall eines Fehlers einen negativen Wert. Die Elemente des Listenfensters werden nicht neu sortiert. Falls Index -1 ist, wird der String am Ende der Liste angehängt.

Siehe auch

TListBox

TListBox.SetSelIndex (Methode)

Syntax

```
function SetSelIndex(Index: Integer): Integer; virtual;
```

Beschreibung

Bewirkt die Wahl des Listenelements an der Position Index. Falls Index -1 ist, werden etwaige Auswahlen gelöscht. Im Fehlerfall gibt SetSelIndex eine negative Zahl zurück.

Siehe auch

TListBox

TListBox.SetSelString (Methode)

Syntax

```
function SetSelString(AString: PChar; AIndex: Integer): Integer; virtual;
```

Beschreibung

Bewirkt die Wahl des ersten Listenelements, das mit dem Text von AString übereinstimmt und nach der Indexposition AIndex auftritt. Falls AIndex -1 ist, wird die Liste von Anfang durchsucht. SetSelString gibt die Indexdposition des neugewählten Elements oder im Fehlerfall einen negativen Wert zurück.

Siehe auch

TListBox

TListBox.Transfer (Methode)

Syntax

```
function Transfer(DataPtr: Pointer; TransferFlag: Word): Word; virtual;
```

Beschreibung

Überträgt die Liste der Einträge und der selektierten Elemente von bzw. zu dem Transfer-Record, auf den DataPtr zeigt.

- Falls TransferFlag auf tf_GetData steht, wird der gewählte Text an die Speicheradresse geschrieben.
- Steht TransferFlag auf tf_SetData, wird die Liste mit dessen Daten geladen. Transfer gibt in beiden Fällen die Anzahl der übertragenen Bytes zurück.
- Steht TransferFlag auf tf_SizeData, gibt Transfer die Anzahl der zu übertragenden Bytes zurück.

Der Record sieht unterschiedlich aus, je nachdem ob das Fenster die Auswahl mehrerer Elemente gestattet: Das erste übertragene Element ist immer ein Zeiger auf eine Kollektion von Strings, die der Inhalt des Listenfensters sind.

- Läßt die Liste nur die Auswahl eines Elements zu, folgt der Kollektion ein Integer-Index auf das selektierte Element.
- Läßt die Liste die gleichzeitige Auswahl mehrerer Elemente zu, folgt der Kollektion ein Zeiger auf einen **TMultiSelRec**-Record. Dieser enthält einen Integer-Array mit den Indizes der ausgewählten Elemente.

Siehe auch

TListBox

TMDIClient (Unit WObjects)

TMDIClient repräsentiert MDI- Client-Fenster. Das sind Steuerelemente, die untergeordnete MDI-Fenster einer MDI-Anwendung verwalten.

Felder

ClientAttr

Methoden

Init
Load
ArrangeIcons
CascadeChildren
GetClassName
Store
TileChildren

TMDIClient.ClientAttr (Feld)

Syntax

```
ClientAttr: TClientCreateStruct;
```

Beschreibung

ClientAttr enthält einen Record mit den Attributen des MDI-Client-Fensters. TClientCreateStruct ist folgendermaßen definiert:

```
type
   PClientCreateStruct = ^TClientCreateStruct;
   TClientCreateStruct = record
      hWindowMenu: THandle;
      idFirstChild: Word;
  end;
```

Siehe auch

TMDIClient

TMDIClient.Init (Methode)

Syntax

```
constructor Init(AParent: PMDIWindow);
```

Beschreibung

Initialisiert das MDI-Client-Fensterobjekt mit AParent als übergeordnetem Fenster. Init setzt die Felder von ClientAttr, ruft TControl.Init auf und fügt den Windows-Stil **ws_ClipChildren** zum Feld Attr.Style des Fensterobjekts hinzu. Zusätzlich entfernt Init das Client-Fenster aus der Liste des übergeordneten Fensters, damit es nicht wie andere untergeordnetes Fenster - etwa Listen und Schalter - behandelt wird.

Siehe auch

TMDIClient

TMDIClient.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Initialisiert und lädt ein MDI Client-Fenster vom Stream S durch Aufruf von TControl.Load und anschließendes Einlesen des zusätzlichen Felds ClientAttr von TMDIClient.

Siehe auch

TMDIClient
TWindow.Load

TMDIClient.ArrangeIcons (Methode)

Syntax

```
procedure ArrangeIcons; virtual;
```

Beschreibung

Richtet die Symbole der untergeordneten MDI-Fenster ordentlich in einer Reihe am unteren Rand des MDI_Client-Fensters aus.

Siehe auch

TMDIClient

TMDIClient.CascadeChildren (Methode)

Syntax

```
procedure CascadeChildren; virtual;
```

Beschreibung

Ordnet alle nicht verkleinerten untergeordneten MDI-Fenster im MDI-Client-Fenster so an, daß sie überlappen und alle Titelleisten sichtbar sind.

Siehe auch

TMDIClient

TMDIClient.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt den Namen der Fensterklasse von **TControl**, 'MDIClient', zurück.

Siehe auch

TMDIClient

TMDIClient.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert das MDI Client-Fenster in den Stream S, indem es erst TControl.Store aufruft und zusätzlich das Feld ClientAttr von TMDIClient schreibt.

Siehe auch

TMDIClient

TMDIClient.TileChildren (Methode)

Syntax

```
procedure TileChildren; virtual;
```

Beschreibung

Ordnet alle nicht verkleinerten untergeordneten MDI-Fenster so im MDI-Client-Fenster an, daß sie ohne Überlappen allen verfügbaren Raum einnehmen.

Siehe auch

TMDIClient

TMDIWindow (Unit WObjects)

TMDIWindow repräsentiert Rahmenfenster des Multiple Document Interface (MDI). Das sind überlappende Fenster, die als Hauptfenster einer MDI-Anwendung dienen.

Eine Haupteigenschaft von TMDIWindow-Objekten ist, daß sie ein **TMDIClient**-Objekt besitzen und es im ClientWnd-Feld speichern. Ein weiteres Merkmal ist ein Menü, das die untergeordneten Fenster verwaltet und automatisch anzeigt.

Felder

ChildMenuPos
ClientWnd

Methoden

Init
Load
Done
ArrangeIcons
CascadeChildren
CloseChildren
CMArrangeIcons
CMCascadeChildren
CMCloseChildren
CMCreateChild
CMTileChildren
CreateChild
DefWndProc
GetClassName
GetClient
GetWindowClass
InitChild
InitClientWindow
SetupWindow
Store
TileChildren

TMDIWindow.ChildMenuPos (Feld)

Syntax

```
ChildMenuPos: Integer;
```
(Read/write)

Beschreibung

ChildMenuPos ist ein Index mit der Position des Verwaltungsmenüs der untergeordneten Fenster. Der Index zählt nur Hauptmenüoptionen; das Element ganz oben links steht an Position Null.

Siehe auch

TMDIWindow

TMDIWindow.ClientWnd (Feld)

Syntax

```
ClientWnd: PMDIClient;
```
(Read only)

Beschreibung

ClientWnd zeigt auf das MDI-Client-Fenster des MDI-Rahmenfensters, einer Objektinstanz von **TMDIClient**.

Siehe auch

TMDIWindow

TMDIWindow.Init (Methode)

Syntax

```
constructor Init(ATitle: PChar; AMenu: HMenu);
```

Beschreibung

Initialisiert ein MDI Rahmenfenster mit dem Titel aus ATitle und dem Menü aus AMenu. Ein MDI-Rahmenfenster muß ein Menü haben und das Hauptfenster des Programms sein; es hat kein übergeordnetes Fenster. Als Vorgabe setzt Init ChildMenuPos auf Null, was bedeutet, daß das Menü der untergeordneten Fenster oben links steht. Um ChildMenuPos zu ändern, müssen Sie Init in abgeleiteten Typen überschreiben. Beispiel:

```
constructor MyMDIWindow.Init(ATitle: PChar; AMenu: HMenu);
begin
   TMDIWindow.Init(ATitle, AMenu);
   ChildMenuPos := 3;
end;
```

Siehe auch

InitClientWindow
TMDIWindow

TMDIWindow.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Initialisiert und lädt ein MDI-Rahmenfenster vom Stream S durch Aufruf von **TWindow.Load** und anschließendes Einlesen der zusätzlichen Felder ClientWnd und ChildMenuPos von TMDIWindow.

Siehe auch

TMDIWindow

TMDIWindow.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Entfernt das MDI-Objekt des untergeordneten Fensters, das in ClientWnd gespeichert ist, bevor **TWindow.Done** aufgerufen wird, um das Rahmenfensterobjekt zu entfernen.

Siehe auch

TMDIWindow

TMDIWindow.ArrangeIcons (Methode)

Syntax

```
procedure ArrangeIcons;
```

Beschreibung

Richtet die Symbole der untergeordneten MDI-Fenster ordentlich in einer Reihe am unteren Rand des MDI-Client-Fensters aus. Ruft ClientWnd^.ArrangeIcons auf.

Siehe auch

TMDIClient.ArrangeIcons
TMDIWindow

TMDIWindow.CMArrangeIcons (Methode)

Syntax

```
procedure CMArrangeIcons(var Msg: TMessage); virtual cm_First + cm_ArrangeIcons;
```

Beschreibung

Reagiert auf die Menüwahl mit der ID cm_ArrangeChildIcons durch Aufruf von ArrangeIcons.

Siehe auch

TMDIWindow

TMDIWindow.CascadeChildren (Methode)

Syntax

```
procedure CascadeChildren;
```

Beschreibung

Ordnet alle nichtverkleinerten untergeordneten MDI-Fenster im MDI-Client-Fenster so an, daß sie sich überlappen und alle Titelleisten sichtbar sind. Ruft ClientWnd^.CascadeChildren auf.

Siehe auch

TMIDClient.CascadeChildren
TMDIWindow

TMDIWindow.CMCascadeChildren (Methode)

Syntax

```
procedure CMCascadeChildren(var Msg: TMessage); virtual cm_First + cm_CascadeChildren;
```

Beschreibung

Reagiert auf die Menüwahl mit der ID cm_CascadeChildren durch Aufruf von **CascadeChildren**.

Siehe auch

TMDIWindow

TMDIWindow.CMCloseChildren (Methode)

Syntax

```
procedure CMCloseChildren(var Msg: TMessage); virtual cm_First +
cm_CloseChildren;
```

Beschreibung

Reagiert auf die Menüwahl mit der ID cm_CloseChildren durch Aufruf von **CloseChildren**.

Siehe auch

TMDIWindow

TMDIWindow.CloseChildren (Methode)

Syntax

```
procedure CloseChildren;
```

Beschreibung

Löscht alle untergeordneten MDI-Fenster, für die CanClose True zurückgibt.

Siehe auch

TMDIWindow

TMDIWindow.CreateChild (Methode)

Syntax

```
function CreateChild: PWindowsObject; virtual;
```

Beschreibung

Initialisiert ein neues untergeordnetes MDI-Fenster durch Aufruf von InitChild und MakeWindow. Im Gegensatz zu **InitChild** brauchen Sie CreateChild nicht zu überschreiben, um es auf abgeleitete untergeordnete Fenster anzupassen. CreateChild gibt einen Zeiger auf das neue untergeordnete MDI-Fenster zurück.

Siehe auch

TApplication.MakeWindow
TMDIWindow

TMDIWindow.CMCreateChild (Methode)

Syntax

```
procedure CMCreateChild(var Msg: TMessage); virtual cm_First +
cm_CreateChild;
```

Beschreibung

Reagiert auf die Menüwahl mit der ID cm_CreateChild durch Aufruf von **CreateChild**, um ein neues untergeordnetes Fenster zu erzeugen.

Siehe auch

TMDIWindow

TMDIWindow.DefWndProc (Methode)

Syntax

```
procedure DefWndProc(var Msg: TMessage); virtual;
```

Beschreibung

Überschreibt die Standard-Botschaftsbearbeitung von **TWindow**, indem es die Windows-Funktion DefFrameProc anstelle von DefWindowProc aufruft.

Siehe auch

TMDIWindow
TWindow.DefWndProc

TMDIWindow.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt 'TurboMDIWindow', den Namen der Fensterklasse von TMDIWindow, zurück.

Siehe auch

TMDIWindow

TMDIWindow.GetClient (Methode)

Syntax

```
function GetClient: PMDIClient; virtual;
```

Beschreibung

Gibt einen Zeiger auf das MDI-Client-Fenster, das in ClientWnd gespeichert ist, zurück.

Siehe auch

TMDIWindow

TMDIWindow.GetWindowClass (Methode)

Syntax

```
procedure GetWindowClass(var AWndClass: TWndClass); virtual;
```

Beschreibung

Ändert den Record der Standardfensterklasse und gibt ihn in AWndClass zurück.

GetWindowClass setzt das Stilfeld auf Null, um den Stil von **TWindow.GetWindowClass** zu löschen.

Siehe auch

TMDIWindow

TMDIWindow.InitClientWindow (Methode)

Syntax

```
procedure InitClientWindow; virtual;
```

Beschreibung

Wird manchmal überschrieben. Initialisiert das MDI-Client-Fenster als **TMDIClient**-Objekt und speichert es in ClientWnd.

Siehe auch

TMDIWindow

TMDIWindow.InitChild (Methode)

Syntax

```
function InitChild: PWindowsObject; virtual;
```

Beschreibung

Initialisiert ein MDI-Objekt eines untergeordneten Fensters (**TWindow**) mit der Überschrift 'MDI Child' und gibt einen Zeiger darauf zurück.

Wenn Sie das untergeordnete Fenster von TWindow ableiten, sollten Sie InitChild, überschreiben, um ein Fenster Ihres neuen MDI-Fenstertyps zu erstellen. Beispiel:

```
function MyMDIWindow.InitChild: PWindowsObject;
begin
   InitChild := New(PMyMDIChild, Init(@Self, 'Untitled Window'));
end;
```

Siehe auch

CreateChild
TMDIWindow

TMDIWindow.SetupWindow (Methode)

Syntax

```
procedure SetupWindow; virtual;
```

Beschreibung

Initialisiert das zum MDI-Client-Fensterobjekt (ClientWnd) gehörige Fensterelement durch Aufruf von InitClientWindow und stellt es mit MakeWindow dar.

Falls Sie **SetupWindow** in einem abgeleiteten Typ überschreiben , sollten Sie das alte SetupWindow explizit aufrufen.

Siehe auch

InitClientWindow
TApplication.MakeWindow
TMDIWindow

TMDIWindow.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert das MDI-Rahmenfenster im Stream S durch Aufruf von **TWindow.Store** und zusätzliches Schreiben der Felder ClientWnd und ChildMenuPos von TMDIWindow.

Siehe auch

TMDIWindow

TMDIWindow.TileChildren (Methode)

Syntax

```
procedure TileChildren;
```

Beschreibung

Ordnet alle nicht minimierten untergeordneten MDI-Fenster so im MDI-Client-Fenster an, daß sie ohne Überlappen den verfügbaren Raum einnehmen. Ruft ClientWnd^.TileChildren auf.

Siehe auch

TMDIClient.TileChildren
TMDIWindow

TMDIWindow.CMTileChildren (Methode)

Syntax

```
procedure CMTileChildren(var Msg: TMessage); virtual cm_First +
cm_TileChildren;
```

Beschreibung

Reagiert auf die Menüwahl mit der ID cm_TileChildren durch Aufruf von **TileChildren**.

Siehe auch

TMDIWindow

TObject (Unit WObjects)

TObject bildet den Ausgangspunkt der Objekthierarchie von Object Windows. Als Basisobjekt hat es keine Vorfahren, aber viele Nachkommen. Außer TPoint und TRect stammen alle Standardobjekte von Object Windows letztlich von TObject ab.

Methoden

Init
Free
Done

TObject.Init (Methode)

Syntax

```
constructor Init;
```

Beschreibung

Reserviert Platz für das Objekt auf dem Heap.

Wird von den Konstruktoren aller abgeleiteten Objekte aufgerufen.

Siehe auch

TObject

TObject.Free (Methode)

Syntax

```
procedure Free;
```

Beschreibung

Entfernt das Objekt und ruft den Destruktor **Done** auf.

Siehe auch

TObject

TObject.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Gibt den von dynamischen Objekten belegten Speicher wieder frei.

Siehe auch

TObject

TRadioButton WObjects unit

TRadioButton ist ein Schnittstellenobjekt, das ein entsprechendes Schaltfeld repräsentiert.

TRadioButton wird normalerweise nicht in Dialogfenstern (**TDialog** oder -**TDlgWindow**) verwendet, sondern wenn man ein selbständiges Schaltfeld im Client-Bereichs eines anderen Fensters darstellen will.

Schaltfelder können gewählt und ungewählt sein. TRadioButton erbt die Zustandsverwaltung von seinem Vorfahren **TCheckBox**. Ein Schaltfeld kann Teil einer Gruppe sein (**TGroupBox**), die Steuerelemente visuell und funktionell zusammenfaßt.

Methoden

Init

TRadioButton.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnID: Integer; ATitle: PChar;
X,Y,W,H: Integer; AGroup: PGroupBox);
```

Beschreibung

Initialisiert ein Schaltfeld-Objekt mit weitergegebenem
- übergeordneten Fenster (AParent),
- Identifikationsnummer (AnId),
- Text (ATitle),
- Position (X, Y) relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters;
- Breite (W) und
- Höhe (H), und
- Gruppenfenster (AGroup).

Init setzt das Feld Attr.Style des Schaltfelds auf ws_Child oder ws_Visible oder **bs_RadioButton**.

Siehe auch

TRadioButton
ws_Window styles

TScrollBar (Unit WObjects)

TScrollBar-Objekte repräsentieren selbständige Bildlaufleisten, aber nicht die, die zu Fenstern gehören. Sie können senkrecht oder waagrecht sein, um nach oben und unten oder bzw. rechts und links zu rollen. Die meisten Methoden von TScrollBar verwalten Position und Bereich des Bildlaufsymbols.

TScrollBar hat Methoden, die automatisch auf Windows-Bildlaufleisten-Botschaften reagieren, wie **wm_HScroll** und **wm_VScroll**. Diese Methoden, (z.B. SBLineUp und SBPageDown) dienen zur Positionierung des Bildlaufsymbols.

Felder

LineMagnitude
PageMagnitude

Methoden

Init
Load
DeltaPos
GetClassName
GetPosition
GetRange
SBBottom
SBLineDown
SBLineUp
SBPageDown
SBPageUp
SBThumbPosition
SBThumbTrack
SBTop
SetPosition
SetRange
SetupWindow
Store
Transfer

TScrollBar.LineMagnitude (Feld)

Syntax

```
LineMagnitude: Integer;
```
(Read/write)

Beschreibung

LineMagnitude ist die Anzahl Bereichseinheiten, um welche die Bildlaufleiste gerollt wird, wenn der Anwender einen Richtungspfeil anklickt. Init setzt LineMagnitude als Vorgabe auf 1. **SetupWindow** setzt den Rollbereich als Vorgabe auf Null bis 100.

Siehe auch

TScrollBar

TScrollBar.PageMagnitude (Feld)

Syntax

```
PageMagnitude: Integer;
```
(Read/write)

Beschreibung

PageMagnitude ist die Anzahl Bereichseinheiten, um welche die Bildlaufleiste verschoben wird, wenn der Anwender für eine größere Verschiebung den Bildlaufbereich anklickt. **Init** setzt PageMagnitude als Vorgabe auf 10 (der Rollbereich wird als Vorgabe auf Null bis 100 gesetzt).

Siehe auch

TScrollBar

TScrollBar.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnID: Integer; X,Y,W,H: Integer;
IsHScrollBar: Boolean);
```

Beschreibung

Erzeugt und initialisiert ein TScrollBar-Objekt mit
- übergeordnetem Fenster (AParent),
- AnID als Steuerelement-ID,
- der Position (X, Y),
- Breite W und
- Höhe H.

Die Bildlaufleiste ist waagrecht (style sbs_Horz), wenn IsHScrollBar True ist, und senkrecht (style sbs_Vert), wenn es False ist.

Wenn die übergebene Höhe bei waagrechten oder die Breite bei senkrechten Bildlaufleisten Null ist, wird ein Standardwert verwendet. LineMagnitude wird mit 1 und PageMagnitude mit 10 initialisiert.

Siehe auch

sbs_ Scroll bar styles
TScrollBar

TScrollBar.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Erzeugt und lädt ein Bildlaufleisten-Element vom Stream S, indem es erst TControl.Load aufruft und danach die weiteren Felder IsHorizontal, LineMagnitude und PageMagnitude von TScrollBar liest.

Siehe auch

TScrollBar
TWindow.Load

TScrollBar.DeltaPos (Methode)

Syntax

```
function DeltaPos(Delta: Integer): Integer; virtual;
```

Beschreibung

Ändert die Position des Bildlaufsymbols um den Wert Delta (Aufruf von SetPosition). Ein negativer Wert bewegt den Regler nach oben bzw. links. Die neue Positon wird zurückgegeben.

Siehe auch

TScrollBar

TScrollBar.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt den Namen der Fensterklasse von TScrollBar, 'Scrollbar', zurück.

Siehe auch

TScrollBar

TScrollBar.GetPosition (Methode)

Syntax

```
function GetPosition: Integer; virtual;
```

Beschreibung

Gibt die aktuelle Position des Bildlaufsymbols zurück.

Siehe auch

SetPosition
TScrollBar

TScrollBar.GetRange (Methode)

Syntax

```
procedure GetRange(var LoVal, HiVal: Integer); virtual;
```

Beschreibung

Liefert den zulässigen Bereich für die Position des Bildlaufsymbols in LoVal und HiVal zurück.

Siehe auch

SetRange
TScrollBar

TScrollBar.SBBottom (Methode)

Syntax

```
procedure SBBottom(var Msg: TMessage); virtual nf_First + sb_Bottom;
```

Beschreibung

Setzt als Reaktion auf eine Eingabe die Position des Bildlaufsymbols auf den höchsten erlaubten Wert.

Diese Methode wird auf eine Bildlaufbotschaft mit dem Code **sb_Bottom** hin aufgerufen. Sie ruft ihrerseits **SetPosition** auf.

Siehe auch

TScrollBar

TScrollBar.SBLineDown (Methode)

Syntax

```
procedure SBLineDown(var Msg: TMessage); virtual nf_First + sb_LineDown;
```

Beschreibung

Bewegt mit **SetPosition** das Bildlaufsymbol um LineMagnitude nach unten oder rechts.

SBLineDown wird auf eine Bildlaufbotschaft mit dem Code **sb_LineDown** hin aufgerufen.

Siehe auch

TScrollBar

TScrollBar.SBLineUp (Methode)

Syntax

```
procedure SBLineUp(var Msg: TMessage); virtual nf_First + sb_LineUp;
```

Beschreibung

Bewegt mit **SetPosition** das Bildlaufsymbol um LineMagnitude nach oben oder links.

SBLineUp wird auf eine Bildlaufbotschaft mit dem Code **sb_LineUp** hin aufgerufen. Diese Botschaft wird z.B. gesendet, wenn der Benutzer das Pfeilsymbold 'nach oben' anklickt.

Siehe auch

TScrollBar

TScrollBar.SBPageDown (Methode)

Syntax

```
procedure SBPageDown(var Msg: TMessage); virtual nf_First + sb_PageDown;
```

Beschreibung

Bewegt mit **SetPosition** das Bildlaufsymbol um PageMagnitude nach unten oder rechts.

SBPageDown wird auf eine Bildlaufbotschaft mit dem Code **sb_Pagedown** hin aufgerufen.

Siehe auch

TScrollBar

TScrollBar.SBPageUp (Methode)

Syntax

```
procedure SBPageUp(var Msg: TMessage); virtual nf_First + sb_PageUp;
```

Beschreibung

Bewegt mit **SetPosition** das Bildlaufsymbol um PageMagnitude nach oben oder links.

SBPageUp wird auf eine Bildlaufbotschaft mit dem Code **sb_PageUp** hin aufgerufen.

Siehe auch

TScrollBar

TScrollBar.SBThumbPosition (Methode)

Syntax

```
procedure SBThumbPosition(var Msg: TMessage); virtual nf_First +
sb_ThumbPosition;
```

Beschreibung

Bewegt mit **SetPosition** das Bildlaufsymbol auf die vom Anwender gewählte Position, die mit einer Bildlaufbotschaft mit dem Code **sb_ThumbPosition**übertragen wurde.

Siehe auch

TScrollBar

TScrollBar.SBThumbTrack (Methode)

Syntax

```
procedure SBThumbTrack(var Msg: TMessage); virtual nf_First +
sb_ThumbTrack;
```

Beschreibung

Bewegt mit **SetPosition** das Bildlaufsymbol, wenn es der Anwender mit der Maus zieht.

Diese Methode wird auf eine Bildlaufbotschaft mit dem Code **sb_ThumbTrack** hin aufgerufen.

Siehe auch

TScrollBar

TScrollBar.SBTop (Methode)

Syntax

```
procedure SBTop(var Msg: TMessage); virtual nf_First + sb_Top;
```

Beschreibung

Bewegt mit **SetPosition** das Bildlaufsymbol auf die niedrigst mögliche Position (wenn der Benutzer dies anfordert). Die Position wird als Bildlaufbotschaft mit dem Code **sb_Top** übertragen.

Siehe auch

TScrollBar

TScrollBar.SetPosition (Methode)

Syntax

```
procedure SetPosition(ThumbPos: Integer); virtual;
```

Beschreibung

Bewegt das Bildlaufsymbols auf ThumbPos. Liegt ThumbPos außerhalb des erlaubten Bereichs, wird der höchstmögliche bzw. niedrigstmögliche Wert gewählt.

Siehe auch

GetPosition
TScrollBar

TScrollBar.SetRange (Methode)

Syntax

```
procedure SetRange(LoVal, HiVal: Integer); virtual;
```

Beschreibung

Setzt den zulässigen Bereich der Positionen des Bildlaufsymbols auf LoVal bis HiVal.

Siehe auch

GetRange
TScrollBar

TScrollBar.SetupWindow (Methode)

Syntax

```
procedure SetupWindow; virtual;
```

Beschreibung

Initialisiert den Bereich der Bildlaufleiste auf Null bis 100. Mit einem Aufruf von **SetRange** können Sie diesen Bereich neu definieren.

Siehe auch

TScrollBar

TScrollBar.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert die Bildlaufleiste auf dem Stream S, indem es erst TControl.Store aufruft und danach die Felder IsHorizontal, LineMagnitude und PageMagnitude von TScrollBar in den Stream schreibt.

Siehe auch

TScrollBar
TWindow.Store

TScrollBar.Transfer (Methode)

Syntax

```
function Transfer(DataPtr: Pointer; TransferFlag: Word): Word; virtual;
```

Beschreibung

Überträgt den Bereich der Bildlaufleiste und die aktuelle Position des Bildlaufsymbols an oder von der Adresse, auf die DataPtr zeigt.

- Ist TransferFlag gleich tf_GetData, wird ein Record mit Bereich und Postion an die Speicheradresse geschrieben.
- Ist TransferFlag gleich tf_SetData, werden mit den Werten dieses Record Bereich und Position festgelegt.

Transfer gibt die Anzahl der übertragenen Bytes zurück. Der genannte Record ist folgendermaßen definiert:

```
ScrollBarTransferRec = record
   LowValue: Integer;
   HighValue: Integer;
   Position: Integer;
end;
```

Siehe auch

TScrollBar

K+TScroller (Unit WObjects)

TScroller-Objekte exisiteren im Feld Scroller von **TWindow** und in abgeleiteten Objekten.

TScroller bietet einerseits einen Bildlaufmechanismus für Fenster, der mit horizontalen und vertikalen Bildlaufleisten arbeitet. Daneben unterstützt es ein als "automatisches Rollen" bezeichnetes Verfahren, das ohne Bildlaufleisten auskommt.

Normalerweise werden TScroller-Objekte von den Methoden der Fensterobjekte initialisiert und gesteuert, zu denen sie gehören.

Felder

AutoMode
AutoOrg
HasHScrollBar
HasVScrollBar
TrackMode
Window
XLine
XPage
XPos
XRange
XUnit
YLine
YPage
YPos
YRange
YUnit

Methoden

Init
Load
AutoScroll
BeginView
EndView
HScroll
IsVisibleRect
ScrollBy
ScrollTo
SetPageSize
SetRange
SetSBarRange
SetUnits
Store
VScroll

TScroller.Window (Feld)

Syntax

```
Window: PWindow;
```
(Read only)

Beschreibung

Window zeigt auf das Fenster, zu dem TScroller gehört.

Siehe auch

TScroller

TScroller.XUnit (Feld)

Syntax

```
XUnit: Integer;
```
(Read only)

Beschreibung

XUnit ist die kleinste Zahl von Pixeln, um die das Fenster horizontal gerollt werden kann. XUnit-Werte werden im Konstruktor Init definiert, können aber geändert werden.

Siehe auch

TScroller

TScroller.YUnit (Feld)

Syntax

```
YUnit: Integer;
```
(Read only)

Beschreibung

YUnit ist die kleinste Zahl von Pixeln, um die das Fenster vertikal gerollt werden kann. YUnit-Werte werden im Konstruktor Init definiert, können aber geändert werden.

Siehe auch

TScroller

TScroller.XPos (Feld)

Syntax

```
XPos: Longint;
```
(Read only)

Beschreibung

XPos ist die aktuelle horizontale Position in **XUnit**-Einheiten.

Siehe auch

TScroller

TScroller.YPos (Feld)

Syntax

```
YPos: Longint;
```
(Read only)

Beschreibung

YPos ist die aktuelle vertikale Position von TScroller in **YUnit**-Einheiten.

Siehe auch

TScroller

TScroller.XRange (Feld)

Syntax

```
XRange: Longint;
```
(Read only)

Beschreibung

XRange ist die Gesamtzahl horizontaler **XUnit**-Einheiten, um die das Fenster verschiebbar ist.

XRange-Werte werden im Konstruktor Init definiert, können aber später geändert werden.

Siehe auch

TScroller

TScroller.YRange (Feld)

Syntax

```
YRange: Longint;
```
(Read only)

Beschreibung

YRange ist die Gesamtzahl vertikaler **YUnit**-Einheiten, um die das Fenster verschiebbar ist. YRange-Werte werden im Konstruktor Init definiert, können aber später geändert werden.

Siehe auch

TScroller

TScroller.XLine (Feld)

Syntax

```
XLine: Integer;
```
(Read/write)

Beschreibung

XLine ist die Zahl von **XUnit**-Einheiten um die das Fenster nach Anklicken des Pfeils auf der Bildlaufleiste horizontal gerollt werden soll. Vorgabe ist 1.

Siehe auch

TScroller

TScroller.YLine (Feld)

Syntax

```
YLine: Integer;
```
(Read/write)

Beschreibung

YLine ist die Zahl **YUnit**-Einheiten, um die das Fenster nach Anklicken des Pfeils auf der Bildlaufleiste vertikal gerollt werden soll. Vorgabe ist 1.

Siehe auch

TScroller

TScroller.XPage (Feld)

Syntax

```
XPage: Integer;
```
(Read/write)

Beschreibung

XPage ist die Zahl von **XUnit**-Einheiten, um die durch Anklicken Bildlaufsymbols horizontal gerollt werden soll. Als Vorgabe entspricht XPage der aktuellen Fensterbreite in XUnit-Einheiten. Jedes Ändern der Fenstergröße aktualisiert diesen Wert.

Siehe auch

TScroller

TScroller.YPage (Feld)

Syntax

```
YPage: Integer;
```
(Read/write)

Beschreibung

YPage ist die Zahl von **YUnit**-Einheiten, um die durch Anklicken Bildlaufsymbols vertikal gerollt werden soll. Als Vorgabe entspricht YPage der aktuellen Fensterhöhe in YUnit-Einheiten. Jedes Ändern der Fenstergröße aktualisiert diesen Wert.

Siehe auch

TScroller

TScroller.AutoMode (Feld)

Syntax

```
AutoMode: Boolean;
```
(Read/write)

Beschreibung

AutoMode ist True, wenn automatisches Rollen verwendet werden soll. Dies dies ist die Vorgabe.

Siehe auch

TScroller

TScroller.AutoOrg (field)

Syntax

```
AutoOrg: Boolean;
```

Beschreibung

AutoOrg dient dazu, den Ursprung des Bildschirmkontextes zu bestimmen, den Paint benutzt, um das Fenster darzustellen.

- Wenn AutoOrg True ist (das ist die Vorgabe), verwendet Paint den Ursprung der Bildlaufleiste.
- Wenn es False ist, ist der Ursprung des Bildschirmkontextes (0,0) und Paint muß selbst den Ursprung der Bildlaufleiste lesen, um zu bestimmen, wie Objekte im Bildschirmkontext darzustellen sind.

Siehe auch

TScroller

TScroller.TrackMode (Feld)

Syntax

```
TrackMode: Boolean;
```
(Read/write)

Beschreibung

TrackMode ist True, wenn TScroller automatisch Bewegung des Bildlaufsymbols und Rollen des Fensterinhalts synchronisiert. Dies ist die Vorgabe.

Siehe auch

TScroller

TScroller.HasHScrollBar (Feld)

Syntax

```
HasHScrollBar: Boolean;
```
(Read/write)

Beschreibung

Wenn das zugehörige Fenster eine horizontale Bildlaufleiste hat, ist HasHScrollBar True.

Siehe auch

TScroller

TScroller.HasVScrollBar (Feld)

Syntax

```
HasVScrollBar: Boolean;
```
(Read/write)

Beschreibung

Wenn das zugehörige Fenster eine vertikale Bildlaufleiste hat, ist HasVScrollBar True.

Siehe auch

TScroller

TScroller.Init (Methode)

Syntax

```
constructor Init(TheWindow: PWindow; TheXUnit, TheYUnit: Integer;
TheXRange, TheYRange: Longint);
```

Beschreibung

Initialisiert ein neues TScroller-Objekt mit TheWindow als zugehörigem Fenster und TheXUnit, TheYUnit, TheXRange, und TheYRange als XUnit, YUnit, XRange und YRange. Setzt AutoMode und TrackMode auf True. HasHScrollBar und HasVScrollBar werden abhängig von den Attributen der Bildlaufleisten des zugehörigen Fensters gesetzt.

Siehe auch

TScroller

TScroller.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Diese Methode initialisiert und lädt eine Bildlaufleiste vom Stream S, indem sie **TObject.Init** aufruft und anschließend die Felder von TScroller (mit Ausnahme von XPage, YPage, XPos und YPos) liest.

Siehe auch

TScroller

TScroller.AutoScroll (Methode)

Syntax

```
procedure AutoScroll; virtual;
```

Beschreibung

Rollt automatisch entsprechend der Position des Mauszeigers.

Siehe auch

TScroller

TScroller.BeginView (Methode)

Syntax

```
procedure BeginView(PaintDC: HDC; var PaintInfo: TPaintStruct); virtual;
```

Beschreibung

Setzt den Ursprung des Bildschirmkontexts (PaintDC) des zugehörigen Fensters entsprechend der aktuellen Position des Bildlaufsymbols.

Siehe auch

TScroller

TScroller.EndView (Methode)

Syntax

```
procedure EndView; virtual;
```

Beschreibung

Synchronisiert die Positionen der Bildlaufleisten des zugehörigen Fensters mit der Position von TScroller.

Siehe auch

TScroller

TScroller.HScroll (Methode)

Syntax

```
procedure HScroll(ScrollRequest: Word; ThumbPos: Integer); virtual;
```

Beschreibung

Reagiert auf Ereignisse der horizontalen Bildlaufleiste durch Ändern der Positionen von TScroller und der Bildlaufleiste.

Siehe auch

TScroller
TWindow.WMHScroll

TScroller.IsVisibleRect (Methode)

Syntax

```
function IsVisibleRect(X, Y: Longint; XExt, YExt: Integer): Boolean;
virtual;
```

Beschreibung

Gibt True zurück, wenn ein Teil des Rechtecks, das von den Argumenten definiert wird, im zugehörigen Fenster sichtbar ist.

Siehe auch

TScroller

TScroller.ScrollBy (Methode)

Syntax

```
procedure ScrollBy(Dx, Dy: Longint); virtual;
```

Beschreibung

Rollt um die die Beträge Dx und Dy. Aktualisiert außerdem die Darstellung des Fensters.

Siehe auch

TScroller

TScroller.ScrollTo (Methode)

Syntax

```
procedure ScrollTo(X, Y: Longint); virtual;
```

Beschreibung

Rollt bis zur Position X und Y. Aktualisiert außerdem die Darstellung des Fensters.

Siehe auch

TScroller

TScroller.SetPageSize (Methode)

Syntax

```
procedure SetPageSize; virtual;
```

Beschreibung

Setzt die Felder XPage und YPage auf die aktuelle Breite und Höhe des zugehörigen Fensters.

Siehe auch

TScroller
TWindow.WMSize

TScroller.SetRange (Methode)

Syntax

```
procedure SetRange(TheXRange, TheYRange: Longint); virtual;
```

Beschreibung

Überschreibt die Werte von XRange und YRange aus dem Aufruf von Init mit TheXRange und TheYRange. Ruft dann **SetSBarRange**, um den Bereich der Bildlaufleisten des zugehörigen Fensters festzulegen.

Siehe auch

TScroller

TScroller.SetSBarRange (Methode)

Syntax

```
procedure SetSBarRange; virtual;
```

Beschreibung

Synchronisiert den Bereich der Bildlaufleisten des zugehörigen Fensters mit dem Bereich von TScroller.

Siehe auch

TScroller
TWindow.SetupWindow

TScroller.SetUnits (Methode)

Syntax

```
procedure SetUnits(TheXUnit, TheYUnit: Longint); virtual;
```

Beschreibung

Setzt die Felder **XUnit** und **YUnit**auf TheXUnit und TheYUnit.

Siehe auch

TScroller

TScroller.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert den Scroller auf dem Stream S durch Schreiben der Felder von TScroller, mit Ausnahme von XPage, YPage, XPos und YPos.

Siehe auch

TScroller

TScroller.VScroll (Methode)

Syntax

```
procedure VScroll(ScrollRequest: Word; ThumbPos: Integer); virtual;
```

Beschreibung

Reagiert auf Ereignisse der vertikalen Bildlaufleiste durch Ändern der Positionen des Scrollers und der Bildlaufleiste.

Siehe auch

TScroller
TWindow.WMVScroll

TSortedCollection (Unit WObjects)

TSortedCollection ist ein Abkömmling von **TCollection**, der Kollektionen implementiert, die nach einem Schlüssel sortiert sind.

Das Sortieren besorgt die virtuelle Methode **TSortedCollection.Compare**, die Sie überschreiben müssen. Neue Elemente werden in der Reihenfolge eingefügt. die Compare vorgibt.

Mit der binären Suchmethode **TSortedCollection.Search** finden Sie Elemente.

Die virtuelle Methode KeyOf, gibt einen Zeiger zurück, den Compare benutzt. Sie kann ebenfalls überschrieben werden, wenn Compare zusätzliche Informationen benötigt.

Felder

Duplicates

Methoden

Load
Compare
IndexOf
Insert
KeyOf
Search
Store

TSortedCollection.Duplicates (Feld)

Syntax

```
Duplicates: Boolean;
```

Beschreibung

Das Feld Duplicates legt fest, ob mehrere Elemente denselben Schlüssel haben dürfen.

- Falls Duplicates gleich True ist, werden Elemente mit einem bereits vorhandenen Schlüssel in die Kollektion eingefügt.
- Falls Duplicates gleich False ist, ersetzt das neue Element das alte mit demselben Schlüssel.

Der Vorgabewert ist False.

Siehe auch

TSortedCollection

TSortedCollection.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Initialisiert und lädt eine sortierte Kollektion vom Stream S, indem es erst **TCollection.Load** aufruft und danach noch das Feld **Duplicates** liest.

Siehe auch

TSortedCollection

TSortedCollection.Compare (Methode)

Syntax

```
function Compare(Key1, Key2: Pointer): Integer; virtual;
```

Beschreibung

Compare ist eine abstrakte Methode, die in allen Abkömmlingen überschrieben werden muß. Compare sollte beide Indexwerte vergleichen und folgendes Ergebnis liefern:

-1	wenn Key1 < Key2
0	wenn Key1 = Key2
1	wenn Key1 > Key2

Key1 und Key2 sind Zeigerwerte, welche die Methode **KeyOf** den entsprechenden Kollektionselementen entnimmt. Die Methode **Search** durchsucht die Elemente binär und vergleicht sie mit Hilfe von Compare.

Siehe auch

TSortedCollection

TSortedCollection.IndexOf (Methode)

Syntax

```
function IndexOf(Item: Pointer): Integer; virtual;
```

Beschreibung

Findet mit **Search** den Index des Elements. Ist es nicht in der Kollektion enthalten, gibt IndexOf den Wert -1 zurück. **IndexOf** ist folgendermaßen implementiert:

```
if Search(KeyOf(Item), I) then IndexOf := I else IndexOf := -1;
```

Siehe auch

TSortedCollection

TSortedCollection.Insert (Methode)

Syntax

```
procedure Insert(Item: Pointer); virtual;
```

Beschreibung

Das Element Item wird an der richtigen Indexposition in die sortierte Kollektion eingefügt. Ruft **Search** auf, um festzustellen, ob das Element schon existiert, und falls nicht, wo es eingefügt werden soll. **Insert** ist folgendermaßen implementiert:

```
if not Search(KeyOf(Item), I) then AtInsert(I, Item);
```

Siehe auch

TSortedCollection

TSortedCollection.KeyOf (Methode)

Syntax

```
function KeyOf(Item: Pointer): Pointer; virtual;
```

Beschreibung

Für das Element Item der Kollektion sollte KeyOf den entsprechenden Schlüssel zurückgeben. **KeyOf** gibt als Vorgabe Item selbst zurück. KeyOf wird dann überschrieben, wenn der Schlüssel, nach dem sortiert werden soll, nicht einfach das Element selbst ist.

Siehe auch

IndexOf
TSortedCollection

TSortedCollection.Search (Methode)

Syntax

```
function Search(Key: Pointer; var Index: Integer): Boolean; virtual;
```

Beschreibung

Gibt True zurück, wenn das von Key bezeichnete Element in der sortierten Kollektion gefunden wurde. Index enthält dann den Index des gefundenen Elements. Ansonsten gibt Index die Stelle an, wo das Element beim Einfügen plaziert würde.

Siehe auch

Compare
Insert
TSortedCollection

TSortedCollection.Store (Methode)

Syntax

```
procedure Store(var S: Stream);
```

Beschreibung

Speichert die sortierte Kollektion und alle ihre Elemente in den Stream S durch Aufruf von **TCollection.Store** zum Schreiben der Kollektion. Danach wird das Feld **Duplicates** in den Stream geschrieben.

Siehe auch

TSortedCollection

TStatic (Unit WObjects)

TStatic ist ein Schnittstellenobjekt, das entsprechende statische Textfelder repräsentiert. TStatic-Objekte werden üblicherweise nicht in Dialogen (**TDialog**) oder Dialogfenstern (**TDlgWindow**) verwendet, sondern sie werden eingesetzt, wenn man ein selbständiges statisches Textfeld im Client-Bereich eines anderen Fensters darstellen will.

Felder

TextLen

Methoden

Init
InitResource
Load
Clear
GetClassName
GetText
SetText
Store
Transfer

TStatic.TextLen (Feld)

Syntax

```
TextLen: Word;
```

Beschreibung

TextLen enthält die Größe des Textpuffers statischer Felder.

Die tatsächliche Anzahl speicherbarer Zeichen ist TextLen -1, weil der Puffer die abschließende Null des nullterminierten Strings aufnehmen muß.

TextLen enthält auch die von **Transfer** übertragene Anzahl Zeichen.

Siehe auch

TStatic

TStatic.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; AnID: Integer; ATitle: PChar;
X,Y,W,H: Integer);
```

Beschreibung

Initialisiert ein statisches Textfeld mit
- dem übergeordneten Fenster (AParent)
- Element-ID (AnID)
- Text (ATitle)
- Position (X, Y) relativ zum Ursprung des Client-Bereichs des übergeordneten Fensters
- Breite (W)
- Höhe (H)
- Textlänge (ATextLen)

Das statische Textfeld ist linksbündig, weil Init den Windows-Stil **ss_Left** zum Feld Attr.Style fügt. Der Stil **ws_TabStop** wird entfernt.

Init ruft DisableTransfer auf, um als Vorgabe TStatic-Objekte vom Transfer auszuschließen.

Siehe auch

TStatic

TStatic.InitResource (Methode)

Syntax

```
constructor InitResource(AParent: PWindowsObject; ResourceID, ATextLen:
Word);
```

Beschreibung

Ordnet einem TStatic-Objekt die Ressource in ResourceID zu und setzt das Feld **TextLen** auf ATextLen.

Ruft **TControl.InitResource** auf, um das Objekt zu initialisieren und zuzuordnen.

Siehe auch

TStatic

TStatic.Clear (Methode)

Syntax

```
procedure Clear; virtual;
```

Beschreibung

Löscht den Text des statischen Textfeldes.

Siehe auch

TStatic

TStatic.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Diese Methode initialisiert und lädt das statische Textfeld aus dem Stream S, indem sie TControl.Load aufruft und danach das Feld **TextLen** liest.

Siehe auch

TStatic
TWindow.Load

TStatic.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt den Namen der Fensterklasse von TStatic, 'Static', zurück.

Siehe auch

TStatic

TStatic.GetText (Methode)

Syntax

```
function GetText(ATextString: PChar; MaxChars: Integer): Integer; virtual;
```

Beschreibung

Ruft den Text des statischen Felds ab und speichert ihn im Argument ATextString.

MaxChars gibt die Maximalgröße von ATextString an. GetText gibt die Größe des abgerufenen String zurück.

Siehe auch

TStatic

TStatic.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Diese Methode speichert das statische Textfeld im Stream S, indem sie TControl.Store aufruft und anschließend das Feld **TextLen** schreibt.

Siehe auch

TStatic
TWindow.Store

TStatic.SetText (Methode)

Syntax

```
procedure SetText(ATextString: PChar); virtual;
```

Beschreibung

Setzt den Text des statischen Felds auf den String aus ATextString.

Siehe auch

TStatic

TStatic.Transfer (Methode)

Syntax

```
function Transfer(DataPtr: Pointer; TransferFlag: Word): Word; virtual;
```

Beschreibung

Überträgt TextLen Zeichen des aktuellen Texts zu oder von der Adresse in DataPtr.

- Ist TransferFlag gleioh tf_GetData, wird der Text aus dem statischen Textfeld an die Adresse geschrieben
- Ist TransferFlag gleioh tf_SetData, wird der Text von der Adresse ins Textfeld geschrieben.

In beiden Fällen gibt Transfer die Anzahl der übertragenen Bytes (TextLen)zurück.

- Ist TransferFlag gleich tf_SizeData,, gibt Transfer die Anzahl der zu übertragenden Bytes zurück.

Siehe auch

TStatic

TStream (Unit WObjects)

TStream ist ein allgemeines abstraktes Objekt, das polymorphe Ein/Ausgabe von einem Speichermedium ermöglicht.

Sie können eigene Stream-Objekte ableiten, indem Sie die virtuellen Methoden GetPos, GetSize, Read, Seek, Truncate, und Write überschreiben. Object Windows selbst leitet **TDosStream** und **TEmsStream** ab.

Für gepufferte abgeleitete Streams müssen Sie außerdem **Flush** überschreiben.

Felder

ErrorInfo
Status

Methoden

CopyFrom
Error
Flush
Get
GetPos
GetSize
Put
Read
ReadStr
Reset
Seek
StrRead
StrWrite
Truncate
Write
WriteStr

TStream.Status (Feld)

Syntax

```
Status: Integer
```
(Read/write)

Beschreibung

Beschreibt den aktuellen Status des Stream.

TStream Fehlercodes

stOk	Kein Fehler
stError	Zugriffsfehler
stInitError	Stream kann nicht initialisiert werden
stReadError	Leseversuch über das Ende des Stream hinaus
stWriteError	Stream kann nicht erweitert werden
stGetError	Get für einen unregistrierten Objekttyp
stPutError	Put für einen unregistrierten Objekttyp

Wenn Status nicht stOk ist, werden keine Stream-Operationen durchgeführt, bis Reset aufgerufen wird.

Siehe auch

TStream

TStream.ErrorInfo (Feld)

Syntax

```
ErrorInfo: Integer
```
(Read/write)

Beschreibung

ErrorInfo enthält weitere Informationen, wenn Status nicht stOk ist.

Für Status-Werte von stError, stInitError, stReadError, und stWriteError, enthält ErrorInfo den DOS- oder EMS-Fehlercode, wenn einer verfügbar ist.

Wenn Status stGetError ist, enthält ErrorInfo die Objekttypen-ID (das Feld ObjType eines TStreamRec) des unregistrierten Objekttyps.

Wenn Status stPutError ist, enthält ErrorInfo den Offset des VMT-Datensegments (das Feld VmtLink eines TStreamRec) des unregistrierten Objekttyps.

Siehe auch

TStream

TStream.CopyFrom (Methode)

Syntax

```
procedure CopyFrom(var S: TStream; Count: Longint);
```

Beschreibung

Kopiert Count Bytes vom Stream S an das aufrufende Stream-Objekt.

```
{Kopie des gesamten Stream}
NewStream := New(TEmsStream, Init(OldStream^.GetSize));
OldStream^.Seek(0);
NewStream^.CopyFrom(OldStream, OldStream^.GetSize);
```

Siehe auch

GetSize
TObject.Init
TStream

TStream.Error (Methode)

Syntax

```
procedure Error(Code, Info: Integer); virtual;
```

Beschreibung

Wird bei jedem Stream-Fehler aufgerufen. Die Vorgabeprozedur Error speichert Code und Info in den Feldern Status und ErrorInfo und ruft dann, wenn die globale Variable StreamError nicht **nil** ist, die Prozedur auf, auf welche StreamError zeigt. Nach einem Fehler werden keine Operationen auf dem Stream ausgeführt, bis **Reset** aufgerufen wird.

Siehe auch

TStream

TStream.Flush (Methode)

Syntax

```
procedure Flush; virtual;
```

Beschreibung

Eine abstrakte Methode, die überschrieben werden muß, wenn der Nachkomme einen Puffer implementiert. Diese Methode kann jeden Puffer leeren, indem der Lesepuffer gelöscht und/oder der Schreibepuffer in den Speicher geschrieben wird. Als Vorgabe tut Flush nichts.

Siehe auch

TBufStream.Flush
TStream

TStream.Get (Methode)

Syntax

```
function Get: PObject;
```

Beschreibung

Liest ein Objekt vom Stream. Dieses muß vorher von **Put** auf den Stream geschrieben worden sein.

Get liest zuerst die Identifikation des Objekttyps (ein Word) vom Stream. Es findet den entsprechenden Objekttyp durch Vergleich mit dem Feld ObjType aller registrierten Objekttypen (siehe TStreamRec), und ruft schließlich den Konstruktor Load des Objekts, um es zu initialisieren und zu laden.

Ist die Objekttyp-ID Null, gibt Get einen **nil**-Zeiger zurück; war das Objekt nicht registriert (mit RegisterType), ruft Get **Error** auf und gibt ebenfalls einen **nil**-Zeiger zurück. Andernfalls gibt Get einen Zeiger auf das neu erzeugte Objekt zurück.

Siehe auch

TStream

TStream.GetPos (Methode)

Syntax

```
function GetPos: Longint; virtual;
```

Beschreibung

Gibt die aktuelle Position des aufrufenden Stream zurück. Diese abstrakte Methode muß immer überschrieben werden.

Siehe auch

Seek
TStream

TStream.GetSize (Methode)

Syntax

```
function GetSize: Longint; virtual;
```

Beschreibung

Gibt die Gesamtgröße des aufrufenden Stream zurück. Diese abstrakte Methode muß immer überschrieben werden.

Siehe auch

TStream

TStream.Put (Methode)

Syntax

```
procedure Put(P: PObject);
```

Beschreibung

Schreibt ein Objekt auf den Stream. Dieses kann später von **Get** wieder gelesen werden.

Die Methode Put ermittelt zunächst den Typregistrierungs-Record, indem sie das VMT-Offset des Objekts mit dem Feld VmtLink aller registrierten Objekttypen vergleicht (siehe TStreamRec). Dann schreibt sie die Objekttypen-ID in den Stream und ruft schhließlich die Methode Store des Objekts auf, um es zu speichern.

Ist das Argument P an Put **nil**, wird ein Word mit dem Wert Null in den Stream geschrieben. War der Objekttyp P nicht registriert (mit RegisterType), ruft Put **Error** auf und schreibt nichts in den Stream.

Siehe auch

TStream

TStream.Read (Methode)

Syntax

```
procedure Read(var Buf; Count: Word); virtual;
```

Beschreibung

Diese abstrakte Methode muß von allen Nachkommen überschrieben werden. Read sollte Count Bytes vom Stream in Buf lesen und die aktuelle Position auf dem Stream um Count Bytes vorwärts bewegen. Falls ein Fehler auftritt, sollte Read **Error** aufrufen und Buf mit Count Null-Bytes füllen.

Siehe auch

TStream
Write

TStream.ReadStr (Methode)

Syntax

```
function ReadStr: PString;
```

Beschreibung

Liest einen String von der aktuellen Position des aufrufenden Stream und gibt einen PString-Zeiger zurück. ReadStr ruft GetMem auf, um (Stringlänge+1) Bytes für den String zu reservieren.

Siehe auch

TStream
WriteStr

TStream.Reset (Methode)

Syntax

```
procedure Reset;
```

Beschreibung

Setzt Fehlerbedingungen auf dem Stream zurück, indem **Status** und **ErrorInfo** auf Null gesetzt werden. Mit dieser Methode kann nach einem behobenen Fehler auf dem Stream weitergearbeitet werden.

Siehe auch

stXXXX-Konstanten
TStream

TStream.Seek (Methode)

Syntax

```
procedure Seek(Pos: Longint); virtual;
```

Beschreibung

Diese abstrakte Methode muß von allen Nachkommen überschrieben werden.Seek setzt die aktuelle Position auf Pos Bytes vom Anfang des aufrufenden Stream. Ein Stream beginnt bei Position Null.

Siehe auch

GetPos
TStream

TStream.StrRead (Methode)

Syntax

```
function StrRead: PChar;
```

Beschreibung

Diese Methode liest einen nullterminierten String vom Stream. Sie liest erst die Stringlänge und dann ensprechend viele weitere Zeichen. Sie gibt einen Zeiger auf den nullterminierten String zurück.

Siehe auch

TStream
StrWrite

TStream.StrWrite (Methode)

Syntax

```
procedure StrWrite(P: PChar);
```

Beschreibung

Diese Methode schreibt den nullterminierten String P in den Stream, indem sie erst die Stringlänge und dann die entsprechende Anzahl von Zeichen schreibt.

Siehe auch

StrRead
TStream

TStream.Truncate (Methode)

Syntax

```
procedure Truncate; virtual;
```

Beschreibung

Diese abstrakte Methode muß von allen Nachkommen überschrieben werden. Truncate löscht alle Daten ab der aktuellen Position bis zum Ende des aufrufenden Stream.

Siehe auch

GetPos
Seek
TStream

TStream.Write (Methode)

Syntax

```
procedure Write(var Buf; Count: Word); virtual;
```

Beschreibung

Diese abstrakte Methode muß von allen Nachkommen überschrieben werden. Write sollte Count Bytes vom Puffer Buf auf den Stream Schreiben und die aktuelle Position um Count Bytes vorwärts bewegen. Im Fehlerfall sollte Write **Error** aufrufen.

Siehe auch

Read
TStream

TStream.WriteStr (Methode)

Syntax

```
procedure WriteStr(P: PString);
```

Beschreibung

Schreibt den String P^ ab der aktuellen Position auf den aufrufenden Stream.

Siehe auch

ReadStr
TStream

TStrCollection (Unit WObjects)

TStrCollection ist ein einfacher Abkömmling von TSortedCollection mit einer sortierten Liste von ASCII-Strings.

Die Methode TSortedCollection.Compare wird überschrieben, um die Sortierung nach ASCII-Konvention zu erreichen. Sie können Compare überschreiben, um andere Sortierordnungen zu verwenden, wie z.B. für nicht-englische .Zeichensätze.

Methoden

Compare
FreeItem
GetItem
PutItem

TStrCollection.Compare (Methode)

Syntax

```
function Compare(Key1, Key2: Pointer): Integer; virtual;
```

Beschreibung

Vergleicht die Strings Key1^ und Key2^. Zurückgegeben wird

- -1, wenn Key1 < Key2,
- 0, wenn Key1 = Key2
- +1, wenn Key1 > Key,2 ist.

Siehe auch

TSortedCollection.Search
TStrCollection

TStrCollection.FreeItem (Methode)

Syntax

```
procedure FreeItem(Item: Pointer); virtual;
```

Beschreibung

Entfernt den String Item^ aus der sortierten Kollektion und gibt ihn frei.

Siehe auch

TStrCollection

TStrCollection.GetItem (Methode)

Syntax

```
function GetItem(var S: TStream): Pointer; virtual;
```

Beschreibung

Liest einen String vom Stream **TStream** durch Aufruf von S.ReadStr.

Siehe auch

TStrCollection
TStream.ReadStr

TStrCollection.PutItem (Methode)

Syntax

```
procedure PutItem(var S: TStream; Item: Pointer); virtual;
```

Beschreibung

Schreibt den String Item^ in den Stream **TStream** durch Aufruf von S.WriteStr.

Siehe auch

TStream.WriteStr
TStrCollection

TWindow (Unit WObjects)

TWindow definiert das grundlegende Verhalten aller Fenster und Steuerelemente. Objektinstanzen von TWindow sind leere generische Fenster, aber sie können Menüs, Mauszeigerformen und Symbole definieren.

Felder

Attr
DefaultProc
FocusChildHandle
Scroller

Methoden

Init
InitResource
Load
Done
Create
DefWndProc
GetID
GetWindowClass
Paint
SetupWindow
Store
WMActivate
WMCreate
WMHScroll
WMLButtonDown
WMMove
WMPaint
WMSize
WMVScroll

TWindow.Attr (Feld)

Syntax

```
Attr: TWindowAttr;
```

Beschreibung

Attr enthält den Record TWindowAttr, der die Fenstererzeugungsattribute definiert. Das sind Text, Stil, erweiterter Stil, Position und Größe, Fenster-Handle und Steuerelement-ID. Diese Attribute werden vom Konstruktor Init mit Standardvorgaben belegt, können aber vom Init-Konstruktor eines Nachkommens überschrieben werden. Der Record TWindowAttr ist wie folgt definiert:

```
TWindowAttr = record
   Title: PChar;
   Style: Longint;
   ExStyle: Longint;
   X, Y, W, H: Integer;
   Param: Pointer;
   case Integer of
    0: (Menu: HMenu);  {Menühandle oder... }
    1: (Id: Integer);  {ID des untergeordneten Fensters}
end;
```

Siehe auch

TWindow

TWindow.DefaultProc (Feld)

Syntax

```
DefaultProc: TFarProc;
```
(Read only)

Beschreibung

Adresse der Standard-Fensterprozedur, die die Standard-Verarbeitung von Windows-Botschaften definiert.

Siehe auch

TWindow

TWindow.FocusChildHandle (Feld)

Syntax

```
FocusChildHandle: THandle;
```
(Read only)

Beschreibung

FocusChildHandle speichert das Windows-Handle zu demjenigen untergeordneten Fenster, welches bei der letzten Aktivieruung des Fensters den Fokus hatte.

Siehe auch

TWindow

TWindow.Scroller (Feld)

Syntax

```
Scroller: PScroller;
```
(Read/write)

Beschreibung

Scroller zeigt auf ein **TScroller-**Objekt, sofern das Fenster eine Bildlaufleiste hat. Das TScroller-Objekt sollte vom Konstruktor Init initialisiert werden.

Siehe auch

TWindow

TWindow.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject; ATitle: PChar);
```

Beschreibung

Erzeugt ein Fensterobjekt mit dem in AParent übergebenen übergeordneten Fenster und dem Titel ATitle. AParent sollte bei Hauptfenstern **nil** sein, weil diese kein übergeordnetes Fenster haben.

Das Feld Attr.Style des Objekts wird auf ws_OverlappedWindow gesetzt, es sei denn, das Fenster ist ein untergeordnetes MDI-Fenster. In diesem Fall wird das Feld Attr.Style auf ws_ClipSiblings gesetzt. (Siehe **ws_ Window styles**.)

Die Positions- und Abmessungsfelder im Record Attr werden auf geeignete Vorgaben für überlappende und Pop-Up-Fenster gesetzt. Sie können Init in Nachkommen überschreiben, wenn Sie zuerst TWindow.Init explizit aufrufen. Danach können Sie den Attr-Feldern neue Werte zuweisen. Scroller ist als Vorgabe auf **nil**.

Siehe auch

TWindow

TWindow.InitResource (Methode)

Syntax

```
constructor InitResource(AParent: PWindowsObject; ResourceID: Word);
```

Beschreibung

Initialisiert ein ObjectWindows-Objekt, das einem mit einer Ressourcendefinition erzeugten Schnittstellenelement (normalerweise einem Steuerelement) zugeordnet werden soll. Ruft **TWindowsObject.Init** auf.

Siehe auch

TWindow

TWindow.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Diese Methode initialisiert und lädt ein Fenster vom Stream S, indem sie erst **TWindowsObject.Load** aufruft und danach die von TWindow hinzugefügten Felder (Attr und Scroller) liest.

Siehe auch

TWindow

TWindow.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Diese Methode gibt zuerst das Objekt **TScroller** in Scroller frei (falls eines vorhanden ist), um dann mit einem Aufruf von **TWindowsObject.Done** das ganze Objekt zu entfernen.

Siehe auch

TWindow

TWindow.Create (Methode)

Syntax

```
function Create: Boolean; virtual;
```

Beschreibung

Erzeugt das Schnittstellenelement eines Fensterobjekts, wenn das Objekt nicht mit InitResource erzeugt wurde (in diesem Fall ist es bereits vorhanden).

Create ruft zuerst Register auf, um die Fensterklasse zu registrieren, wenn diese noch nicht registriert war. Dann erzeugt Create das Fenster und ruft SetupWindow auf. Sie können SetupWindow zum Initialisieren neuer Fenster selbst definieren. Normalerweise geschieht das durch Erzeugen untergeordneter Fenster und Zeichnen von Grafik oder Text.

Create gibt bei erfolgreicher Ausführung True zurück; ansonsten ist der Rückgabewert False und Create ruft Error auf.

Siehe auch

InitResource
TApplication.MakeWindow
TWindow
TWindowsObject.Register
TWindowsObject.SetupWindow

TWindow.DefWndProc (Methode)

Syntax

```
procedure DefWndProc(var Msg: TMessage); virtual;
```

Beschreibung

Ruft die Standardprozedur des Fensters auf, die Windows-Botschaften verwaltet. Das Ergebnis dieses Aufrufs wird im Feld Result des Botschaft-Records Msg gespeichert.

Siehe auch

TWindow

TWindow.GetID (Methode)

Syntax

```
function GetId: Integer; virtual;
```

Beschreibung

Gibt die Fensterbezeichner (z.B. eine Steuerelement-ID) zurück.

Siehe auch

TWindow

TWindow.GetWindowClass (Methode)

Syntax

```
procedure GetWindowClass(var AWndClass: TWndClass); virtual;
```

Beschreibung

Füllt einen mit AWndClass übergebenen Fensterklassen-Record mit Standardwerten für seine Registrierungsattribute. Das Stil-Feld wird mit cs_HRedraw oder cs_VRedraw belegt (siehe **cs_ Class styles**), das Symbol wird als generisches Symbol und der Mauszeiger als normaler Pfeilzeiger definiert. Die Hintergrundfarbe wird auf die des Systems gesetzt und die zu registrierende Klassenbezeichnung wird über einen Aufruf von GetClassName ermittelt.

Siehe auch

Create
TWindow
TWindowsObject.GetClassName
TWindowsObject.Register

TWindow.Paint (Methode)

Syntax

```
procedure Paint(PaintDC: HDC; var PaintInfo: TPaintStruct); virtual;
```

Beschreibung

Dient als Platzhalter für abgeleitete Typen, die Paint-Methoden definieren. Paint wird automatisch auf eine Windows-Anforderung hin (**wm_Paint**) aufgerufen, um den Fensterinhalt neu darzustellen. PaintDC muß als Bildschirm-Kontext benutzt werden; er wird schon vor dem Aufruf von Paint eingeholt und nach Paint wieder freigegeben. Die Struktur PaintInfo enthält genauere Informationen (hauptsächlich Windows-interne) über die Zeichenanforderung.

Siehe auch

TWindow
TPaintStruct

TWindow.SetupWindow (Methode)

Syntax

```
procedure SetupWindow; virtual;
```

Beschreibung

Stellt ein neu erzeugtes Fenster dar. Ist das Fenster ein untergeordnetes MDI-Fenster, ruft SetupWindow SetFocus auf, um ihm den Fokus zuzuteilen. Wenn das Fenster eine Bildlaufleiste hat, ruft SetupWindow SetSBarRange auf, um deren Bereich festzusetzen.

Siehe auch

TScroller.SetSBarRange
TWindow

TWindow.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Die Methode speichert das Fenster in den Stream S, indem sie erst **TWindowsObject.Store** aufruft und dann die von TWindow hinzugefügten Felder (Attr und Scroller) schreibt.

Siehe auch

TWindow

TWindow.WMCreate (Methode)

Syntax

```
procedure WMCreate(var Msg: TMessage); virtual wm_First + wm_Create;
```

Beschreibung

Reagiert auf die Botschaft **wm_Create**durch Aufruf von SetupWindow und DefWndProc. Da das Erstellen von Fenstern unter ObjectWindows anders als unter Windows behandelt wird, muß die Meldung wm_Create abgefangen und zum Einrichten von Fensterattributen verwendet werden.

Siehe auch

TWindow

TWindow.WMActivate (Methode)

Syntax

```
procedure WMActivate(var Msg: TMessage); virtual wm_First + wm_Activate;
```

Beschreibung

Für Fenster, die Tastaturbotschaften für ihre Steuerelemente entgegennehmen. Die Methode reagiert auf Änderungen des Fokus, indem sie das Handle des untergeordneten Fensters, das gerade den Fokus hat, in FocusChildHandle speichert und den Fokus wiederherstellt.

Siehe auch

TWindow
TWindowsObject.EnableKBHandler

TWindow.WMHScroll (Methode)

Syntax

```
procedure WMHScroll(var Msg: TMessage); virtual wm_First + wm_HScroll;
```

Beschreibung

Für Fenster mit Bildlaufleisten. Die Methode reagiert auf Ereignisse der horizontalen Bildlaufleiste, indem sie deren Methode HScroll und die Methode DefWndProc aufruft.

Siehe auch

TScroller.HScroll
TWindow

TWindow.WMLButtonDown (Methode)

Syntax

```
procedure WMLButtonDown(var Msg: TMessage); virtual wm_First +
wm_LButtonDown;
```

Beschreibung

Für Fenster mit automatischem Rollmechanismus. Die Methode reagiert auf das Klicken der linken Maustaste, indem sie alle weiteren Mauseingaben abfängt, bis die Taste losgelassen wird. Sie weist Windows an, **wm_Timer**-Botschaften zu versenden, solange die Taste gedrückt bleibt. Wenn Sie diese Methode überschreiben und dennoch automatisches Rollen einsetzen wollen, sollten Sie sie in Ihrer neuen WMLButtonDown-Methode aufrufen.

Siehe auch

TWindow

TWindow.WMMove (Methode)

Syntax

```
procedure WMMove(var Msg: TMsg); virtual wm_First + wm_Move;
```

Beschreibung

Bringt die Felder Attr.X und Attr.Y auf den neuen Stand, wenn eine **wm_Move**-Botschaft empfangen wird.

Wenn das Fenster zum Symbol verkleinert oder zum Vollbild vergrößert worden ist, wird die Botschaft ignoriert.

TWindow.WMPaint (Methode)

Syntax

```
procedure WMPaint(var Msg: TMessage); virtual wm_First + wm_Paint;
```

Beschreibung

Reagiert auf die Windows-Meldung **wm_Paint** durch Aufruf der Methode Paint des Fensterobjekts. Wenn das Fenster eine Bildlaufleiste hat, ruft WMPaint BeginView vor dem Aufruf von Paint und EndView nach dem Aufruf von Paint auf.

Siehe auch

Paint
TWindow
TScroller.BeginView
TScroller.EndView

TWindow.WMSize (Methode)

Syntax

```
procedure WMSize(var Msg: TMessage); virtual wm_First + wm_Size;
```

Beschreibung

Für Fenster mit Bildlaufleisten. Reagiert auf eine Änderung der Fenstergröße mit einem Aufruf von SetPageSize.

Siehe auch

TScroller.SetPageSize
TWindow

TWindow.WMVScroll (Methode)

Syntax

```
procedure WMVScroll(var Msg: TMessage); virtual wm_First + wm_VScroll;
```

Beschreibung

Für Fenster mit Bildlaufleisten. Die Methode reagiert auf Ereignisse der senkrechten Bildlaufleiste, indem sie deren Methode VScroll und die Methode DefWndProc aufruft.

Siehe auch

TScroller.VScroll
TWindow

TWindowsObject (Unit WObjects)

TWindowsObject definiert das grundlegende Verhalten aller Schnittstellenobjekte, wie Fenster, Dialogfenster und Steuerelemente

TWindowsObject ist ein abstraktes Objekt, dessen Methoden nur für abgeleitete Typen sinnvoll sind.

Die Methoden implementieren das grundlegende Konzept der Erzeugung und Entfernung von Schnittstellenelementen, der Fensterklassenregistrierung und den Mechanismus zur Beantwortung von Botschaften.

Felder

ChildList
Flags
HWindow
Instance
Parent
Status
TransferBuffer

Methoden

Init
Load
Done
CanClose
ChildWithID
CloseWindow
CMExit
Create
CreateChildren
DefChildProc
DefCommandProc
DefNotificationProc
DefWndProc
Destroy
DisableAutoCreate
DisableTransfer
DispatchScroll
EnableAutoCreate
EnableKBHandler
EnableTransfer
FirstThat
ForEach
GetChildPtr
GetChildren
GetClassName
GetClient
GetID
GetSiblingPtr
GetWindowClass
IsFlagSet
Next
Previous
PutChildPtr

PutChildren
PutSiblingPtr
Register
SetFlags
SetupWindow
Show
Store
Transfer
TransferData
WMActivate
WMClose
WMCommand
WMDestroy
WMHScroll
WMNCDestroy
WMQueryEndSession
WMVScroll

TWindowsObject.ChildList (Feld)

Syntax

```
ChildList: PWindowsObject;
```
(Read only)

Beschreibung

ChildList ist eine verkettete Liste aller untergeordneten Fensterobjekte des Schnittstellenobjekts, einschließlich Fenstern, Dialogfenstern und Steuerelementen.

ChildList zeigt auf das zuletzt hinzugefügte Objekt.

Siehe auch

TWindowsObject

TWindowsObject.Flags (Feld)

Syntax

```
Flags: Byte;
```
(Read/write)

Beschreibung

Flags ist ein Datenbyte, in dessen Bits folgende Fensterattribute gespeichert sind: Tastaturverwaltung, automatische Erzeugung, Transfer, MDI-Status, Ressourcenerzeugung.

Siehe auch

IsFlagSet
SetFlags
TWindowsObject

TWindowsObject.HWindow (Feld)

Syntax

```
HWindow: HWnd;
```
(Read only)

Beschreibung

HWindow enthält ein Handle zum zugehörigen Schnittstellenelement. Wenn es kein solches Element gibt, ist HWindow gleich Null, dem Wert eines ungültigen Handle. Wird ein Element erzeugt (Create), dann wird HWindow auf das neue Handle gesetzt; wird es entfernt, (WMNCDestroy), dann wird HWindow auf Null gesetzt.

Siehe auch

TWindowsObject

TWindowsObject.Instance (Feld)

Syntax

```
Instance: TFarProc;
```
(Read only)

Beschreibung

Instance zeigt auf den Programmcode, der vor Eintritt in die Fensterprozedur von Object Windows ausgeführt wird.

Siehe auch

TWindowsObject

TWindowsObject.Parent (Feld)

Syntax

```
Parent: PWindowsObject;
```
(Read only)

Beschreibung

PWindowsObject zeigt auf das Schnittstellenobjekt, das für dieses Schnittstellenobjekt das übergeordnete Fenster darstellt.

Siehe auch

TWindowsObject

TWindowsObject.Status (Feld)

Syntax

```
Status: Integer;
```
(Read/write)

Beschreibung

Status zeigt an, ob der Versuch, ein Schnittstellenobjekt und das zugehörige Schnittstellenelement zu initialisieren, erfolgreich war.

Der Prozess ist erfolgreich, solange Status größer oder gleich Null, erfolglos, wenn er kleiner Null ist.

Nachkommen von TWindowsObject, wie **TWindow** und **TDialog**, prüfen Status, bevor sie ihre zugeordneten Elemente erzeugen. Verwenden Sie Status im Code abgeleiteter Typen, um einen Initialisierungsfehler anzuzeigen.

Mögliche Fehlerwerte sind em_InvalidWindow, em_InvalidClient, em_InvalidChild und em_InvalidMainWindow.

Siehe auch

em_XXXX-Konstanten
TWindowsObject

TWindowsObject.TransferBuffer (Feld)

Syntax

```
TransferBuffer: Pointer;
```
(Read/write)

Beschreibung

TransferBuffer zeigt auf einen Transfer-Record, der von einer Anwendung definiert wurde, die den Transfermechanismus benutzt. Ansonsten ist er **nil**.

Siehe auch

TWindowsObject

TWindowsObject.Init (Methode)

Syntax

```
constructor Init(AParent: PWindowsObject);
```

Beschreibung

Erzeugt und initialisiert ein Schnittstellenobjekt.

Konstruktoren von abgeleiteten Typen müssen TWindowsObject.Init aufrufen.

Ruft EnableAutoCreate auf, damit ein untergeordnetes Fenster standardmäßig zusammen mit seinem übergeordneten Fenster erzeugt und dargestellt wird. Fügt das Schnittstellenobjekt in die Liste der untergeordneten Fenster des übergeordneten Fensterobjekts ein.

Siehe auch

EnableAutoCreate
TWindowsObject

TWindowsObject.Load (Methode)

Syntax

```
constructor Load(var S: TStream);
```

Beschreibung

Diese Methode initialisiert und lädt ein Schnittstellenobjekt vom Stream S, indem sie den Status, weitere Attribute und die Größe von ChildList liest und dann alle untergeordneten Fenster lädt.

Siehe auch

TWindowsObject

TWindowsObject.Done (Methode)

Syntax

```
destructor Done; virtual;
```

Beschreibung

Diese Methode entfernt das Schnittstellenobjekt, indem sie zuerst das Schnittstellenelements löscht (falls eines existiert) und dann TObject.Done aufruft.

Sie gibt alle untergeordneten Fenster frei und löscht ihre eigene Instanz aus der Liste der untergeordneten Fenster ihres übergeordneten Fensters. Destruktoren abgeleiteter Typen müssen stets mit einem Aufruf von Done enden.

Siehe auch

TWindowsObject

TWindowsObject.CanClose (Methode)

Syntax

```
function CanClose: Boolean; virtual
```

Beschreibung

Ruft die Methode CanClose für jedes untergeordneten Fenster auf, und gibt False zurück, wenn ein untergeordnetes Fenster False zurückgibt, weil das Schnittstellenelement dann nicht geschlossen werden sollte. Geben die CanClose-Methoden aller untergeordneten Fenster True zurück, gibt CanClose auch True zurück.

Siehe auch

TWindowsObject

TWindowsObject.ChildWithID (Methode)

Syntax

```
function ChildWithId(Id: Integer): PWindowsObject; virtual;
```

Beschreibung

Gibt einen Zeiger auf das Fenster aus der Liste der untergeordneten Fenster zurück, das die weitergegebene ID hat.

Wenn kein solches Fenster gefunden wurde, gibt ChildWithID **nil** zurück.

Ist Id = -1, gibt ChildWithID das erste Nicht-Steuerelementobjekt aus der Liste der untergeordneten Fenster zurück.

Siehe auch

TWindowsObject

TWindowsObject.CloseWindow (Methode)

Syntax

```
procedure CloseWindow;
```

Beschreibung

Ruft CanClose auf, um zu prüfen, ob das Fenster geschlossen werden kann. Falls CanClose True zurückgibt, wird das Fensterelement entfernt und das dazugehörige Fensterobjekt freigegeben.

Siehe auch

TWindowsObject

TWindowsObject.CMExit (Methode)

Syntax

```
procedure CMExit(var Msg:TMessage); virtual cm_First + cm_Exit;
```

Beschreibung

Wenn ein Aufruf von CanClose True zurückgibt, bewirkt diese Methode als Reaktion auf eine cm_Exit-Botschaft, daß die Anwendung beendet wird.

Eine cm_Exit-Botschaft wird normalerweise nur an das Hauptfenster der Anwendung gesendet.

Siehe auch

TWindowsObject

TWindowsObject.Create (Methode)

Syntax

```
function Create: Boolean; virtual;
```

Beschreibung

Dies ist eine abstrakte Methode, die in abgeleiteten Typen überschrieben wird, um das zugehörige Schnittstellenelement des Schnittstellenobjekts zu erzeugen.

Siehe auch

TWindowsObject

TWindowsObject.CreateChildren (Methode)

Syntax

```
function CreateChildren: Boolean;
```

Beschreibung

Ruft für alle untergeordneten Fenster **Create** auf.

SetupWindow ruft diese Methode automatisch auf, so daß man das normalerweise nicht selbst machen muß.

Rufen Sie CreateChildren nach einem Aufruf von **GetChildren** auf, wenn Sie visuelle Elemente für untergeordnete Fenster von einem Stream laden wollen.

TWindowsObject.DefChildProc (Methode)

Syntax

```
procedure DefChildProc(var Msg: TMessage); virtual;
```

Beschreibung

Wenn eine Botschaft ankommt, die auf der ID eines untergeordneten Fensters basiert, so aktiviert diese Methode die Standard-Botschaftsbearbeitung, indem sie das Feld Result von Msg auf Null setzt, was heißt, daß die Botschaft nicht bearbeitet wurde.

Siehe auch

TWindowsObject

TWindowsObject.DefCommandProc (Methode)

Syntax

```
procedure DefCommandProc(var Msg: TMessage); virtual;
```

Beschreibung

Wenn eine Befehlsbotschaft ankommt, so aktiviert diese Methode die Standard-Botschaftsbearbeitung, indem sie das Feld Result von Msg auf Null setzt, was heißt, daß die Botschaft nicht bearbeitet wurde.

Siehe auch

TWindowsObject

TWindowsObject.DefNotificationProc (Methode)

Syntax

```
procedure DefNotificationProc(var Msg: TMessage); virtual;
```

Beschreibung

Wenn eine Informationsbotschaft ankommt, so aktiviert diese Methode die Standard-Botschaftsbearbeitung, indem sie das Feld Result von Msg auf Null setzt, was heißt, daß die Botschaft nicht bearbeitet wurde.

Siehe auch

TWindowsObject

TWindowsObject.DefScrollProc (Methode)

Syntax

```
procedure DefScrollProc(var Msg: TMessage); virtual;
```

Beschreibung

Wenn eine Bildlaufbotschaft ankommt, so aktiviert diese Methode die Standard-Botschaftsbearbeitung, indem sie das Feld Result von Msg auf Null setzt, was heißt, daß die Botschaft nicht bearbeitet wurde.

Siehe auch

TWindowsObject

TWindowsObject.DefWndProc (Methode)

Syntax

```
procedure DefWndProc(var Msg: TMessage); virtual;
```

Beschreibung

Diese Standardfensterprozedur tut nichts und wird normalerweise überschrieben. Das Feld Result von Msg bleibt Null, was bedeutet, daß die Botschaft nicht bearbeitet wurde.

TWindow überschreibt DefWndProc, um die von Windows zur Verfügung gestellten Standardantworten auf Windows-Botschaften aufzurufen.

Siehe auch

TWindow.DefWndProc
TWindowsObject

TWindowsObject.Destroy (Methode)

Syntax

```
procedure Destroy; virtual;
```

Beschreibung

Diese Methode bewirkt, daß das dem Schnittstellenobjekt zugehörige Element entfernt wird, indem sie veranlasst, daß das Fensterobjekt eine **wm_Destroy**-Botschaft erhält. Sie ruft außerdem EnableAutoCreate für jedes aktuell erzeugte untergeordnete Fenster auf. Dies stellt sicher, daß das Schnittstellenobjekt bei der nächsten Erzeugung wie vor dem Entfernen aussieht.

Siehe auch

EnableAutoCreate
TWindowsObject
WMDestroy

TWindowsObject.DisableAutoCreate (Methode)

Syntax

```
procedure DisableAutoCreate;
```

Beschreibung

Bewirkt, daß ein Schnittstellenobjekt als untergeordnetes Fenster nicht mehr zusammen mit seinen übergeordneten Fenstern erzeugt wird.

Rufen Sie DisableAutoCreate für Pop-Up-Fenster und Elemente auf, die später als ihre übergeordneten Fenster erzeugt und angezeigt werden sollen.

Siehe auch

EnableAutoCreate
TWindowsObject

TWindowsObject.DisableTransfer (Methode)

Syntax

```
procedure DisableTransfer;
```

Beschreibung

Schaltet für das Schnittstellenobjekt den Transfermechanismus aus.

Siehe auch

EnableTransfer
TWindowsObject

TWindowsObject.DispatchScroll (Methode)

Syntax

```
procedure DispatchScroll(var Msg: TMessage); virtual;
```

Beschreibung

Wird von WMHScroll und WMVScroll aufgerufen, um Bildlaufmeldungen an entsprechende Objekte weiterzuleiten.

Siehe auch

TWindowsObject
WMHScroll
WMVScroll

TWindowsObject.EnableAutoCreate (Methode)

Syntax

```
procedure EnableAutoCreate;
```

Beschreibung

Stellt sicher, daß ein Schnittstellenobjekt als untergeordnetes Fenster zusammen mit dem übergeordneten Fenster erzeugt und angezeigt wird.

Für Fenster und Steuerelemente ist dieses Merkmal standardmäßig angeschaltet, für Dialogfenster abgeschaltet. Rufen Sie EnableAutoCreate auf, wenn Sie ein Dialogfenster zusammen mit seinen übergeordneten Fenstern erzeugen und anzeigen wollen.

Siehe auch

DisableAutoCreate
TWindowsObject

TWindowsObject.EnableKBHandler (Methode)

Syntax

```
procedure EnableKBHandler;
```

Beschreibung

Stellt Fenstern und nichtmodalen Dialogfenstern eine Tastaturschnittstelle für untergeordnete Steuerelemente zur Verfügung, ähnlich der bei modalen Dialogen. So kann der Anwender z.B. mit der Tabulatortaste Steuerelemente im Fenster auswählen. Als Vorgabe ist diese Funktion ausgeschaltet.

Siehe auch

TWindowsObject

TWindowsObject.EnableTransfer (Methode)

Syntax

```
procedure EnableTransfer;
```

Beschreibung

Schaltet für das Schnittstellenobjekt den Transfermechanismus ein, mit dem der Status von Steuerelementen zu und von einem Transferpuffer übertragen wird.

Siehe auch

DisableTransfer
TWindowsObject

TWindowsObject.FirstThat (Methode)

Syntax

```
function FirstThat(Test: Pointer): PWindowsObject;
```

Beschreibung

Durchläuft die Liste der untergeordneten Fenster und übergibt jedes ihrer Objekte der Booleschen Funktion, auf die Test zeigt, als Parameter. FirstThat gibt einen Zeiger auf das erste untergeordnete Fensterobjekt zurück, für das die Boolesche Funktion True zurückgibt und beendet den Durchlauf. FirstThat gibt **nil** zurück, wenn die Boolsche Funktion immer False zurückgibt. Sie können z.B. eine Methode GetFirstChecked schreiben, die mit FirstThat das erste untergeordnete Wahlfeld im gewählten Zustand findet:

```
function MyWindow.GetFirstChecked: PWindowsObject;

   function IsThisOneChecked(ABox: PWindowsObject); Boolean; far;
   begin
      IsThisOneChecked := ABox^.GetCheck <> 0;
   end;

begin
   GetFirstChecked := FirstThat(@IsThisOneChecked);
end;
```

Siehe auch

TWindowsObject

TWindowsObject.ForEach (Methode)

Syntax

```
procedure ForEach(Action: Pointer);
```

Beschreibung

Durchläuft die Liste der untergeordneten Fenster, ruft für jedes untergeordnete Fenster die Prozedur auf, auf welche Action zeigt, und übergibt ihr jedes Fensterobjekt als Parameter. Sie können z.B. eine Methode CheckAllBoxes schreiben, die mit ForEach jedes untergeordnete Wahlfeld im gewählten Zustand findet:

```
procedure MyWindow.CheckAllBoxes;

   procedure CheckTheBox(ABox: PWindowsObject); far;
   begin
      PCheckBox(ABox)^.Check;
   end;

begin
   ForEach(@CheckAllBoxes);
end;
```

Siehe auch

TWindowsObject

TWindowsObject.GetChildren (Methode)

Syntax

```
procedure GetChildren(var S: Stream);
```

Beschreibung

Liest untergeordnete Fenster vom übergebenen Stream und fügt sie in die entsprechende Liste des übergeordneten Fensters ein.
GetChildren nimmt an, daß ChildList ursprünglich leer ist. Bereits in der Liste enthaltene Zeiger auf untergeordnete Fenster gehen verloren.

Siehe auch

TWindowsObject
PutChildren

TWindowsObject.GetClassName (Methode)

Syntax

```
function GetClassName: PChar; virtual;
```

Beschreibung

Gibt den Vorgabe-Fensterklassennamen zurück, 'TurboWindow'.

Siehe auch

TWindowsObject

TWindowsObject.GetClient (Methode)

Syntax

```
function GetClient: PMDIClient; virtual;
```

Beschreibung

Gibt **nil** für alle Nicht-MDI- Schnittstellenobjekte zurück, die kein MDI-Client-Fenster haben. **TMDIWindow** überschreibt diese Methode, um sein MDI-Client-Fenster zu liefern.

Siehe auch

TWindowsObject

TWindowsObject.GetID (Methode)

Syntax

```
function GetId: Integer; virtual;
```

Beschreibung

Im allgemeinen wird GetID dazu verwendet, den Fensterbezeichner zurückzuliefern. Als Vorgabe gibt GetID -1 zurück.

Siehe auch

TWindowsObject

TWindowsObject.GetWindowClass (Methode)

Syntax

```
procedure GetWindowClass(var AWndClass:TWndClass); virtual;
```

Beschreibung

Dient als Platzhalter für abgeleitete Typen, um den Fensterklassen-Record zu definieren und ihn in AWndClass zurückzugeben. Diese Version von GetWindowClass tut nichts.

Siehe auch

TWindowsObject

TWindowsObject.IsFlagSet (Methode)

Syntax

```
function IsFlagSet(Mask: Byte); Boolean;
```

Beschreibung

Gibt den Status des Bit-Flags im Feld Flags zurück, das durch Mask bezeichnet wird. IsFlagSet gibt True zurück, wenn das Bit-Flag gesetzt, und False, wenn es nicht gesetzt ist.

Siehe auch

SetFlags
TWindowsObject

TWindowsObject.Next (Methode)

Syntax

```
function Next: PWindowsObject;
```

Beschreibung

Gibt einen Zeiger auf das nächste Objekt in der Liste der untergeordneten Fenster des übergeordneten Fensters zurück.

Siehe auch

Previous
TWindowsObject

TWindowsObject.Previous (Methode)

Syntax

```
function Previous: PWindowsObject;
```

Beschreibung

Gibt einen Zeiger auf das vorhergehende Objekt in der Liste der untergeordneten Fenster des übergeordneten Fensters zurück.

Siehe auch

Next
TWindowsObject

TWindowsObject.PutChildren (Methode)

Syntax

```
procedure PutChildren(var S: Stream);
```

Beschreibung

Schreibt alle Fenster der Liste der untergeordneten Fenster in den gegebenen Stream.

PutChildren wird automatisch von Store aufgerufen, kann aber auch dann verwendet werden, wenn Sie den Inhalt eines Fensters speichern wollen, ohne das Fenster selbst zu speichern.

Siehe auch

TWindowsObject
GetChildren

TWindowsObject.Register (Methode)

Syntax

```
function Register: Boolean; virtual;
```

Beschreibung

Registriert die Fensterklasse, die in der Methode GetWindowClass des Objekts definiert und in der Methode GetClassName benannt ist, falls sie noch nicht registriert ist. Register gibt nach erfolgreicher Registrierung der Klasse True zurück.

Siehe auch

TWindowsObject

TWindowsObject.SetFlags (Methode)

Syntax

```
procedure SetFlags(Mask: Byte; OnOff: Boolean);
```

Beschreibung

Setzt oder löscht das Bit-Flag des Feldes Flags, das in Mask bezeichnet ist. Ist OnOff True, wird das Bit gesetzt, sonst gelöscht. Mask kann folgende Werte haben:

wb_KBHandler
wb_FromResource
wb_AutoCreate
wb_MDIChild
wb_Transfer

Siehe auch

IsFlagSet
TWindowsObject
wb_XXXX-Konstanten

TWindowsObject.SetupWindow (Methode)

Syntax

```
procedure SetupWindow; virtual;
```

Beschreibung

Initialisiert das neu erzeugte Schnittstellenelement, normalerweise durch Erzeugen von untergeordneten Fensterelementen (falls es welche gibt).

Die Methode erzeugt nur die untergeordneten Fenster, die automatisch erzeugt werden sollen. Standardmäßig schließt dies keine Dialogfenster ein. Wenn Create ein untergeordnetes Fenster nicht erzeugen kann, setzt es Status auf em_InvalidChild. SetupWindow ruft TransferData auf, um Daten in die neuen untergeordneten Fenster zu kopieren.

Siehe auch

TWindowsObject

TWindowsObject.Show (Methode)

Syntax

```
procedure Show(ShowCmd: Integer); virtual
```

Beschreibung

Show zeigt das Schnittstellenelement auf dem Bildschirm an. ShowCmd. bestimmt die Art der Darstellung:

Wert	Beschreibung
sw_Hide	verdeckt
sw_Show	in der aktuellen Größe und Position des Fensters
sw_ShowMaximized	als Vollbild und aktiv
sw_ShowMinimized	als Symbol und aktiv
sw_ShowNormal	wiederhergestellt und aktiv

Siehe auch

sw_ Show window Konstanten
TWindowsObject

TWindowsObject.Store (Methode)

Syntax

```
procedure Store(var S: TStream);
```

Beschreibung

Speichert das Schnittstellenobjekt auf dem Stream S. Zuerst werden Status, andere Attribute und die Größe von ChildList gespeichert, danach alle untergeordneten Fenster.

Siehe auch

TWindowsObject

TWindowsObject.Transfer (Methode)

Syntax

```
function Transfer(DataPtr: Pointer; TransferFlag: Word): Word; virtual;
```

Beschreibung

Gibt Null zurück. Wird von Nachkommen von **TControl** überschrieben, um Statusdaten zu und vom Transferpuffer zu übertragen. Dann gibt Transfer die Zahl der übertragenen Bytes zurück.

Siehe auch

TWindowsObject

TWindowsObject.TransferData (Methode)

Syntax

```
procedure TransferData(Direction: Word); virtual;
```

Beschreibung

Der Transfermechanismus wird eingeschaltet, indem TransferBuffer auf einen Transfer-Record gesetzt wird. Dann werden Daten zu und von dem Puffer und den beteiligten untergeordneten Fenstern des Schnittstellenobjekts übertragen. TransferData ruft die Methode Transfer für jedes teilnehmende untergeordnete Fenster auf und übergibt ihr einen Zeiger auf den Transferpuffer sowie die Richtung des Transfers in Direction. Direction kann tf_SetData oder tf_GetData sein.

Siehe auch

EnableTransfer
DisableTransfer
SetupWindow
tf_XXXX-Konstanten
TWindowsObject

TWindowsObject.WMActivate (Methode)

Syntax

```
procedure WMActivate(var Msg: TMessage); virtual wm_First + wm_Activate;
```

Beschreibung

Wenn das Schnittstellenobjekt in den Mechanismus der Tastaturverarbeitung einbezogen ist, reagiert die Methode darauf, daß das Schnittstellenobjekt zum aktiven Fenster wird, indem sie SetKBHandler aufruft.

Siehe auch

TApplication.SetKBHandler
TWindowsObject

TWindowsObject.WMClose (Methode)

Syntax

```
procedure WMClose(var Msg: TMessage); virtual wm_First + wm_Close;
```

Beschreibung

Reagiert auf die Aufforderung, das Fenster zu schließen, mit einem Aufruf der Methode CanClose dieses Objekts bzw. der CanClose-Methode des Anwendungsobjekts, wenn dieses Objekt das Hauptfenster der Anwendung ist. Gibt CanClose True zurück, wird das Schnittstellenelement mit Destroy entfernt.

Siehe auch

Destroy
TWindowsObject

TWindowsObject.WMCommand (Methode)

Syntax

```
procedure WMCommand(var Msg: TMessage); virtual wm_First + wm_Command;
```

Beschreibung

Stellt den Mechanismus zur Bearbeitung von Befehlsbotschaften und Botschaften untergeordneter Fenster und zum Aufruf geeigneter Antwortmethoden zur Verfügung.

Siehe auch

TWindowsObject

TWindowsObject.WMDestroy (Methode)

Syntax

```
procedure WMDestroy(var Msg: TMessage); virtual wm_First + wm_Destroy;
```

Beschreibung

Wenn das Objekt das Hauptfenster des Programms ist, informiert WMDestroy Windows beim Entfernen dieses Schnittstellenelements darüber, daß das Programm beendet wird. Eine **wm_Quit**-Botschaft wird gesendet. Ist das Objekt nicht das Hauptfenster, wird das Standardverhalten des Fensters aufgerufen.

Siehe auch

TWindowsObject

TWindowsObject.WMHScroll (Methode)

Syntax

```
procedure WMHScroll(var Msg: TMessage); virtual wm_First + wm_HScroll;
```

Beschreibung

Fängt Botschaften der horizontalen Bildlaufleiste ab und ruft DispatchScroll auf.

Siehe auch

DispatchScroll
TWindowsObject

TWindowsObject.WMNCDestroy (Methode)

Syntax

```
procedure WMNCDestroy(var Msg: TMessage); virtual wm_First + wm_NCDestroy;
```

Beschreibung

Reagiert auf die letzte Botschaft, die ein Schnittstellenelement vor dem Entfernen erhält, indem es HWindow auf Null setzt.

Siehe auch

TWindowsObject

TWindowsObject.WMQueryEndSession (Methode)

Syntax

```
procedure WMQueryEndSession(var Msg: TMsg);virtual wm_First +
wm_QueryEndSession
```

Beschreibung

Wenn das Fenster das Hauptfenster des Programms ist, reagiert diese Methode auf eine **wm_QueryEndSession**-Botschaft, indem sie **TApplication.CanClose** aufruft.

Wenn TApplication.CanClose True zurückgibt, wird Msg.Result auf 1 gesetzt. Sonst wird es auf 0 gesetzt.

TWindowsObject.WMVScroll (Methode)

Syntax

```
procedure WMVScroll(var Msg: TMessage); virtual wm_First + wm_VScroll;
```

Beschreibung

Fängt Botschaften der vertikalen Bildlaufleiste ab und ruft DispatchScroll auf.

Siehe auch

DispatchScroll
TWindowsObject

TWindowsObject.PutChildPtr (Methode)

Syntax

```
procedure PutChildPtr(var S: Stream; var P: PWindowsObject);
```

Beschreibung

Speichert den Zeiger auf ein untergeordnetes Fenster in den Stream S.

PutChildPtr sollte nur innerhalb der Methode Store zum Schreiben der Zeigerwerte verwendet werden, die später durch einen Aufruf von **GetChildPtr** vom Konstruktor Load gelesen werden.

Siehe auch

TWindowsObject
GetSiblingPtr
PutSiblingPtr

TWindowsObject.PutSiblingPtr (Methode)

Syntax

```
procedure PutSiblingPtr(var S: TStream; var P: PWindowsObject);
```

Beschreibung

Speichert P, den Zeiger auf ein Nebenfenster, in den Stream S.

Man sollte PutSiblingPtr nur innerhalb einer Store-Methode verwenden, um Zeigerwerte zu schreiben, die später mit einem Aufruf von **GetSiblingPtr** von einem Load-Konstruktor gelesen werden können.

Siehe auch

GetChildPtr
PutChildPtr
TWindowsObject

TWindowsObject.GetSiblingPtr (Methode)

Syntax

```
procedure GetSiblingPtr(var S: TStream; var P);
```

Beschreibung

Lädt den Zeiger P, einen Zeiger auf ein Nebenfenster, vom Stream S.

Man sollte GetSiblingPtr nur innerhalb eines Load-Konstruktors verwenden, um Zeigerwerte zu lesen, die durch **PutSiblingPtr** in einer Store-Methode geschrieben wurden.

Der Wert in P wird erst gültig, wenn das übergeordnete Fenster seine Load-Funktion abgeschlossen hat.

Siehe auch

GetChildPtr
PutChildPtr
TWindowsObject

TWindowsObject.GetChildPtr (Methode)

Syntax

```
procedure GetChildPtr(var S: TStream; var P);
```

Beschreibung

Lädt P, einen Zeiger auf ein untergeordnetes Fenster, vom Stream S.

Man sollte GetChildPtr nur innerhalb eines Load-Konstruktors verwenden, um Zeigerwerte zu lesen, die durch **PutChildPtr** in einer Store-Methode geschrieben wurden.

Siehe auch

GetSiblingPtr
PutSiblingPtr
TWindowsObject

Hintergrundmodi

Beschreibung

Diese Konstanten bestimmen die Darstellung des Hintergrunds, wenn Text oder Grafik mit einem Schraffurpinsel gezeichnet werden.

Konstante	Bedeutung
Opaque	Hintergrund wird mit aktueller Hintergrundfarbe gefüllt.
Transparent	Hintergrund bleibt unberührt.

In den Funktionen **GetBkMode** und **SetBkMode** verwendet.

bi_xxx

Beschreibung

Diese Konstanten dienen als Werte im Feld biCompression des Windows-Recordtyps **TBitmapInfoHeader** und werden in Aufrufen an **CreateDIBitmap** weitergegeben.

Konstante	Bedeutung
bi_RGB	Bitmap nicht komprimiert
bi_RLE8	Bitmap mit lauflängen-kodiertem Format, 8 Bits pro Pixel.
bi_RLE4	Bitmap mit lauflängen-kodiertem Format,, 4 Bits pro Pixel.

bn_xxx

Beschreibung

Die folgenden Benachrichtigungscodes von **wm_Command** beziehen sich auf Aktionschalter, bei denen eine Aktion eingetreten ist.

Konstante	Bedeutung
bn_Clicked	Zeigt an, daß ein Aktionsschalter betätigt wurde.
bn_DoubleClicked	Zeigt an, daß der Anwender einen Doppelklick auf einem besitzergezeichneten Aktionsschalter oder einem Schaltfeld durchgeführt hat. Gilt für den Stil **bs_OwnerDraw**.

bs_xxx Pinselstile

Beschreibung

Diese Konstanten bestimmen logische Pinselstile und werden im Feld lbStyle des Records LogBrush der Funktion **CreateBrushIndirect** übergeben.

Konstante	Bedeutung
bs_DIBPattern	Pinselmuster aus geräteunabhängigem Bitmap
bs_Hatched	Schraffur-Pinsel.
bs_Hollow	Pinsel malt mit der Farbe des Hintergrunds.
bs_Null	Wie bs_Hollow.
bs_Pattern	Pinselmuster aus Speicherbitmap.
bs_Solid	Kompakter Pinsel.

bs_xxx

Beschreibung

Diese Konstanten bestimmen den Aktionsschalter-Stil mit den Funktionen **CreateWindow** und **CreateWindowEx**.

Konstante	Bedeutung
bs_3State	Der Aktionsschalter-Stil ist ein Feld, das gewählt, nicht gewählt oder grau sein kann. Grau heißt, daß das Feld nicht aktiv ist.
bs_Auto3State	Der Aktionsschalter-Stil entspricht bs_3State, nur ändert sich der Feldstatus beim Anklicken.
bs_AutoCheckBox	Der Aktionsschalter-Stil entspricht bs_CheckBox, nur ändert sich der Feldstatus beim Anklicken.
bs_AutoRadioButton	Der Aktionsschalter-Stil entspricht bs_RadioButton, nur wird die Wahl angezeigt und in allen anderen Feldern der Gruppe entfernt.
bs_CheckBox	Der Aktionsschalter-Stil ist ein Feld, das gewählt und nicht gewählt sein kann. Text steht rechts im Feld.
bs_DefPushButton	Der Aktionsschalter-Stil entspricht bs_PushButton, nur ist dieser Aktionsschalter die Vorgabe, bis ein anderer gewählt wird.
bs_GroupBox	Der Aktionsschalter-Stil ist einFeld, in dem andere Aktionsschalter gruppiert werden. Text steht oben links im Feld.
bs_LeftText	Bei bs_3State, bs_CheckBox, oder bs_RadioButton steht Text links anstatt rechts.
bs_OwnerDraw	Der Aktionsschalter-Stil ist selbst definiert. Neben normalen **wm_Command**-Botschaften wird das Elternfenster angefragt, den Aktionsschalter zu zeichnen, invertieren, und zu desaktivieren.
bs_PushButton	Der Aktionsschalter-Stil ist Text innerhalb des Schalters.
bs_RadioButton	Der Schalter-Stil legt ein kleines rundersSchaltfeld fest, das gewählt oder nicht gewählt werden kann. Text steht rechts vom Schaltfeld. Diese Schaltfelder sind meist gruppiert, um sich gegenseitig ausschließende Auswahlen darzustellen.

cb_xxx

Beschreibung

Diese Werte werden von Botschaften aus Windows-Kombinationsfenstern zurückgegeben, wie z.B. **cb_AddString**. Ein negativer Wert bedeutet einen Fehler.

Konstante	Bedeutung
cb_Err	Ein Fehler trat auf, die Aktion wurde nicht durchgeführt.
cb_ErrSpace	Es gibt nicht genügend Platz im Kombinationsfenster, um die Aktion auszuführen.
cb_Okay	Kein Fehler.

cbm_Init

Beschreibung

Wenn die Konstante cbm_Init im Parameter Usage der Funktion **CreateDIBitmap** weitergegeben wird, muß das Bitmap im Speicher initialisiert werden.

cbn_xxx

Beschreibung

Diese Nachrichtencodes werden in **wm_Command**-Botschaften von Kombinationsfenstern übergeben, wenn eine Aktion stattfand.

Konstante	Bedeutung
cbn_DblClk	Wird übergeben, wenn der Anwender eine Zeichenkette durch einen Doppelklick mit der Maus auswählt.
cbn_DropDown	Informiert den Besitzer eines Kombinationsfenster über die Aktivierung seiner Liste.
cbn_EditChange	Zeigt an, daß der Anwender den Text im Eingabefeld verändert hat.
cbn_EditUpdate	Zeigt an, daß im Eingabefeld veränderter Text dargestellt wird.
cbn_ErrSpace	Wird übergeben, wenn im System kein Speicher mehr frei ist.
cbn_KillFocus	Zeigt an, daß das Kombinationsfenster den Fokus verloren hat.
cbn_SelChange	Wird übergeben, wenn die Auswahl geändert wurde.
cbn_SetFocus	Zeigt an, daß das Kombinationsfenster den Fokus erhalten hat.

cbs_xxx

Beschreibung

Diese Konstanten bestimmen den Kombinationsfensterstil, und werden mit den Funktionen **CreateWindow** und **CreateWindowEx** verwendet.

Konstante	Bedeutung
cbs_AutoHScroll	Rollt automatisch den Text im Editierfeldnach rechts, wenn der Benutzer ein Zeichen amEnde der Zeile eintippt. Wenn dieser Stil nichtgesetzt ist, ist nur Text erlaubt, der in die rechteckige Umrandung paßt.
cbs_DropDown	Ähnlich wie cbs_Simple, außer daß die Liste nicht angezeigt wird, wenn der Benutzer nicht ein Symbol neben dem Auswahlfeld auswählt.
cbs_DropDownList	Ähnlich wie cbs_DropDown, außer daß das Editierfeld von einem statischen Textelement ersetzt wird, das die momentane Auswahl aus der Liste anzeigt.
cbs_HasStrings	Ein anwendungsgezeichnetes Kombinationsfenster enthält Einträge, die aus Strings bestehen. Das Kombinationsfenster behält den Speicher und die Zeiger für die Strings bei, so daß die Anwendung die Botschaft **cb_GetLBText** benutzen kann, um den Text für einen bestimmten Eintrag zurückzuholen.
cbs_NoIntegralHeight	Der Kombinationsfenster-Stil ist genau die Größe zum Erzeugungszeitpunkt. Normalerweise kann die Größe eines Kombinationsfensters geändert werden, damit Einträge nicht abgeschnitten werden.
cbs_OEMConvert	Text, der in den Editierfeldern eines Kombinationsfensters eingegeben wird, wird vom ANSI-Zeichensatz in den OEM-Zeichensatz und dann zurück in ANSI umgewandelt. Dies sorgt für eine genaue Zeichenumwandlung, wenn die Anwendung die Funktion **AnsiToOem** aufruft, um einen ANSI-String im Kombinationsfenster in OEM-Zeichen umzuwandeln. Dieser Stil ist höchst nützlich für Kombinationsfenster, die Dateinamen enthalten und wird nur für Kombinationsfenster verwendet, die mit den Stilen cbs_Simple oder cbs_DropDown erzeugt wurden.
cbs_OwnerDrawFixed	Der Besitzer der Liste ist für das Zeichnen ihres Inhalts verantwortlich; die Einträge in der Liste haben alle die gleiche Höhe.
cbs_OwnerDrawVariable	Der Besitzer der Liste ist für das Zeichnen ihres Inhalts verantwortlich; die Einträge in der Liste haben eine variable Höhe.
cbs_Simple	Die Liste wird zu jeder Zeit angezeigt. Die momentane Auswahl aus der Liste wird im Editierfeld angezeigt.
cbs_Sort	Sortiert automatisch Strings, die in die Liste eingegeben werden. Die Sortierordnug kann bei den Stilen cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable verschieden sein.

cc_xxx

Beschreibung

Diese Konstanten repräsentieren die Kurvendarstellungsmöglichkeiten eines Geräts.

Konstante	Bedeutung
cc_None	Keine Kurvendarstellung
cc_Chord	Kann Bogensegmente
cc_Circles	Kann Kreise.
cc_Ellipses	Kann Ellipsen.
cc_Interiors	Kann Innenbereiche.
cc_Pie	Kann Tortendiagramme.
cc_Styled	Kann verschiedene Linienarten.
cc_Wide	Kann breite Linien.
cc_WideStyled	Kann verschiedene breite Linienarten.

cchDeviceName

Beschreibung

Die Konstante cchDeviceName begrenzt den String für einen Gerätenamen auf 32 Zeichen.

ce_xxx

Beschreibung

Diese Flags geben einen Kommunikationsfehler an, der von der Funktion **GetCommError** gemeldet wurde.

Konstante	Bedeutung
ce_Break	Break festgestellt.
ce_STSTO	Clear-to-send Timeout.
ce_DNS	LPTx device nicht bereit
ce_DSRTO	Data-set-ready Timeout.
ce_Frame	Framing-Fehler
ce_IOE	LPTx I/O Fehler
ce_Mode	Modus nicht verfügbar oder ungültige CID.
ce_OOP	LPTx ohne Papier
ce_Overrun	Zeichen wurde von folgenden Zeichen überrannt und nicht gelesen.
ce_PTO	LPTx timeout.
ce_RLSDTO	Receive-line-signal-detect timeout.
ce_RxOver	Receive-Warteschlange voll
ce_RxParity	Paritätsfehler
ce_TxFull	Transmit Warteschlange voll

cf_xxx

Beschreibung

Diese Konstanten identifizieren Zwischenablage-Formate beim Austausch von Daten mit der Windows-Zwischenablage.

Konstante	Bedeutung
cf_Bitmap	Windows Bitmap.
cf_DIB	Windows geräteunabhängige Bitmap
cf_DIF	Software Arts' Daten-Austauschformat.
cf_DSPBitmap	Private Bitmap-Information.
cf_DSPMetaFilePict	Private Metafile-Information.
cf_DSPText	Private Textanzeigeinformation.
cf_MetaFilePict	Windows Metafile.
cf_OEMText	Text mit OEM Zeichensatz
cf_OwnerDisplay	Anzeige wird von der Applikation, die die Daten in der Zwiischenablage bereitstellt, aktualisiert.
cf_Palette	Farbpalette für Zwischenablagedaten
cf_PrivateFirst	Erster einer Reihe von privaten Datenformaten. Handles zu Daten dieser Formate werden nicht automatisch freigegeben und sollten dies vor Ende des Programms.
cf_PrivateLast	Letzter einer Reihe von privaten Datenformaten.
cf_SYLK	Microsoft Symbolisches Linkformat.
cf_Text	Text.
cf_TIFF	Tag-Image Dateiformat.

Diese Konstanten werden beim dynamischen Datenaustausch (DDE) und in folgenden Funktionen verwendet:

EnumClipboardFormats
GetClipboardData
GetClipboardFormatName
GetPriorityClipboardFormat
IsClipboardFormatAvailable
SetClipboardData

clip_xxx

Beschreibung

Diese Konstanten definieren die Clipping-Präzision von Schriften. Die Clipping-Präzision bestimmt, wie Zeichen, die mit der Funktion **CreateFont** erzeugt wurden, abgeschnitten werden sollen, wenn sie teilweise außerhalb des Clipping-Bereiches sind. Er kann einen der folgenden Werte annehmen:

clip_Character_Precis
clip_Default_Precis
clip_Stroke_Precis

color_xxxx

Beschreibung

In der Windows-Benutzerschnittstelle können Anwender mit der Funktion **SetSysColors** Farben direkt setzen.

Konstante	Bedeutung
color_ActiveBorder	Aktiver Fensterrand
color_ActiveCaption	Actor Fenstertitel
color_AppWorkSpace	Hintergrund von MDI-Rahmenfenstern
color_Background	Windows Desktop.
color_BtnFace	Oberfläche von Aktionsschaltern
color_BtnShadow	Rand von Aktionsschaltern
color_BtnText	Text von Aktionsschaltern
color_CaptionText	Titeltext, Größenfeld und Bildlaufleistenfeld
color_GrayText	Grauer Text.
color_Highlight	Gewählte Elemente
color_HighlightText	Text gewählter Elemente
color_InactiveBorder	Nicht aktiver Fensterrand
color_InactiveCaption	Nicht aktiver Titel
color_Menu	Menüleisten-Hintergrund
color_MenuText	Menüleisten-Text
color_ScrollBar	Bildlaufleisten-Hintergrund
color_Window	Fenster-Hintergrund
color_WindowFrame	Fensterrahmen
color_WindowText	Fenstertext

com_xxx

Beschreibung

Diese Flags können als Bitmaske verwendet werden, um Bits des Feldes Flags eines **TComStat**-Records abzufragen oder zu setzen.

Konstante	Bit
com_CtsHold	fCtsHold
com_DsrHold	fDsrHold
com_RlsdHold	fRlsdHold
com_XoffHold	fXoffHold
com_XoffSent	fXoffSent
com_Eof	fEof
com_Txim	fTxim

Schnittstellenkonfigurations-Konstanten

Beschreibung

Eine der folgenden Konstanten wird im Feld Parity des Records **TDCB** verwendet. Diese Konstanten zeigen den Paritätstyp an, der bei serieller Übertragung verwendet wird.

Konstante	Paritätseinstellung
EvenParity	Gerade Parität
MarkParity	Markierung
NoParity	Kein Paritätsbit
OddParity	Ungerade Parität
SpaceParity	Leer

Eine der folgenden Konstanten wird im Feld StopBits des Records TDCB verwendet. Diese Konstanten zeigen die Zahl der Stopbits an, die bei serieller Übertragung verwendet wird.

Konstante	Zahl der Stopbits
OneStopBit	Eines
One5StopBits	1.5
TwoStopBits	Zwei

cp_xxx

Beschreibung

Diese Konstanten bedeuten die Abschneidemöglichkeiten des Geräts.

Konstante	Bedeutung
cp_None	Kein Abschneiden der Ausgabe
cp_Rectangle	Abschneiden in Rechtecken

cs_xxx

Beschreibung

Diese Stilkonstanten von Fensterklassen werden im Feld Style der Datenstruktur **TWNDCLASS** verwendet und können mit or kombiniert werden.

Konstante	Bedeutung
cs_ByteAlignClient	Ausrichtung des Client-Bereichs eines Fensters auf eine Byte-Grenze (in der x-Richtung).
cs_ByteAlignWindow	Ausrichtung eines Fenster auf eine Byte-Grenze (in der x-Richtung).
cs_ClassDC	Gibt der Fensterklasse ihren eigenen Gerätekontext (geteilt mit Instanzen)
cs_DblClks	Sendet Doppelklickbotschaften an ein Fenster.
cs_GlobalClass	Die globale Klasse einer Anwendung ist für alle Anwendungen verfügbar.
cs_HRedraw	Neuzeichnung des kompletten Fensters, wenn sich die horizontale Größe ändert.
cs_NoClose	Sperrt die Schließoption im Systemmenü.
cs_OwnDC	Gibt jeder Fensterinstanz ihren eigenen Gerätekontext. Jeder Gerätekontext benötigt ungefähr 800 Byte Speicher.
cs_ParentDC	Gibt den Gerätekontext des übergeordneten Fensters an die Fensterklasse.
cs_SaveBits	Sichert einen Bildschirmbereich, der von einem Fenster verdeckt wird. Windows benutzt das gesicherte Bitmap, um den Bildschirmbereich wiederherzustellen, wenn ein Fenster bewegt wird. Nur minimal verwenden!
cs_VRedraw	Neuzeichnung des kompletten Fensters, wenn sich die vertikale Größe ändert.

ctlcolor_xxx

Beschreibung

Flags für den Elementtyp oder das Dialogfenster, das gezeichnet werden soll, für die Botschaft **wm_CtlColor**.

Konstante	Bedeutung
ctlcolor_Btn	Aktionsschalter
ctlcolor_Dlg	Dialogfenster
ctlcolor_Edit	Editorelement
ctlcolor_ListBox	Liste
ctlcolor_MsgBox	Meldungsfeld
ctlcolor_ScrollBar	Bildlaufleiste
ctlcolor_Static	Statisches Element

cw_UseDefault

Beschreibung

cw_UseDefault wird als Parameter der Funktionen **CreateWindow** oder **CreateWindowEx** verwendet.

Windows gibt damit dem zu erzeugenden Fenster eine Vorgabegröße oder -position.

dc_xxx

Beschreibung

Diese Konstanten werden beim Aufrufen von **DeviceCapabilities** als Werte für den Parameter Index verwendet, um spezifische Eigenschaften eines Druckertreibers anzugeben.

Konstante	Bedeutung
dc_BinNames	Information über Papiereinzüge
dc_Bins	Liste der Papiereinzüge
dc_Driver	Versionsnummer des Druckertreibers
dc_Duplex	Ebene der Duplex-Druckerunterstützung
dc_Extra	Anzahl Bytes am Ende des Records **TDevMode** für gerätespezifische Daten
dc_Fields	Das Feld dmFields des Records TDevMode
dc_MaxExtent	Maximale Papiergröße
dc_MinExtent	Minimale Papiergröße
dc_Papers	Liste möglicher Papiergrößen
dc_Papersize	Genaue Abmessungen der möglichen Papiergrößen
dc_Size	Das Feld dmSize des Records TDevMode
dc_Version	Spezifikationsversion des Druckers

dcb_xxx

Beschreibung

Diese Flags können als Bitmaske verwendet werden, um Bits des Feldes Flags eines **TDCB**-Records abzufragen oder zu setzen.

Konstante	Bit
dcb_Binary	fBinary
dcb_RtsDisable	fRtsDisable
dcb_Parity	fParity
dcb_OutxCtsFlow	fOutxCtsFlow
dcb_OutxDsrFlow	fOutxDsrFlow
dcb_DtrDisable	fDtrDisable
dcb_OutX	fOutX
dcb_InX	fInX
dcb_PeChar	fPeChar
dcb_Null	fNull
dcb_ChEvt	fChEvt
dcb_Dtrflow	fDtrflow
dcb_Rtsflow	fRtsflow

dde_xxx Flags

Beschreibung

Diese Flags können als Bitmaske verwendet werden, um Bits des Feldes Flags eines Records des Typs **TDDEAdvise**, **TDDEData**, oder **TDDEPoke** abzufragen oder zu setzen.

Konstante	Feld
dde_AckReq	fAckReq (TDDEAdvise oder TDDEData).
dde_DeferUpdt	fDeferUpd (TDDEAdvise).
dde_Response	fResponse (TDDEData).
dde_Release	fRelease (TDDEData oder TDDEPoke).

dde_xxx

Beschreibung

Diese Konstanten werden an DDE-aktive Programme übergeben, und zwar in Feldern der Records **TDDEAdvise**, **TDDEData** und **TDDEPoke**.

Folgende Bitmasken spezifizieren Bits im Record **TDDEAck**:

Konstante	BIt mask
dde_Ack	Das Ack-bit von TDDEAck.Status word. Sind LParamLo und dde_Ack = 1, wird die Anfrage akzeptiert, bei = 0 verweigert.
dde_AppReturnCode	Für Programm-spezifische Rückgabewerte.
dde_Busy	Das Busy-bit von TDDEAck.Status word. Wenn die Anfrage abgelehnt wird, zeigen LParamLo und dde_Busy = 1, daß das Programm nicht antworten konnte.

Gerätefähigkeits-Konstanten

Beschreibung

Diese Konstanten gelten für die Fähigkeiten eines Gerätekontexts und werden in der Funktion **GetDeviceCaps** verwendet.

Konstante	Bedeutung
AspectX	Relative Pixelbreite
AspectXY	Diagonale Pixellänge
AspectY	Relative Pixelhöhe
BitsPixel	Bits pro Pixel.
ClipCaps	Schneiden. Gibt **cp_xxx**-Konstante zurück.
ColorRes	Farbauflösung in Bits pro Pixel.
CurveCaps	Kurven. Gibt **cc_xxx-**Konstante zurück.
DriverVersion	Treiberversion. Z.B. ist$100 1.0.
HorzRes	Horizontale Breite in Pixel.
HorzSize	Horizontale Größe in Millimeter.
LineCaps	Linien. Gibt **lc_xxx-**Konstante zurück.
LogPixelsX	Pixel pro Horizontalzoll
LogPixelsY	Pixel pro Vertikalzoll
NumBrushes	Pinsel das Geräts
NumColors	Farben das Geräts
NumFonts	Schriften das Geräts
NumMarkers	Markers das Geräts
NumPens	Stifte das Geräts
NumReserved	reservierte Einträge der Palette.
PDeviceSize	Größe für Gerätebeschreibung
Planes	Zahl der farbebenen
PolygonalCaps	Polygone. **pc_xxx-**Konstante.
RasterCaps	Raster. **rc_xxx-**Konstante.
SizePalette	Einträge der physikalischen Palette.
Technology	Geräteklassifikation. gibt **dt_xxx-** Konstante zurück.
TextCaps	Textfähigkeiten. Gibt **tc_xxx-** Konstante zurück.
VertRes	Vertikale Breite in Pixel.
VertSize	Vertikale Größe in Millimeter.

dib_xxx

Beschreibung

Diese Konstanten bestimmen, wie auf Farben für geräteunabhängige Bitmaps zugegriffen wird.

Konstante	Bedeutung
dib_Pal_Colors	Farbtabelle ist ein Array von 16-Bit-Indices in die aktuelle logische Tabelle.
dib_RGB_Colors	Farbtabelle enthält RGB-Werte.

Sie werden in den folgenden Funktionen verwendet:

CreateDIBitmap
CreateDIBPatternBrush
GetDIBits
SetDIBits
SetDIBitsToDevice
StretchDIBits

dlgc_xxx

Beschreibung

Dialogeingaben, die das Programm, nicht Windows, bearbeitet. Rüchgabewerte der Botschaft **wm_GetDlgCode**.

Konstante	Bedeutung
dlgc_DefPushButton	Vorgabe-Aktionsschalter-Botschaften
dlgc_HasSetSel	em_SetSel-Botschaften
dlgc_RadioButton	Schaltfeld-Botschaften
dlgc_UndefPushButton	Schaltfeld-Botschaften
dlgc_WantAllKeys	Tastatur-Botschaften
dlgc_WantArrows	Pfeiltasten-Botschaften
dlgc_WantChars	wm_Char-Botschaften
dlgc_WantMessage	Tastatur-Botschaften für das Element
dlgc_WantTab	Tabulator-Botschaften

DlgWindowExtra

Beschreibung

Wenn Sie eine neue Klasse für ein Dialogfenster aus einer Ressource definieren, setzen Sie das Feld cbWndExtra desRecordtyps **TWndClass** auf DlgWindowExtra.

dm_xxx Feldauswahlbits

Beschreibung

Diese Konstanten repräsentieren Bits zur Feldauswahl im Feld dmFields des Records **TDevMode**. Wenn das bestimmte Bit gesetzt ist, wurde das entsprechende Feld des Records initialisiert.

Konstante	Feld
dm_Color	dmColor
dm_Copies	dmCopies
dm_DefaultSource	dmDefaultSource
dm_Duplex	dmDuplex
dm_Orientation	dmOrientation
dm_PaperLength	dmPaperLength
dm_PaperSize	dmPaperSize
dm_PaperWidth	dmPaperWidth
dm_PrintQuality	dmPrintQuality
dm_Scale	dmScale

dm_xxx

Beschreibung

Diese Konstanten werden im Parameter Mode der Funktion ExtDeviceMode für bestimmte Operationen verwendet. Mode kann folgende Werte, oder eine Kombination dieser, annehmen.

Konstante	Bedeutung
dm_Copy	Schreibt die Einstellungen des Druckertreibers in den Record **TDevMode** in DevModeOutput.
dm_Modify	Ändert die Einstellungen des Druckertreibers nach den Werten des Records **TDevMode** aus DevModeInput.
dm_Prompt	Stellt den Drucker nach den Eingaben im Dialogfenster ein.
dm_Update	Aktualisiert die Druckerparameter und die Datei WIN.INI mit den neuen Einstellungen.

dmbin_xxx

Beschreibung

Diese Konstanten werden im Feld dmDefaultSource des Records **TDevMode** verwendet, in welchem die Standard-Papierzufuhr festgelegt wird.

Konstante

dmbin_Auto
dmbin_Cassette
dmbin_Envelope
dmbin_EnvManual
dmbin_LargeCapacity
dmbin_LargeFmt
dmbin_Lower
dmbin_Manual
dmbin_Middle
dmbin_OnlyOne
dmbin_SmallFmt
dmbin_Tractor
dmbin_Upper

dmcolor_xxx

Beschreibung

Diese Konstanten zeigen im Feld dmColor des Records **TDevMode**Druckmöglichkeiten in Farbe.

Konstante	Bedeutung
dmcolor_Monochrome	Drucker druckt nur schwarz auf weiß
dmcolor_Color	Drucker druckt farbig

dmdup_xxx

Beschreibung

Diese Konstanten werden im Feld dmDuplex des Records **TDevMode** für doppelseitiges Drucken verwendet.

Konstante	Bedeutung
dmdup_Horizontal	Horizontaldruck
dmdup_Simplex	Druck nur auf einer Seite
dmdup_Vertical	Vertikaldruck

dmorient_xxx

Beschreibung

Diese Konstanten zeigen im Feld dmOrientation des Records **TDevMode** die Druckrichtung.

Konstante	Bedeutung
dmorient_Portrait	Hochformat
dmorient_Landscape	Querformat

dmpaper_xxx

Beschreibung

Diese Konstanten zeigen im Feld dmPaperSize des Records **TDevMode** die Papiergröße.

Konstante	Bedeutung
dmpaper_10X14	10 x 14 inches
dmpaper_11X17	11 x 17 inches
dmpaper_A3	297 x 420 mm
dmpaper_A4	210 x 297 mm
dmpaper_A4Small	210 x 297 mm
dmpaper_A5	148 x 210 mm
dmpaper_B4	250 x 354 mm
dmpaper_B5	182 x 257 mm
dmpaper_CSheet	C size sheet
dmpaper_DSheet	D size sheet
dmpaper_Env_10	4 1/8 x 9 1/2 inches
dmpaper_Env_11	4 1/2 x 10 3/8 inches
dmpaper_Env_12	4 3/4 x 11 inches
dmpaper_Env_14	5 x 111/2 inches
dmpaper_Env_9	3 7/8 x 8 7/8 inches
dmpaper_ESheet	E size sheet
dmpaper_Executive	7 1/2 x 10 inches
dmpaper_Folio	8 1/2 x 13 inches
dmpaper_Ledger	17 x 11 inches
dmpaper_Legal	8 1/2 x 14 inches
dmpaper_Letter	8 1/2 x 11 inches
dmpaper_LetterSmall	8 1/2 x 11 inches
dmpaper_Note	8 1/2 x 11 inches
dmpaper_Quarto	215 x 275 mm
dmpaper_Statement	5 1/2 x 8 1/2 inches
dmpaper_Tabloid	11 x 17 inches

dmres_xxx

Beschreibung

Diese Konstanten zeigen im Feld dmPrintQuality des Records **TDevMode** die Auflösung beim Druck.

Konstante	Bedeutung
dmres_Draft	Draft-Modus
dmres_Low	Niedrige Druckqualität
dmres_Medium	Mittlere Druckqualität
dmres_High	Hohe Druckqualität

drive_xxx

Beschreibung

Diese Konstanten bestimmen den Diskettentyp und werden von der Funktion **GetDriveType** zurückgegeben.

Konstante	Bedeutung
drive_Fixed	Diskette kann nicht aus dem Laufwerk entfernt werden
drive_Remote	Diskette befindet sich entfernt im Netz
drive_Removeable	Diskette kann aus dem Laufwerk entfernt werden

ds_xxx

Beschreibung

Diese Konstanten bestimmen den Stil beim Erzeugen von Dialogfenstern mit den Funktionen **CreateWindow** und **CreateWindowEx**.

Konstante	Bedeutung
ds_AbsAlign	Das Dialogfenster wird an der oberen linken Ecke des Bildschirms anstatt an der oberen linken Ecke des Fensterers der Applikation, die das Dialogfenster besitzt, ausgerichtet.
ds_LocalEdit	Bestimmt, daß Textspeicher für Eingabefelder im lokalen Datensegment einer Anwendung reserviert wird. Dies erlaubt die Nutzung der Botschaften **em_GetHandle** und **em_SetHandle**. Wenn dieser Stil nicht festgelegt ist, befindet sich der Datenbereich für Eingabefelder in einem separaten globalen Datenblock.
ds_ModalFrame	Die Dialogbox erhält einen modalen Rahmen. Die .**ws_xxx (Fenster-Stile)**, ws_Caption und ws_SysMenu können mit ds_ModalFrame kombiniert werden.
ds_NoIdleMsg	Bestimmt, daß keine **wm_EnterIdle**-Botschaft gesendet werden, wärend das Dialogfenster angezeigt wird.
ds_SetFont	Bestimmt, daß eine andere Schrift als die Systemschrift verwendet wird, um den Text in dem Dialogfenster darzustellen. Hierfür muß eine erweiterte **dialog template**-Struktur verwendet werden. Diese setzt sich aus der Standard- und einer darauffolgenden Schriftart-Informations-Struktur zusammen. Ein Dialog vom Stil ds_SetFont erhält eine **wm_SetFont**-Botschaft, bevor die einzelnen Dialagelemente erzeugt werden.
ds_SysModal	Bestimmt ein modales Systemfenster.

dt_xxx

Beschreibung

Diese Konstanten repräsentieren den verwendeten Gerätetyp.

Konstante	Bedeutung
dt_CharStream	Zeichen-Stream
dt_DispFile	Display Datei
dt_MetaFile	Metafile
dt_Plotter	Vector-Plotter
dt_RasCamera	Raster-Kamera
dt_RasDisplay	Rasteranzeige
dt_RasPrinter	Rasterdrucker

dt_xxx Formatoptionen

Beschreibung

Diese Konstanten bestimmen Formatoptionen für die Funktion **DrawText**.

Konstante	Bedeutung
dt_Bottom	Unten ausgerichtet
dt_CalcRect	Berechne das Rechteck um den Text, ohne den Text zu schreiben.
dt_Center	Horizontal ausgerichtet
dt_ExpandTabs	Expandiert Tabulatoren
dt_ExternalLeading	Schließt die Oberlinien der Schrift ein.
dt_Internal	Schließt die Unterlinien der Schrift ein.
dt_Left	Links ausgerichtet
dt_NoClip	Ohne Abschneiden
dt_NoPrefix	Ohne Präfixe wie '&'.
dt_Right	Rechts ausgerichtet
dt_SingleLine	Nur eine Linie
dt_TabStop	Tabulatoren setzen
dt_Top	Oben ausgerichtet
dt_VCenter	Vertikal zentriert
dt_WordBreak	Wortumbruch

en_xxx

Beschreibung

Diese Nachrichten werden in **wm_Command**-Botschaften von Editorelementen übergeben und zeigen die erfolgte Aktion an.

Konstante	Bedeutung
en_Change	Zeigt an, daß der Anwender eine Aktion durchgeführt hat, die möglicherweise den Inhalt des Textes verändert hat.
en_ErrSpace	Zeigt an, daß kein zusätzlicher Platz im Eingabefeld vorhanden ist.
en_HScroll	Zeigt an, daß der Benutzer die horizontale Bildlaufleiste eines Eingabefeldes mit der Maus angeklickt hat. Das übergeordnete Fenster wird vor dem Aktualisieren des Bildschirms benachrichtigt.
en_KillFocus	Zeigt an, daß das Eingabefeld den Fokus verloren hat.
en_MaxText	Zeigt an, daß die aktuelle Einfügung die festgelegte Anzahl von Zeichen für dieses Eingabefeld überschritten hat.
en_SetFocus	Zeigt an, daß das Eingabefeld den Fokus erhalten hat.
en_Update	Zeigt an, daß das Eingabefeld veränderten Text darstellt.
en_VScroll	Zeigt an, daß der Anwender die vertikale Bildlaufleiste des Eingabefeldes mit der Maus angeklickt hat. Das übergeordnete Fenster wird vor dem Aktualisieren des Bildschirms benachrichtigt.

EnumFonts Bitmasken

Beschreibung

Diese Masken enthüllen, kombiniert mit dem Argument FontType der Callback-Funktion **EnumFonts**, Charakteristika der verwendeten Schrift.

Konstante	Bedeutung
Raster_FontType	Ist die Kombination 1, Rasterschrift, 0 = Vektorschrift
Device_FontType	Ist die Kombination 1, Druckereigene Schrift, bei 0 GDI-Schrift

es_xxx

Beschreibung

Diese Konstanten bestimmen den Editorstil beim Erstellen mit den Funktionen **CreateWindow** und **CreateWindowEx**.

Konstante	Bedeutung
es_AutoHScroll	Rollt automatisch Text um 10 Zeichen nach rechts, wenn der Benutzer am Ende der Zeile ein Zeichen eintippt. Wenn der Benutzer die RETURN-Taste drückt, rollt das Steuerelement den ganzen Text zur Nullposition zurück.
es_AutoVScroll	Rollt automatisch Text um eine Seite nach oben, wenn der Benutzer in der letzten Zeile RETURN drückt.
es_Center	Zentriert Text in einem mehrzeiligen Editierfeld. Wird nur in Kombination mit es_MultiLine benutzt.
es_Left	Richtet Text linksbündig aus. Wird nur in Kombination mit es_MultiLine benutzt.
es_LowerCase	Wandelt alle Zeichen in Kleinbuchstaben um, sowie sie im Editierfeld eingegeben werden.
es_MultiLine	Bestimmt ein mehrzeiliges Editierfeld (die Voreinstellung ist einzeilig). Wenn der Stil es_AutoVScroll angegeben wird, zeigt das Editierfeld so viele Zeilen wie möglich und rollt vertikal, wenn der Benutzer die RETURN-Taste drückt. Wenn es_AutoVScroll nicht gesetzt ist und in der letzten Zeile des Editierfeldes RETURN gedrückt wird ertönt ein Warnton, der anzeigt, daß keine weiteren Zeilen hinzugefügt werden können.
es_NoHideSel	Normalerweise macht ein Editierfeld die Auswahl unsichtbar, wenn das Steuerelement den Fokus verliert und invertiert das Auswahlfenster, wenn das Steuerelement den Fokus erhält. Die Angabe von es_NoHideSel löscht diese voreingestellte Aktion.
es_OEMConvert	Text, der im Editierfeld eingegeben wird, wird vom ANSI-Zeichensatz in den OEM-Zeichensatz und dann zurück in ANSI umgewandelt. Dies sorgt für eine genaue Zeichenumwandlung, wenn die Anwendung die Funktion **AnsiToOem** aufruft, um einen ANSI-String in dem Editierfeld in OEM-Zeichen umzuwandeln. Dieser Stil ist höchst nützlich für Editierfelder, die Dateinamen enthalten.
es_Password	Stellt alle Zeichen als Stern (*) dar, sobald sie im Editierfeld eingegeben werden. Eine Anwendung kann die Botschaft **em_SetPasswordChar** verwenden, um das Zeichen zu verändern, das angezeigt wird.
es_Right	Richtet Text in einem mehrzeiligen Editierfeld rechtsbündig aus. Wird nur in Kombination mit es_MultiLine verwendet.
es_UpperCase	Wandelt alle Zeichen in Großbuchstaben um, sobald sie in das Editierfeld eingegeben werden.

Escape-Konstanten für serielle Schnittstellen

Beschreibung

Diese Konstanten bestimmen einen Funktionscode für das Kommunikationsgerät. Verwenden Sie diese in Aufrufen der Funktion **EscapeCommFunction**.

Konstante	Bedeutung
ClrDTR	Löscht data-set-ready Signal.
ClrRTS	Löscht request-to-send Signal.
ResetDev	Reset des Geräts, wenn möglich
SetDTR	Löscht data-terminal-ready Signal.
SetRTS	Löscht request-to-send signal.
SetXoff	Simuliert Empfang eines XOFF .
SetXon	Simuliert Empfang eines XON .

eto_xxx

Beschreibung

Diese Optionen bestimmen, wie **ExtTextOut** den Hintergrund malt.

Konstante	Bedeutung
eto_Clipped	Text wird am Rechteckrand abgeschnitten
eto_Opaque	Hintergrundfarbe füllt das Rechteck.

ev_xxx

Beschreibung

Diese Konstanten dienen als Zeiger auf Ereignismasken eines Kommunikationsgeräts und sind Rückgabewerte in **SetCommEventMask**.

Konstante	Bedeutung
ev_Break	Break erhalten
ev_CTS	Clear to send Status geändert
ev_DSR	Data set ready Status geändert
ev_Err	Line status Fehler
ev_PErr	Druckerfehler
ev_Ring	Telefon läutet
ev_RLSD	Receive line Status geändert
ev_RxChar	Zeichen erhalten
ev_RxFlag	Ereigniszeichen aus dem Steuerblock erhalten
ev_TxEmpty	Übertragungs-Warteschlange leer

Floodfill-Flags

Beschreibung

Fülltyp der Funktion **ExtFloodFill**.

Konstante	Bedeutung
FloodFillBorder	Bereich bis zur angegebenen Farbe wird gefüllt, mittels der Funktion **FloodFill**.
FloodFillSurface	Bereich mit der angegebenen Farbe wird gefüllt. Dies wird hauptsächlich für komplexe mehrfarbige Flächen verwendet.

ff_xxx

Beschreibung

Diese Konstanten bestimmen den Zeichensatz für Schriften, die mit der Funktion **CreateFont** erzeugt werden.

Konstante	Bedeutung
ff_Decorative	Schriften wie Old English.
ff_DontCare	Weiß nicht oder egal
ff_Modern	Konstante Strichbreite mit oder ohne Serifen, wie Pica, Elite, und Courier.
ff_Roman	Variable Strichbreite mit Serifen, wie Times Roman und Century Schoolbook.
ff_Script	Handerstellt Schrift wie Script.
ff_Swiss	Variable Strichbreite ohne Serifen, wie Helvetica und Swiss.

Schrift-Zeichensatz-Flags

Beschreibung

Diese Konstanten bestimmen den Zeichensatz für Schriften, die mit der Funktion **CreateFont** erzeugt werden.

ANSI_CharSet
Symbol_CharSet
ShiftJIS_CharSet
OEM_CharSet

Schriftausgabequalitäts-Flags

Beschreibung

Diese Konstanten bestimmen die Wiedergabequalität von Schriften, die mit der Funktion **CreateFont** erzeugt werden.

Konstante	Bedeutung
Default_Quality	Erscheinung unwichtig
Draft_Quality	Erscheinung begrenzt wichtig
Proof_Quality	Zeichenqualität ist wichtiger als logische Schriftattribute

Schriftneigungs-Flags

Beschreibung

Diese Konstanten bestimmen Neigung der Schriften, die mit der Funktion **CreateFont** erzeugt werden.

Default_Pitch
Fixed_Pitch
Variable_Pitch

fw_xxx

Beschreibung

Diese Konstanten bestimmen Gewichtung der Schriften, die mit der Funktion **CreateFont** erzeugt werden.

fw_DontCare
fw_Thin
fw_ExtraLight, fw_UltraLight
fw_Light
fw_Normal, fw_Regular
fw_Medium
fw_SemiBold, fw_DemiBold
fw_Bold, fw_Black
fw_ExtraBold, fw_UltraBold
fw_Heavy

gcl_xxx

Beschreibung

Diese Konstanten bestimmen den Byte-Offset der Fensterklasseninformation, die mit den Funktionen **GetClassLong** oder **SetClassLong** ermittellt bzw. gesetzt wird.

Konstante	Bedeutung
gcl_MenuName	Zeiger auf den Menünamen (SetClassLong only).
gcl_WndProc	Zeiger auf die Fensterfunktion

gcw_xxx

Beschreibung

Diese Konstanten bestimmen den Byte-Offset der Fensterklasseninformation, die mit den Funktionen **GetClassWord** oder **SetClassWord** ermittelt oder verändert wird.

Konstante	Bedeutung
gcw_CBClsExtra	Holt oder setzt die Anzahl der zusätzlichen Bytes in der Klasseninformation.
gcw_CBWndExtra	Holt oder setzt die Anzahl der zusätzlichen Bytes in der Fensterinformation.
gcw_HBrBackground	Handle zum Hintergrund-Pinsel.
gcw_HCursor	Handle zum Cursor.
gcw_HIcon	Handle zum Icon.
gcw_HModule	Handle zum Modul. (GetClassWord only).
gcw_Style	Stil-Bits der Fensterklasse

gmem_xxx

Beschreibung

Charakteristika des globalen Speicherblocks, der mit den Funktionen **GlobalAlloc** und **GlobalReAlloc** eingerichtet wurde.

Sie werden auch in **GlobalFlags** und **GetFreeSpace** verwendet.

Konstante	Bedeutung
gmem_DDEShare	Reserviert gemeinsam benutzbaren Speicherplatz. Wird nur für dynamischen Datenaustausch (DDE) benutzt. Es ist jedoch zu beachten, daß Windows automatisch Speicher verwirft, der mit diesem Attribut reserviert wurde, wenn die Anwendung, die diesen Speicher belegte, die Ausführung beendet.
gmem_Discardable	Reserviert verwerfbaren Speicher. Kann nur mit gmem_Moveable benutzt werden.
gmem_Discarded	Speicherblock wurde verworfen. (GlobalFlags only.)
gmem_Fixed	Belegt festen Speicher.
gmem_LockCount	Kombiniert mit dem niederwertigen Byte des Wiedergabewerts von GlobalFlags wird der Referenzzähler zurückgegeben. (GlobalFlags only.)
gmem_Lower	Wie gmem_Not_Banked.
gmem_Modify	Globale Speicherflags können geändert werden, wenn dieses Flag gesetzt wird. Findet nur bei GlobalReAlloc Verwendung.
gmem_Moveable	Reserviert verschiebbaren Speicher.
gmem_NoCompact	Weder verschiebt Windows noch verwirft es Speicherblöcke um genügend freien Speicher zu erhalten.
gmem_NoDiscard	Es werden keine Speicherblöcke verworfen um freien Speicher zu erhalten.
gmem_Not_Banked	Belegt Speicher, der nicht in Bänke aufgeteilt ist.
gmem_Notify	Ruft die Benachrichtigungsroutine auf, falls das Speicherobjekt jemals verworfen wird.
gmem_Share	Wie gmem_DDEShare.
gmem_ZeroInit	Initialisiert Speicherinhalte mit Null.

gw_xxx

Beschreibung

Diese Konstanten bestimmen die Beziehung zwischen dem erwünschten Fenster und dem Fenster, das mittels Aufrufen von **GetNextWindow** und **GetWindow** bereitgestellt wird. GetNextWindow akzeptiert nur gw_HWndNext und gw_HWndPrev.

Konstante	Bedeutung
gw_Child	Das erste untergeordnete
gw_HWndFirst	Das erste Nachbarfenster eines untergeordneten Fensters
gw_HWndLast	Das letzte Nachbarfenster eines untergeordneten Fensters
gw_HWndNext	Das nächste Fenster der Liste
gw_HWndPrev	Das vorherige Fenster der Liste
gw_Owner	Das übergeordnete Fenster

gwl_xxx

Beschreibung

Diese Konstanten bestimmen den Byte-Offset der Fensterattribute, die mit den Funktionen **GetWindowLong** oder **SetWindowLong** ermittelt oder verändert werden.

Konstante	Bedeutung
gwl_ExStyle	Erweiterter Fensterstil
gwl_Style	Fensterstil
gwl_WndProc	Zeiger zur Fensterfunktion

gww_xxx

Beschreibung

Diese Konstanten bestimmen Byte offset der Fensterinformation, die mit den Funktionen **GetWindowWord** oder **SetWindowWord** eingeholt oder verändert werden.

Konstante	Bedeutung
gww_HInstance	Instanz-ID des Moduls, dem das Fenster gehört
gww_HWndParent	Elternfenster (GetWindowWord only.)
gww_ID	Element-ID des untergeordneten Fensters

help_xxx

Beschreibung

Diese Konstanten geben der Windows-Hilfe den Hilfetyp des Programms an, der in Aufrufen von **WinHelp** verwendet wird.

Konstante	Bedeutung
help_Context	Hilfe für den Kontext im der im Parameter Data übergeben wird.
help_HelpOnHelp	Hilfe zum Hilfesystem
help_Index	Hilfeindex
help_Key	Hilfe zu einem Stichwort im Parameter Data
help_MultiKey	Hilfe zu einem Stichwort in einer alternativen Tabelle
help_Quit	Ende von Hilfe
help_SetIndex	Hilfeindex vom Parameter Data der Datei im Parameter HelpFile

hs_xxx

Beschreibung

Schraffurstile für den Pinsel. Sie werden in der Funktion **CreateHatchBrush** verwendet.

Konstante	Bedeutung
hs_BDiagonal	\\\\\
hs_Cross	+++++
hs_DiagCross	xxxxx
hs_FDiagonal	/////
hs_Horizontal	-----
hs_Vertical	|||||

htxxx

Beschreibung

Position des Cursors in Relation zum Fenster, die in **wm_MouseActivate**, **wm_SetCursor** und **wm_NCHitTest** gemeldet wird.

Konstante	Bedeutung
htBottom	Unterer Fensterrand
htBottomLeft	Untere linke Fensterecke
htBottomRight	Untere rechte Fensterecke
htCaption	Titelbereich
htClient	Innerhalb des Client-Bereichs
htError	Wie htNowhere, plus einem Warnton.
htGrowBox	Größenveränderungsfeld
htHScroll	Horizontale Bildlaufleiste
htLeft	Linker Fensterrand
htMenu	Menübereich
htNowhere	Hintergrund oder Trennlinie zwischen einzelnen Fenstern
htReduce	Minimierfeld
htRight	Rechter Fensterrand
htSize	Wie htGrowBox.
htSysMenu	System-Menüfeld
htTop	Oberer Fensterrand
htTopLeft	Oberer linker Fensterrand
htTopRight	Oberer rechter Fensterrand
htTransparent	Verdecktes Fenster
htVScroll	Vertikale Bildlaufleiste
htZoom	Zoomfeld

idxxx

Beschreibung

Rückgabewerte für **MessageBox**,, die das jeweilige Ergebnis anzeigen.

Konstante	Bedeutung
idAbort	Aktionsschalter Abort wurde gedrückt
idCancel	Aktionsschalter Cancel wurde gedrückt
idIgnore	Aktionsschalter Ignore wurde gedrückt
idNo	Kein Schalter wurde gedrückt
idOk	Aktionsschalter OK wurde gedrückt
idRetry	Aktionsschalter Retry wurde gedrückt
idYes	Aktionsschalter Yes wurde gedrückt

idc_xxx

Beschreibung

IDs der einzelnen Windows-Zeiger, die mit **LoadCursor** geladen werden.

Konstante	Bedeutung
idc_Arrow	Pfeil.
idc_Cross	Fadenkreuz.
idc_IBeam	I für Text.
idc_Icon	Leeres Symbol (konzentrische Rechtecke).
idc_Size	Pfeile in alle Himmelrichtungen
idc_SizeNESW	Zwei Pfeilspitzen, Nordost und Südwest
idc_SizeNS	Zwei Pfeilspitzen, Nord und Süd
idc_SizeNWSE	Zwei Pfeilspitzen, Nordwest und Südost
idc_SizeWE	Zwei Pfeilspitzen, West unf Ost
idc_UpArrow	Vertikaler Pfeil nach oben
idc_Wait	Sanduhr

idi_xxx

Beschreibung

IDs für Windows-Symbole, die mit **LoadIcon** geladen werden.

Konstaante	Bedeutung
idi_Application	Vorgabe
idi_Asterisk	'i' für Informationen
idi_Exclamation	'!' für Warnungen
idi_Hand	Stopzeichen für ernste Warnungen
idi_Question	'?' für Eingabeaufforderungen

ie_xxx

Beschreibung

Negativwerte die von **OpenComm** zurückgegeben werden, falls ein Fehler beim Öffnen eines Kommunikationsgerätes aufgetreten ist.

Konstante	Bedeutung
ie_BadID	Ungültige ID.
ie_BaudRate	Ungültige Baudrate.
ie_ByteSize	Ungültige Bytegröße
ie_Default	Fehler in Vorgabe-Parametern.
ie_Hardware	Hardware nicht vorhanden
ie_Memory	Kann Warteschlange nicht bereitstellen
ie_NOpen	Gerät nicht offen
ie_Open	Gerät bereits offen

lb_xxx

Beschreibung

Diese Werte werden von Windows-Listenbotschaften, wie **lb_AddString**,als Rückgabewert geliefert. Ein negativer Wert bedeutet einen Fehler.

Konstante	Bedeutung
lb_Err	Ein Fehler ist aufgetreten; Aktion wurde nicht ausgeführt
lb_ErrSpace	Zu wenig Platz im Listenfenster für die Aktion
lb_Okay	Kein Fehler

lbn_xxx

Beschreibung

Diese Nachrichtencodes werden in **wm_Command** Botschaften weitergegeben, die von Listenfenster-Steuerelementen erzeugt werden, wenn eine Aktion erfolgte.

Konstante	Bedeutung
lbn_DblClk	Wird übergeben, wenn der Anwender eine Zeichenkette durch einen Doppelklick auswählt.
lbn_ErrSpace	Wird übergeben, wenn im System kein Speicher mehr frei ist.
lbn_KillFocus	Zeigt an, daß die Liste den Fokus verloren hat.
lbn_SelChange	Wird übergeben, wenn sich die Auswahl geändert hat
lbn_SetFocus	Zeigt an, daß die Liste den Fokus erhalten hat.

lbs_xxx

Beschreibung

Diese Konstanten bestimmen den Stil von Listenfenstern, die mit **CreateWindow** und **CreateWindowEx** erstellt werden.

Konstante	Bedeutung
lbs_ExtendedSel	Der Benutzer kann mehrere Einträge auswählen, indem er die SHIFT-Taste und die Maus benutzt, oder spezielle Tastenkombinationen verwendet.
lbs_HasStrings	Dieser Stil wird in Verbindung mit lbs_OwnerDrawFixed oder lbs_OwnerDrawVariable benutzt. Er steht für eine anwendungsgezeichnete Liste, die Einträge enthält, die aus Strings bestehen. Die Liste der Strings wird vom System verwaltet und können durch die Botschaft **lb_GetText** ermittelt werden.
lbs_MultiColumn	Gibt eine mehrspaltige Liste an, die horizontal gerollt wird. Die Botschaft **lb_SetColumnWidth** legt die Breite der Spalten fest.
lbs_MultipleSel	Die Stringauswahl wechselt jedesmal ihren Zustand, wenn der Benutzer den String einfach oder doppelt anklickt. Jede Anzahl von Strings kann ausgewählt werden.
lbs_NoIntegralHeight	Die Größe des Listenfensters wird durch die Applikation während desErstellens der gleichen festgelegt und nicht von Windows beeinflußt. Normalerweise legt Windows die Größe einer Liste so fest, daß die Liste keine Teileinträge anzeigt.
lbs_NoRedraw	Die Anzeige der Liste wird nicht aktualisiert, wenn Änderungen gemacht werden. Dieser Stil kann jederzeit geändert werden, indem man eine Botschaft **wm_SetRedraw** sendet.
lbs_Notify	Das übergeordnete Fenster empfängt immer dann eine Eingabebotschaft, wenn der Benutzer einen String einfach oder doppelt anklickt.
lbs_OwnerDrawFixed	Der Besitzer der Liste ist für das Zeichnen ihres Inhalts verantwortlich; die Einträge in der Liste haben alle die gleiche Höhe.
lbs_OwnerDrawVariable	Der Besitzer der Liste ist für das Zeichnen ihres Inhalts verantwortlich; die Einträge in der Liste haben eine variable Höhe.
lbs_Sort	Strings in der Liste werden alphabetisch sortiert.
lbs_Standard	Strings in der Liste werden alphabetisch sortiert und das übergeordnete Fenster empfängt immer dann eine Eingabebotschaft, wenn der Benutzer einen String einfach oder doppelt anklickt. Die Liste ist umrandet.
lbs_UseTabStops	Erlaubt einer Liste Tabulatorzeichen zu erkennen und zu erweitern, wenn sie ihre Strings zeichnet. Die voreingestellten Tabulatorpositionen sind 32 Dialogeinheiten. (Eine Dialogeinheit ist eine horizontale oder vertikale Distanz. Eine horizontale Dialogeinheit ist gleich 1/4 der zugehörigen Basiseinheit. Die Basiseinheiten werden ent- sprechend der Höhe und Breite der momentanen Systemschrift berechnet. Die Funktion

	GetDialogBaseUnits liefert die gegenwärtigen Dialogbasiseinheiten in Pixel zurück.)
lbs_WantKeyboardInput	Der Besitzer der Liste erhält jedesmal **wm_VKeyToItem**- oder **wm_CharToItem**-Botschaften, wenn der Benutzer eine Taste drückt, während die Liste den Fokus besitzt. Dies erlaubt einer Anwendung, eine spezielle Bearbeitung der Tastatureingabe vorzunehmen.

lc_xxx

Beschreibung

Diese Konstanten stellen die Liniendarstellungs-Möglichkeiten eines Geräts dar.

Konstante	Bedeutung
lc_None	Keine Linien
lc_Interiors	Kann Innenlinien
lc_Marker	Kann Markierzeichen
lc_PolyLine	Kann mehrfache Linien
lc_PolyMarker	Kann mehrfache Markierzeichen
lc_Styled	Kann verschiedene Linienstile.
lc_Wide	Kann breite Linien.
lc_WideStyled	Kann verschiedene breite Linienstile.

lf_FaceSize

Beschreibung

Die Konstante lf_FaceSize enthält die Anzahl Bytes mit denen der Name der Schrift im Feld lfFaceName des Records **TLogFont** gespeichert wird. Der Standardwert ist 32.

lmem_xxx

Beschreibung

Charakteristika des lokalen Speicherblocks, der mit **LocalAlloc** und **LocalReAlloc** erstellt wird bzw. erstellt wurde.

lmem_Discardable, lmem_Discarded, lmem_LockCount werden in der Funktion **LocalFlags** verwendet.

Konstante	Bedeutung
lmem_Discardable	Speicherblock kann entfernt werden, muß mit lmem_Moveable verwendet werden.
lmem_Discarded	Speicherblock wurde entfernt. (LocalFlags)
lmem_LockCount	Kombiniert mit dem niederwertigen Byte des Rückgabewerts von LocalFlags wird der Referenzzähler des Speicherblocks zurückgegeben. (nur LocalFlags)
lmem_Fixed	Speicherblock an fester Speicheradresse
lmem_Modify	Ändert das Flag lmem_Discardable
lmem_Moveable	Speicherblock im Speicher verschiebbar
lmem_NoCompact	Beim Zuweisen des Speicherblocks werden keine anderen verschoben oder entfernt
lmem_NoDiscard	Beim Zuweisen des Speicherblocks werden keine anderen entfernt
lmem_ZeroInit	Initialisiert den Inhalt des Speicherblocks auf 0.

LPTx-Konstante

LPTx ist eine Bitmaske für das Feld ID des Records **TDCB**. Ist die Bitmaske gesetzt, ist das Gerät ein LPT (eine parallele Schnittstelle).

ma_xxx

Beschreibung

Diese Werte zeigen, wenn sie von einer **wm_MouseActivate**-Botschaften zurückgegeben werden, ob das Fenster aktiviert und ob das Mausereignis eliminiert werden soll.

Konstante	Bedeutung
ma_Activate	Fenster aktivieren.
ma_ActivateAndEat	Fenster aktivieren; Mausereignis eliminieren
ma_NoActivate	Fenster nicht aktivieren.

mb_xxx

Beschreibung

Charakteristika des Meldungsfelds, das mit **MessageBox** erstellt wurde. Die einzelnen Flags können kombiniert werden.

Konstante	Bedeutung
mb_AbortRetryIgnore	Meldungsfenster enthält drei Schalter: Abort, Retry und Ignore.
mb_ApplModal	Erzeugt modales Meldungsfenster (Vorgabe)
mb_DefButton1	Erster Schalter ist die Voreinstellung (Vorgabe)
mb_DefButton2	Zweiter Schalter ist die Voreinstellung.
mb_DefButton3	Dritter Schalter ist die Voreinstellung.
mb_IconAsterisk	Wie mb_IconInformation.
mb_IconExclamation	Mit '!' Symbol.
mb_IconHand	Wie mb_IconStop.
mb_IconInformation	Mit 'i' Symbol.
mb_IconQuestion	Mit '?' Symbol.
mb_IconStop	Mit STOP Symbol.
mb_Ok	Mit Aktionsschalter OK
mb_OkCancel	Mit Aktionsschaltern OK und Cancel
mb_RetryCancel	Mit Aktionsschaltern Retry und Cancel
mb_SystemModal	Alle Anwendungen werden unterbrochen, bis der Anwender dem Meldungsfenster antwortet.
mb_TaskModal	Gleichbedeutend mit mb_ApplModal mit der Ausnahme, daß alle Hauptfenster, die zum aktuellen Task gehören, gesperrt werden, wenn der Parameter WndOwner Null ist.
mb_YesNo	Mit Aktionsschaltern Yes und No
mb_YesNoCancel	Mit Aktionsschaltern Yes, No und Cancel

Bitmasken für Gruppen von mb_xxx-Konstanten:

Konstante	maskierte Konstanten
mb_DefMask	mb_DefButton1, mb_DefButton2, mb_DefButton3
mb_IconMask	mb_IconAsterisk, mb_IconExclamation, mb_IconHand, mb_IconInformation, mb_IconQuestion, mb_IconStop
mb_ModeMask	mb_ApplModal, mb_SystemModal, mb_TaskModal
mb_TypeMask	mb_AbortRetryIgnore, mb_Ok, mb_OkCancel, mb_RetryCancel, mb_YesNo, mb_YesNoCancel

meta_xxx

Beschreibung

Diese Konstanten entsprechen bestimmten GDI-Funktionen.

Der Indexwert dieser Funktionen und entsprechender Konstanten kann in einer Windows-Metadatei gespeichert werden, die eine Liste von GDI-Befehlen enthält, welche von einem Programm für grafische Ausgaben benutzt wird.

Diese Konstanten dienen zum Lesen einer solchen Metadatei.

Konstante

meta_AnimatePalette
meta_Arc
meta_BitBlt
meta_Chord
meta_CreateBitmap
meta_CreateBitmapIndirect
meta_CreateBrush
meta_CreateBrushIndirect
meta_CreateFontIndirect
meta_CreatePalette
meta_CreatePatternBrush
meta_CreatePenIndirect
meta_CreateRegion
meta_DeleteObject
meta_DIBBitBlt
meta_DIBCreatePatternBrush
meta_DIBStretchBlt
meta_DrawText
meta_Ellipse
meta_Escape
meta_ExcludeClipRect
meta_ExtTextOut
meta_FillRegion
meta_FloodFill
meta_FrameRegion
meta_IntersectClipRect
meta_InvertRegion
meta_LineTo
meta_MoveTo
meta_OffsetClipRgn
meta_OffsetViewportOrg
meta_OffsetWindowOrg
meta_PaintRegion
meta_PatBlt
meta_Pie
meta_Polygon
meta_PolyLine
meta_PolyPolygon
meta_RealizePalette
meta_Rectangle
meta_ResizePalette

meta_RestoreDC
meta_RoundRect
meta_SaveDC
meta_ScaleViewportExt
meta_ScaleWindowExt
meta_SelectClipRegion
meta_SelectObject
meta_SelectPalette
meta_SetBKColor
meta_SetBKMode
meta_SetDIBToDev
meta_SetMapMode
meta_SetMapperFlags
meta_SetPalEntries
meta_SetPixel
meta_SetPolyFillMode
meta_SetRelAbs
meta_SetROP2
meta_SetStretchBltMode
meta_SetTextAlign
meta_SetTextCharExtra
meta_SetTextColor
meta_SetTextJustification
meta_SetViewportExt
meta_SetViewportOrg
meta_SetWindowExt
meta_SetWindowOrg
meta_StretchBlt
meta_TextOut

mf_xxx

Folgende Konstanten werden als Flags in vielen Menüfunktionen und der Botschaft **wm_MenuSelect** verwendet.

Konstante	Bedeutung
mf_Bitmap	Menüelement ist ein Bitmap, kein String
mf_ByCommand	Menüelement wird durch ID bezeichnet
mf_ByPosition	Menüelement wird duch Position bezeichnet, das erste ist auf 0
mf_Checked	Erzeugt Markierung neben dem Menüelement
mf_Disabled	Deaktiviere Menüelement.
mf_Enabled	Aktiviere Menüelement.
mf_Grayed	Deaktiviere Menüelement und zeichne es in Grau.
mf_Help	Hilfemenüelement
mf_Hilite	Hebt Menüelement hervor.
mf_MenuBarBreak	Element des Popup-Menüs in neuer Spalte, durch Linie getrennt
mf_MenuBreak	Element des Popup-Menüs in neuer Spalte bzw. Element einer Menüzeile in eine neue Zeile
mf_MouseSelect	Menüelement wurde mit Maus ausgewählt
mf_OwnerDraw	Menüelement ist selbst definiert
mf_Popup	Menüelement hat Unterpunkte
mf_Separator	Horizontale Trennlinie im Menü
mf_String	Neues Menüelement ist ein String
mf_SysMenu	Element ist im Systemmenü.
mf_Unchecked	Entfernt Markierung, falls vorhanden
mf_Unhilite	Entfernt Hervorhebung

mk_xxx

Beschreibung

Diese Masken werden in Mausbotschaften kombiniert und weitergegeben, um den Status bestimmter Tasten und Maustasten zu bestimmen.

Konstante	Bedeutung
mk_Control	Ctrl gedrückt
mk_LButton	linke Maustaste gedrückt
mk_MButton	mittlere Maustaste gedrückt
mk_RButton	rechte Maustaste gedrückt
mk_Shift	Shift gedrückt

mm_xxx

Beschreibung

Diese Konstanten bestimmen den Abbildungsmodus des Gerätekontextes, mit dem logische Einheiten in Geräteeinheiten verwandelt werden.

Sie werden in den Funktionen **GetMapMode** und **SetMapMode** verwendet.

Konstante	Bedeutung
mm_Anisotropic	Einheiten und Achsen beliebig
mm_HiEnglish	Ein logische Einheit entspricht 0.001 Zoll. Die positive X-Achse ist rechts, die positive Y-Achse oben.
mm_HiMetric	Ein logische Einheit entspricht 0.01 mm. Die positive X-Achse ist rechts, die positive Y-Achse oben.
mm_Isotropic	Eine logische X-Achseneinheit entspricht einer logischen Y-Achseneinheit.
mm_LoEnglish	Ein logische Einheit entspricht 0.01 Zoll. Die positive X-Achse ist rechts, die positive Y-Achse oben.
mm_LoMetric	Ein logische Einheit entspricht 0.1 millimeters. Die positive X-Achse ist rechts, die positive Y-Achse oben.
mm_Text	Ein logische Einheit entspricht einem Pixel. Die positive X-Achse ist rechts, die positive Y-Achse unten.
mm_Twips	Ein logische Einheit entspricht 1/1440 eines Zoll. Die positive X-Achse ist rechts, die positive Y-Achse oben.

msgf_xxx (wm_EnterIdle)

Beschreibung

Diese Konstanten werden in der Botschaft **wm_EnterIdle** weitergegeben, wobei angegeben wird, ob die Botschaft aus einem Dialogfenster oder Menü stammt.

Konstante	Bedeutung
msgf_DialogBox	Das Sytem ist untätig, weil ein Dialogfenster angezeigt ist.
msgf_Menu	Das Sytem ist untätig, weil ein Menü angezeigt ist.

msgf_xxx

Beschreibung

Diese Konstanten werden im Parameter Code der Botschaftsfilterfunktionen wh_MsgFilter und wh_SysMsgFilter zur Angabe des Botschaftstyps benutzt.

Konstante	wh_MsgFilter	wh_SysMsgFilter
msgf_DialogBox	Meldungs- und Dialogfenster	nur Dialogfenster
msgf_Menu	Tastatur und Maus	Tastatur und Maus
msgf_MessageBox		Nur Meldungsfenster

obj_xxx

Beschreibung

Diese zwei Konstanten geben den Typ des GDI-Objekts an, (Stift oder Pinsel) in einem Aufruf der Funktion **EnumObjects**. Sie sollten im Parameter ObjectType an EnumObjects übergeben werden.

Konstante	Bedeutung
obj_Pen	Stifte
obj_Brush	Pinsel

obm_xxx

Beschreibung

Diese Konstanten bestimmen Bitmaps für Windows-Zwecke. Der Programmierer kann sie mit der Funktion **LoadBitmap** verwenden. Die Konstanten, die mit obm_Old beginnen, beziehen sich auf Windows Version vor der Version 3.0.

obm_BtnCorners
obm_BtSize
obm_Check
obm_CheckBoxes
obm_Close
obm_Combo
obm_DnArrow
obm_DnArrowD
obm_LfArrow
obm_LfArrowD
obm_MnArrow
obm_Old_Close
obm_Old_DnArrow
obm_Old_LfArrow
obm_Old_Reduce
obm_Old_Restore
obm_Old_RgArrow
obm_Old_UpArrow
obm_Old_Zoom
obm_Reduce
obm_Reduced
obm_Restore
obm_Restored
obm_RgArrow
obm_RgArrowD
obm_Size
obm_UpArrow
obm_UpArrowD
obm_Zoom
obm_ZoomD

oda_xxx

Beschreibung

Diese Konstanten definieren eine Aktion für ein selbstdefiniertes Element, wenn sie im Feld itemAction des Record **TDrawItemStruct** verwendet werden. Der Wert dieses Feldes kann eine Kombination dieser Konstanten sein.

Konstante	Bedeutung
oda_DrawEntire	Gesamtes Element muß gezeichnet werden
oda_Focus	Fokus erhalten oder verloren
oda_Select	Auswahlstatus verändert

ods_xxx

Beschreibung

Diese Konstanten definieren den Status für ein selbstdefiniertes Element, wenn sie im Feld itemState des Record **TDrawItemStruct** verwendet werden. Der Wert dieses Feld kann eine Kombination dieser Konstanten sein.

Konstante	Bedeutung
ods_Checked	Prüfe Menüelement
ods_Disabled	Deaktiviere Element
ods_Focus	Fokussiere Element für Eingabe
ods_Grayed	Zeichne Element grau
ods_Selected	Selektiere Element

odt_xxx

Beschreibung

Diese Konstanten bestimmen einen besonderen Typ selbstdefinierter Elemente. Sie werden im Feld CtlType der Records **TCompareItemStruct**, **TDrawItemStruct** und **TMeasureItemStruct** verwendet.

Konstante	Bedeutung
odt_Menu	selbstdefiniertes Menü
odt_ListBox	selbstdefiniertes Listenfenster
odt_ComboBox	selbstdefiniertes Kombifenster
odt_Button	selbstdefinierter Aktionsschalter

of_xxx

Beschreibung

Diese Konstanten geben, wenn sie kombiniert und in Aufrufen an Funktion **OpenFile** oder **_lopen** weitergegeben werden, den Modus zum Öffnen der entsprechende Datei, an.

Konstante	Bedeutung
of_Cancel	Fügt den Aktionsschalter Cancel zum Dialogfenster, welches erzeugt wird, wenn of_Prompt angegeben wurde, hinzu. Wird Cancel gedrückt, gibt **OpenFile** die Fehlermeldung Datei nicht gefunden zurück. (nur OpenFile.)
of_Create	Öffnet neue Datei oder löscht existierende (nur OpenFile)
of_Delete	Löscht Datei (nur OpenFile)
of_Exist	Prüft ob angegebene Datei existiert (nur OpenFile)
of_Parse	Füllt nur den Record OFStruct im Parameter ReOpenBuff (nur OpenFile)
of_Prompt	Wenn die Datei nicht gefunden werden kann , erscheint ein Dialogfenster mit der Meldung, daß die Datei nicht gefunden werden kann, und fordert den Benutzer auf eine Diskette in Laufwerk A zu legen.
of_Read	Öffnet die Datei nur zum Lesen
of_ReadWrite	Öffnet die Datei zum Lesen und Schreiben
of_ReOpen	Öffnet Datei mit dem Record OFStruct des Parameters ReOpenBuff (nur OpenFile)
of_Share_Compat	Öffnet Datei, erlaubt aber anderen Prozessen, die Datei ebenfalls zu öffnen (Comaptible Mode)
of_Share_Deny_None	Öffnet Datei, erlaubt aber anderen Prozessen, die Datei zu lesen oder zu beschreiben.
of_Share_Deny_Read	Öffnet Datei, hindert aber andere Prozesse, die Datei zu lesen.
of_Share_Deny_Write	Öffnet Datei, hindert aber andere Prozesse, in die Datei zu schreiben.
of_Share_Exclusive	Öffnet Datei, hindert aber andere Prozesse, die Datei zu lesen oder zu beschreiben.
of_Verify	Prüft. ob Datum und Zeit der Datei mit dem Zeitpunkt des letzten Öffnens der Datei übereinstimmen (nur OpenFile)
of_Write	Öffnet eine Datei nur zum Schreiben.

out_xxx

Beschreibung

Diese Konstanten bestimmen die Ausgabepräzision von Schriften, die mit der Funktion **CreateFont** erstellt wurden.

Out_Default_Precis
Out_String_Precis
Out_Character_Precis
Out_Stroke_Precis

pc_xxx Paletteninformationen

Beschreibung

Diese Flags enthalten Paletteninformationen.

Konstante	Bedeutung
pc_Explicit	Bestimmt, daß das niederwertige Word des logischen Paletteneintrags einen Hardware-Palettenindex beschreibt.
pc_NoCollapse	Bestimmt, daß die Farbe in einem ungenutzten Eintrag in der Systempalette abgelegt wird, anstelle der Anpassung an einem existierenden Farbeintrag in der Systempalette.
pc_Reserved	Bestimmt, daß der Paletteneintrag für Palettenanimation genutzt wird.

Diese Konstanten werden im Feld peFlags des Records **TPaletteEntry** verwendet und in Aufrufen an folgende Funktionen weitergegeben.

AnimatePalette
GetPaletteEntries
GetSystemPaletteEntries
SetPaletteEntries

pc_xxx

Beschreibung

Diese Konstanten stellen die Vieleck-Möglichkeiten eines Geräts dar.

Konstante	Bedeutung
pc_Interiors	Kann Innenflächen zeichnen
pc_None	Keine Polygone
pc_Polygon	Kann Polygone zeichnen.
pc_Rectangle	Kann Rechtecke zeichnen.
pc_ScanLine	Kann Scan-Linien zeichnen.
pc_Styled	Kann verschiedene Ränderarten zeichnen.
pc_Trapezoid	Kann Trapeze zeichnen.
pc_Wide	Kann breite Ränder zeichnen
pc_WideStyled	Kann verschiene breite Ränderarten zeichnen.
pc_WindPolygon	Kann verzogene Polygone zeichnen.

pm_xxx

Beschreibung

Diese Optionen bestimmen die Bearbeitung von Botschaften mit der Funktion **PeekMessage**.

Konstante	Bedeutung
pm_NoRemove	Nach der Bearbeitung bleiben Botschaften in der Warteschlange.
pm_NoYield	Kein anderer Task darf den aktuellen unterbrechen.
pm_Remove	Nach der Bearbeitung werden Botschaften aus der Warteschlange entfernt.

PolyFill-Modus

Beschreibung

Diese Konstanten bestimmen, wie Vielecke mit mehreren Flächen gefüllt werden.

Konstante	Bedeutung
Alternate	Füllt alternierende Flächen.
Winding	Füllt jede Fläche.

Diese Konstanten werden in Aufrufen folgender Funktionen benutzt:

CreatePolygonRgn
CreatePolyPolygonRgn
GetPolyFillMode
SetPolyFillMode

pr_JobStatus

pr_JobStatus ist der in wParam übergebene Wert an die Botschaft **wm_SpoolerStatus**.

Drucker-Escapesequenzen

Beschreibung

Diese Konstanten werden in der Funktion **Escape** verwendet, um Anwendungen den direkten Zugriff auf diejenigen Gerätefunktionen zu ermöglichen, die von GDI nicht unterstützt werden.

Konstante	Bedeutung
AbortDoc	Bricht aktuellen Druckauftrag ab
BandInfo	Kopiert Informationen über die spezifischen Fähigkeiten eines Geräts in einen Record, der an die Funktion Escape übergeben wird. Diese Fähigkeiten beziehen sich auf die Möglichkeit, die Ausgabe in einzelne Abschnitte aufzuteilen, die vor der Ausgabe in einer Metadatei gespeichert werden.
Begin_Path	Öffnet einen Pfad, der ein Polygon oder einen Linienzug aus verbundenen Punkten darstellt. Pfade werden verwendet, um die Befehlsübergabe von Befehlen an PostScript-Drucker zu erleichtern.
Clip_To_Path	Definiert einen Clipping-Bereich auf Grundlage des aktuellen Pfads.
DeviceData	Wie PassThrough.
DraftMode	Schaltet den Draft-Modus (EDV-Qualität) an oder aus.
DrawPatternRect	Zeichnet ein gemustertes, ein schwarzes oder ein Graustufenrechteck auf einem PCL-kompatiblen Hewlett-Packard-Drucker.
EnableDuplex	Schaltet die Druckerfunktion zum beidseitigen Druck ein.
EnablePairKerning	Aktiviert oder deaktiviert die Kerning-Funktion des Druckers.
EnableRelativeWidths	Aktiviert oder deaktiviert die relative Zeichenbreite. Ist diese Option aktiviert, müssen Sie auf die Breitentabellen der Schrift zugreifen und die Breite der Strings berechnen.
End_Path	Beendet einen Pfad. Siehe Begin_Path.
EndDoc	Beenden einen Druckauftrag, der durch StartDoc gestartet wurde.
EnumPaperBins	Ruft Informationen über die Papierschächte des Druckers ab. Sie sollten besser die Konstante GetSetPaperBins verwenden.
EnumPaperMetrics	Ruft die Anzahl unterstützter Papiertypen oder die Größen der druckbaren Rechtecke ab. Sie sollten besser die Funktion ExtDeviceMode verwenden.
EPSPrinting	Unterdrückt die Steuerungsabschnitte des PostScript-Headers und unterbindet dadurch sämtliche GDI-Aufrufe.
Ext_Device_Caps	Ruft Informationen über gerätespezifische Druckerfähigkeiten ab, die über die Rückgabe der Funktion **GetDeviceCaps** hinausgehen.
FlushOutput	Entleert den Gerätepuffer.
GDIExtTextOut	Entspricht der Funktion **ExtTextOut**.
GDIStretchBlt	Entspricht der Funktion **StretchBlt**.
GetColorTable	Ruft einen RGB-Farbtabelleneintrag ab.
GetExtendedTextMetrics	Ruft die erweiterten Textabmessungen der ausgewählten Schrift ab.
GetExtentTable	Ruft die Ausdehnung (Breite) eines Zeichenbereichs im

	aktuellen Zeichensatz der Schrift ab.
GetPairKernTable	Ruft die Zeichenpaar-Kerning-Tabelle ab.
GetPhysPageSize	Ruft die physikalische Papiergröße ab.
GetPrintingOffset	Ruft den Offset der oberen linken Position auf einer Seite ab, an der das Drucken oder Zeichnen begonnen werden soll.
GetScalingFactor	Ruft den Skalierungsfaktor des Druckers in x- und y-Richtung ab.
GetSetPaperBins	Ruft die Anzahl der Papierschächte des Druckers ab und setzt den aktuellen Schacht.
GetSetPaperMetrics	Ruft die Papierabmessungsinformationen des Druckers ab und stellt sie neu ein. Sie sollten besser die Funktion ExtDeviceMode verwenden.
GetSetPrintOrient	Ruft die aktuelle Papierausrichtung ab oder stellt sie neu ein. Sie sollten besser die Funktion ExtDeviceMode vewenden.
GetTechnolgy	Ruft einen String ab, der die grundlegende Arbeitsweise des Druckers abezeichnet, z.B. PostScript.
GetTrackKernTable	Ruft die Track-Kerning-Tabelle der aktuell gewählten Schrift ab.
GetVectorBrushSize	Ruft die Breite des Plotterstifts in Geräteeinheiten ab, der zum Füllen geschlossener Formen verwendet wird.
GetVectorPenSize	Ruft die Breite des Plotterstifts in Geräteeinheiten ab, der für Schraffurmuster verwendet wird.
MFComment	Fügt einen Kommentar zu einer Metadatei hinzu.
NewFrame	Weist den Drucker an, auf einer neuen Seite fortzufahren.
NextBand	Teilt dem Gerätetreiber mit, daß die Ausgabe eines Abschnitt in eine Metadatei abgeschlossen wurde.
PassThrough	Erlaubt einer Anwendung die unmittelbare Übergabe von Daten an den Drucker.
QueryEscSupport	Ermittelt, ob der Gerätetreiber einen bestimmten Escape-Code unterstützt.
Restore_CTM	Stellt die vorher gesicherte, aktuelle Transformationsmatrix wieder her.
Save_Ctm	Sichert die aktuelle Transformationsmatrix.
SelectPaperSource	Wird von GetSetPaperBins ersetzt und wurde nur aus Gründen der Abwärtskompatibilität übernommen.
Set_Arc_Direction	Bestimmt die Richtung, in der Kreisbögen durch die Funktion **Arc** gezeichnet werden.
Set_Background_Color	Ruft die Hintergrundfarbe eines Geräts ab oder stellt sie neu ein.
Set_Bounds	Stellt das eingrenzende Rechteck für das darzustellende Bild ein.
Set_Clip_Box	Setzt ein Clipping-Rechteck oder stellt das vorhergehende wieder her.
Set_Poly_Mode	Setzt den Poly-Modus ein, der festlegt, wie Aufrufe der Funktionen **Polygon** und **PolyLine** interpretiert werden.
Set_Screen_Angle	Setzt den aktuellen Neigungswinkel der Rasterung und simuliert so die Neigung eines photographischen Farbfilters, um die Farbtrennung für eine Grundfarbe zu ermöglichen.
Set_Spread	Setzt den Umfang, um den nicht-weiße Grundelemente erweitert werden, um Fehler bei der Reproduktion auszugleichen.
SetAbortProc	Setzt die Abbruchfunktion für den aktuellen Druckauftrag.

SetAllJustValues	Setzt die Ausrichtungswerte für die Textausgabe.
SetColorTable	Setzt einen RGB-Farbtabelleneintrag.
SetCopyCount	Bestimmt die Anzahl der Kopien, die von jeder Seite gedruckt werden sollen.
SetKernTrack	Bestimmt, welcher Kerning-Trach verwendet wird.
SetLineJoin	Legt fest, wie schneidende Linien miteinander verbunden werden: mit abgerundeten, eckigen oder bis zur Spitze verlängerten Ecken.
SetMiterLimit	Setzt den Gehrungswinkel, ab dem Linienverbindungen nicht bis zur Spitze verlängert, sondern eckig abgeschnitten werden.
StartDoc	Teilt dem Gerätetreiber mit, daß ein neuer Druckauftrag beginnt.
Transform_CTM	Verändert die aktuelle Transformationsmatrix.

proc_xxx

Diese Konstanten werden im Parameter ProcName der Funktion **GetProcAddress** weitergegeben, wenn das Module-Handle im Module Parameter ein Gerätetreiber ist.

Konstante

proc_ExtDeviceMode
proc_DeviceCapabilities
proc_OldDeviceMode

ps_xxx

Beschreibung

Stile für Stifte, verwendet in der Funktion **CreatePen**.

Konstante	Bedeutung
ps_Solid	Durchgezogene Linie
ps_Dash	-------
ps_Dot	.....
ps_DashDot	_._._._
ps_DashDotDot	_.._.._
ps_Null	Unsichtbar
ps_InsideFrame	Innenrahmen von Polygonen und Doppellinien.

r2_xxx

Beschreibung

Diese Konstanten bestimmen Zeichenmodus und Farben des aktuellen Stifts. Sie werden als Parameter in **GetROP2** und **SetROP2** verwendet.

Konstante	Bedeutung
r2_Black	Ergebnisfarbe ist schwarz
r2_CopyPen	Ergebnisfarbe ist die Stiftfarbe
r2_MaskNotPen	Ergebnisfarbe ist eine Kombination der aktuellen Farben und der Inversfarbe des Stifts
r2_MaskPen	Ergebnisfarbe ist eine Kombination der aktuellen Farben und der Stiftfarbe
r2_MaskPenNot	Ergebnisfarbe ist invers von r2_MaskPen.
r2_MergeNotPen	Ergebnisfarbe ist eine Kombination der Inversfarbe des Stifts und der aktuellen Farben
r2_MergePenNot	Ergebnisfarbe ist eine Kombination der aktuellen Stiftfarbe, invers der Anzeigefarben
r2_Nop	Darstellung ändert sich nicht
r2_Not	Ergebnisfarbe ist invers der Anzaigefarbe
r2_NotCopyPen	Ergebnisfarbe ist invers der Stiftfarbe
r2_NotMaskPen	Ergebnisfarbe ist invers der Farbe aus r2_MaskPen
r2_NotMergePen	Ergebnisfarbe ist invers der Farbe aus r2_MergePen
r2_NotXorPen	Ergebnisfarbe ist invers der Farbe aus r2_XorPen
r2_White	Ergebnisfarbe ist weiß
r2_XorPen	Ergebnisfarbe ist eine Kombination der vorhanden Anzeigefarben und des Stifts, aber nicht beider.

rc_xxx

Beschreibung

Diese Konstanten repräsentieren Rasterfähigkeiten eines Geräts.

Konstante	Bedeuung
rc_Banding	Benötigt Frequenzunterstützung
rc_BigFont	Fähig zur Unterstützung mehr als 64 KByte Schriften
rc_BitBlt	Fähig zur Bitmap-Übertragung
rc_Bitmap64	Fähig zur Unterstützung von Bitmaps mit mehr als 64 KByte
rc_DI_Bitmap	Fähig zur Unterstützung von DIB
rc_DIBToDev	Unterstützt die Funktion **SetDIBitsToDevice**
rc_FloodFill	Unterstützt die Funktion **FloodFill**
rc_GDI20_Output	Unterstützt Windows 2.0
rc_Palette	Unterstützt eine Palette.
rc_Scaling	Braucht Unterstützung zum Skalieren
rc_StretchBlt	Unterstützt die Funktion **StretchBlt**
rc_StretchDIB	Unterstützt die Funktion **StretchDIBits**

Bereichs-Flags

Beschreibung

Diese Flags bestimmen einen grafischen Bereichstyp.

Konstante	Bedeutung
ComplexRegion	Bereich hat überlappende Ränder
Error	Kein neuer Bereich erstellt
NullRegion	Bereich ist leer
SimpleRegion	Bereich hat keine überlappenden Ränder

Diese Konstanten werden als Rückgabewerte folgender Funktionen verwendet:

CombineRgn
ExcludeClipRect
ExcludeUpdateRgn
GetClipBox
GetRgnBox
GetUpdateRgn
OffsetClipRgn
OffsetRgn
SelectClipRgn

Ressourcentyp Konstante

Beschreibung

Dieser Wert ist die Identifikationsnummer der Ressource minus 1. Fügen Sie einen neuen Typ ein, müssen Sie Difference inkrementieren, die zu Beginn mit -11 definiert ist.

rgn_xxx

Beschreibung

Diese Bereichsflags bestimmen die Methode, mit der die Funktion **CombineRgn** zwei Bereiche kombiniert.

Konstante	Bedeutung
rgn_And	Benutzt überlappende Bereiche beider Regionen (Schnittpunkt).
rgn_Copy	Erzeugt eine Kopie von Region 1 (wird durch SrcRgn1 bezeichnet).
rgn_Diff	Speichert die Bereiche von Region 1 (die durch den Parameter SrcRgn1 bezeichnet wird), die nicht Teil von Region 2 (die durch den Parameter SrcRgn2 bezeichnet wird) sind.
rgn_Or	Verbindet beide Regionen (Vereinigung).
rgn_Xor	Verbindet beide Regionen, entfernt aber überlappende Bereiche.

rt_xxx

Beschreibung

Diese Konstanten repräsentieren verfügbare Typen von Windows Ressourcen. Sie werden als Parameter in der Funktion **FindResource** verwendet.

Konstante	Bedeutung
rt_Accelerator	Accelerator-Tabelle
rt_Bitmap	Bitmap-Ressource
rt_Cursor	Cursor-Ressource
rt_Dialog	Dialogfenstermuster
rt_Font	Schrift-Ressource
rt_FontDir	Schriftverzeichnis-Ressource
rt_Icon	Symbol-Ressource
rt_Menu	Menü-Ressource
rt_RcData	Selbst definierte Ressource
rt_String	String-Ressource

s_xxx

Beschreibung

Diese Konstanten werden in Klangfunktionen verwendet und durch ihre Funktion kategorisiert.

Fehlerwerte

Negative Fehlerwerte werden von den Klangfunktionen zurückgegeben.

Konstante	Bedeutung
s_SerBDNT	Ungültige Note.
s_SerDCC	Ungültiger Notenzähler
s_SerDDR	Ungültige Dauer
s_SerDFQ	Ungültige Frequenz
s_SerDLN	Ungültige Notenlänge
s_SerDMD	Ungültiger Modus
s_SerDPT	Ungültige Höhe
s_SerDSH	Ungültige Form
s_SerDSR	Ungültige Quelle
s_SerDST	Ungültiger Status
s_SerDTP	Ungültiges Tempo.
s_SerDVL	Ungültiges Volumen
s_SerDVNA	Gerät nicht verfügber
s_SerMACT	Musik aktiv
s_SerOFM	Kein Speicher mehr
s_SerQFUL	Warteschlange voll

SetSoundNoise Funktions-Konstanten

Diese Konstanten geben eine Klangquelle an, wobei N eine Zielfrequenz ableitet.

Konstante	Bedeutung
s_Period512	Quellfrequenz ist N/512. Hoher Ton
s_Period1024	Quellfrequenz ist N/1024.
s_Period2048	Quellfrequenz ist N/2048. Tiefer Ton
s_PeriodVoice	Quellfrequenz ist Voice-Kanal 3.
s_White512	Quellfrequenz ist N/512. weniger heller Ton
s_White1024	Quellfrequenz ist N/1024.
s_White2048	Quellfrequenz ist N/2048. Tiefer Ton
s_WhiteVoice	Quellfrequenz ist Voice-Kanal 3.

SetVoiceAccent Funktions-Konstanten

Diese Konstanten geben den Akzent der Noten an.

Konstante	Bedeutung
s_Legato	Der Ton wird für die ganze Dauer gehalten und mit dem Beginn des nächsten Tons ausgeblendet.
s_Normal	Der Ton wird für die ganze Dauer gehalten und bricht ab, bevor der nächste Ton beginnt.
s_Staccato	Der Ton wird nur für einen Teil der Dauer gehalten und weist einen deutlichen Abstand zwischen ihm und dem nächsten Ton.

WaitSoundState Funktions-Konstanten

Diese Konstanten geben den Status der Stimmschleife an.

Konstante	Bedeutung
s_AllThreshold	Alle Stimmenschleifen haben den Schwellenwert erreicht.
s_QueueEmpty	Alle Stimmenschleifen sind leer; Klangtreiber nicht aktiv.
s_Threshold	Eine Stimmenschleife hat den Schwellenwert erreicht und gibt die Stimme zurück.

sb_xxx (Ereigniskonstaten)

Beschreibung

Diese Konstanten geben Bildlaufleisten-Ereignisse an.

Konstante	Bedeutung
sb_Bottom	Bildlaufleiste wurde nach unten oder rechts verschoben.
sb_EndScroll	Bildlaufleiste wurde zum Ende verschoben.
sb_LineDown	Bildlaufleiste wurde eine Zeile nach unten verschoben.
sb_LineUp	Bildlaufleiste wurde eine Zeile nach oben verschoben.
sb_PageDown	Bildlaufleiste wurde eine Seite nach unten verschoben.
sb_PageUp	Bildlaufleiste wurde eine Seite nach oben verschoben.
sb_ThumbPosition	Der Mausknopf wurde zu einer absoluten Position gezogen
sb_ThumbTrack	Der Mausknopf wurde zur angegebenen Position gezogen
sb_Top	Bildlaufleiste wurde nach oben oder links verschoben.

Diese Konstanten werden als Parameter in folgenden Botschaften übergeben:

wm_HScroll
wm_HScrollClipboard
wm_VScroll
wm_VScrollClipboard

sb_xxx

Beschreibung

Diese Konstanten bezeichnen die Bildlaufleiste, die durch die unten genannten Funktionen bzw. Prozeduren definiert werden.

Konstante	Bedeutung
sb_Both	Horizontale und vertikale Bildlaufleisten. sb_Both wird nur mit **ShowScrollBar** verwendet.
sb_Ctl	Selbständiges Bildlaufleisten-Element
sb_Horz	Horizontale Bildlaufleiste
sb_Vert	Vertikale Bildlaufleiste

Diese Konstanten werden in folgenden Funktionen bzw. Prozeduren verwendet:

GetScrollPos
GetScrollRange
SetScrollPos
SetScrollRange
ShowScrollBar

sbs_xxx

Beschreibung

Diese Konstanten Geben beim Erzeugen einer Bildlaufleiste den Stil mit den Funktionen **CreateWindow** und **CreateWindowEx** an.

Konstante	Bedeutung
sbs_BottomAlign	Wird mit dem Stil sbs_Horz benutzt. Der untere Rand der Bildlaufleiste wird am unteren Rand des Rechtecks ausgerichtet. Die Bildlaufleiste hat die voreingestellte Höhe für Systembildlaufleisten.
sbs_Horz	Bestimmt eine horizontale Bildlaufleiste. Wenn weder der Stil sbs_BottomAlign noch sbs_TopAlign gesetzt ist, hat die Bildlaufleiste die Höhe, Breite und Position, die mit der Funktion CreateWindow gesetzt sind.
sbs_LeftAlign	Wird mit dem Stil sbs_Vert benutzt. Der linke Rand der Bildlaufleiste wird nach dem linken Rand des Rechtecks ausgerichtet, das durch die mit der Funktion CreateWindow gesetzten Parameter X, Y, Width und Height festgelegt wird. Die Bildlaufleiste hat die voreingestellte Breite für Systembildlaufleisten.
sbs_RightAlign	Wird mit dem Stil sbs_Vert benutzt. Der rechte Rand der Bildlaufleiste wird nach dem rechten Rand des Rechtecks ausgerichtet, das durch die in der Funktion CreateWindow gegebenen Parameter X, Y, Width und Height festgelegt wird. Die Bildlaufleiste hat die voreingestellte Breite für Systembildlaufleisten.
sbs_SizeBox	Bestimmt ein Größenveränderungsfeld. Wenn weder der Stil sbs_SizeBoxBottomRightAlign noch der Stil sbs_SizeBoxTopLeftAlign gesetzt sind, hat das Größenveränderungsfeld die Höhe, Breite und Position, die in der Funktion CreateWindow gegeben sind.
sbs_SizeBoxBottomRightAlign	Wird mit dem Stil sbs_SizeBox benutzt. Die untere rechte Ecke des Größenveränderungsfeldes wird an der unteren rechten Ecke des Rechtecks ausgerichtet, das durch die in der Funktion CreateWindow gegebenen Parameter X, Y, Width und Height festgelegt wird. Das Größenveränderungsfeld hat die voreingestellte Größe des Systemgrößenveränderungsfeldes.
sbs_SizeBoxTopLeftAlign	Wird mit dem Stil sbs_SizeBox benutzt. Die obere linke Ecke des Größenveränderungsfeldes wird an der oberen linken Ecke des Rechtecks ausgerichtet, das durch die in der Funktion CreateWindow gegebenen Parameter X, Y, Width und Height festgelegt wird. Das Größenveränderungsfeld hat die voreingestellte Größe des Systemgrößenveränderungsfeldes.
sbs_TopAlign	Wird mit dem sbs_SizeBox benutzt. Der obere Rand der Bildlaufleiste wird an dem oberen Rand des Rechtecks ausgerichtet, das durch die in der Funktion CreateWindow gegebenen Parameter X, Y, Width und Height festgelegt wird. Die Bildlaufleiste hat die voreingestellte Höhe für Systembildlaufleisten.
sbs_Vert	Bestimmt eine vertikale Bildlaufleiste. Wenn weder der

Stil sbs_RightAlign noch der Stil sbs_LeftAlign gesetzt ist, hat die Bildlaufleiste die Höhe, Breite und Position, die in der Funktion CreateWindow gegeben sind.

sc_xxx

Beschreibung

Folgende Befehle werden in der Botschaft **wm_SysCommand** als Antwort auf eine Menüselektion oder Verkleinern/Vergrößern gegeben.

Konstante	Bedeutung
sc_Close	Fenster schließen
sc_HScroll	Horizontal verschieben
sc_KeyMenu	Menü über Tastendruck
sc_Maximize	Vergrößere das Fenster
sc_Minimize	Verkleinere das Fenster
sc_MouseMenu	Menü über Mausklick
sc_Move	Fenster bewegen
sc_NextWindow	Zum nächsten Fenster
sc_PrevWindow	Zum vorigen Fenster
sc_Restore	Von maximiert/minimiert auf die alte Größe
sc_Size	Größenveränderung des fensters
sc_TaskList	Anzeige der Task-Liste
sc_VScroll	Vertikal verschieben

show_xxx

Beschreibung

Alte Konstanten für Abwärtskompatibilität mit früheren Windows-Versionen. Verwenden Sie stattdessen lieber die sw_xxx-Konstanten.

Konstante	Wert	Bedeutung
hide_Window	0	Verdecke Fenster
show_OpenWindow	1	Vergrößere Symbol zu Fenster
show_IconWindow	2	Verkleinere Fenster zu Symbol
show_FullScreen	3	Vergrößere Fenster auf Bildschirmgröße
show_OpenNoActivate	4	Restauriere Fenster ohne es zu aktivieren

Sizexxx

Beschreibung

Diese Konstanten geben die Größenveränderung des Fensters an und werden in **wm_Size** Botschaften weitergegeben.

Konstante	Bedeutung
SizeFullScreen	Fenster wurde vergrößert
SizeIconic	Fenster wurde verkleinert
SizeNormal	Fenster wurde verändert, aber nicht vergrößert oder verkleinert
SizeZoomHide	Anderes Fenster wurde maximiert
SizeZoomShow	Anderes Fenster wurde maximierten Staus wiederhergestellt

sm_xxx

Beschreibung

Diese Codes zeigen Aspekte der Windows-Anwenderschnittstelle, deren Dimensionen der Programmierer mit der Funktion **GetSystemMetrics** erhalten kann.

Konstante	Bedeutung
sm_CXBorder	Fensterbreite nicht veränderbar
sm_CXCursor	Cursorbreite
sm_CXDlgFrame	Fensterbreite im ws_DlgFrame-Stil
sm_CXFrame	Änderbare Fensterbreite
sm_CXFullScreen	Vergrößertes Fenster, Client-Bereich-Breite
sm_CXHScroll	Horizontale Bildlaufleiste, Pfeilbreite
sm_CXHThumb	Horizontale Bildlaufleiste, Mausknopfbreite
sm_CXIcon	Symbolbreite
sm_CXMin	Minimalbreite des Fensters
sm_CXMinTrack	Minimalverschiebung des Fensters
sm_CXScreen	Bildschirmbreite
sm_CXSize	Titel-Bitmap Bretie
sm_CXVScroll	Vertikale Bildlaufleiste, Pfeilbreite
sm_CYBorder	Feste Fensterhöhe
sm_CYCaption	Titelhöhe
sm_CYCursor	Cursorhöhe
sm_CYDlgFrame	Fensterhöhe im ws_DlgFrame-Stil
sm_CYFrame	Änderbare Fensterhöhe
sm_CYFullScreen	Vergrößertes Fenster, Client-Bereich-Höhe
sm_CYHScroll	Horizontale Bildlaufleiste, Pfeilhöhe
sm_CYIcon	Symbolhöhe
sm_CYKanjiWindow	Kanji-Fensterhöhe
sm_CYMenu	Einzeilige Menühöhe
sm_CYMin	Minimalhöhe des Fensters
sm_CYMinTrack	Minimalverschiebehöhe des Fensters
sm_CYScreen	Bildschirmhöhe
sm_CYSize	Titel-Bitmap Höhe
sm_CYVScroll	Vertikale Bildlaufleiste, Pfeilhöhe
sm_CYVThumb	Vertikale Bildlaufleiste, Mausknopfbreite
sm_Debug	Gibt 0 zurück, wenn die laufende Windows-Version nicht die debuggende ist
sm_MousePresent	Gibt 0 zurück, wenn Maus nicht vorhanden
sm_SwapButton	Gibt 0 zurück, wenn Maustasten nicht vertauscht

sp_xxx

Beschreibung

Diese Konstanten werden als Fehlercodes der Funktion **Escape** zurückgegeben.

Konstante	Bedeutung
sp_AppAbort	Programm beendete Druckvorgang
sp_Error	Allgemeiner Fehler
sp_OutOfDisk	Nicht genug Plattenspeicher für Spooling.
sp_OutOfMemory	Nicht genug Speicher für Spooling.
sp_UserAbort	Anwender brach Druckvorgang ab.

ss_xxx

Beschreibung

Diese Konstanten dienen für den Stil statischer Elemente, die mit den Funktionen **CreateWindow** und **CreateWindowEx** erstellt werden.

Konstante	Bedeutung
ss_BlackFrame	Gibt ein Feld mit einem Rahmen an, der in der gleichen Farbe wie Fensterrahmen gezeichnet wird. Diese Farbe ist Schwarz im voreingestellten Windows-Farbschema.
ss_BlackRect	Gibt ein Rechteck an, das mit der gleichen Farbe gefüllt ist, mit der Fensterrahmen gezeichnet werden. Diese Farbe ist Schwarz im voreingestellten Windows-Farbschema.
ss_Center	Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text zentriert im Rechteck an. Der Text wird formattiert, bevor er angezeigt wird. Wörter, die über das Ende einer Zeile hinausreichen würden, werden automatisch auf den Anfang der nächsten zentrierten Zeile umgebrochen.
ss_GrayFrame	Gibt ein Feld mit einem Rahmen an, der in der gleichen Farbe wie der Bildschirmhintergrund (Desktop) gezeichnet wird.
ss_GrayRect	Gibt ein Rechteck an, das mit der Farbe gefüllt wird, die benutzt wird, um den Bildschirmhintergrund zu zeichnen. Dies ist Grau im voreingestellten Windows-Farbschema.
ss_Icon	Bestimmt ein Symbol, das im Dialogfenster angezeigt wird. Der gegebene Text ist der Name eines Symbols (kein Dateiname), der anderswo in der Ressourcendatei definiert ist. Die Parameter nWidth und nHeight werden ignoriert; das Symbol legt seine Größe selbst fest.
ss_Left	Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text linksbündig im Rechteck an. Der Text wird formatiert, bevor er angezeigt wird. Wörter, die über das Ende einer Zeile hinausreichen würden, werden automatisch auf den Anfang der nächsten linksbündigen Zeile umbrochen.
ss_LeftNoWordWrap	Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text linksbündig im Rechteck an. Tabulatorzeichen werden erweitert, aber Wörter werden nicht umbrochen. Text, der über das Ende einer Zeile hinausreicht, wird abgeschnitten.
ss_NoPrefix	Wenn dieser Stil nicht angegeben wird, wird Windows jedes »&« Zeichen im Text des Steuerelements als Akzeleratorpräfixzeichen interpretieren. In diesem Fall wird das »&« entfernt und das nächste Zeichen im String unterstrichen. Wenn ein statisches Steuerelement Text enthalten soll, bei dem dieses Merkmal unerwünscht ist, kann ss_NoPrefix hinzugefügt werden. Dieser statische Steuerelementstil kann in jedes der definierten Steuerelemente mit einbezogen werden.
ss_Right	Bestimmt ein einfaches Rechteck und zeigt den gegebenen Text rechtsbündig im Rechteck an. Der Text wird formatiert, bevor er angezeigt wird. Wörter, die über das Ende einer Zeile hinausreichen würden, werden automatisch auf den Anfang der nächsten rechtsbündigen Zeile umbrochen.
ss_Simple	Bestimmt ein einfaches Rechteck und zeigt eine einzelne Textzeile linksbündig im Rechteck an. Die Textzeile kann auf

	keinen Fall gekürzt oder geändert werden. (Das übergeordnete Fenster oder das Dialogfenster des Steuerelements darf die Botschaft **wm_CtlColor** nicht bearbeiten.)
ss_UserItem	Gibt einen benutzerdefinierten Eintrag an.
ss_WhiteFrame	Gibt ein Feld mit einem Rahmen an, der in der gleichen Farbe wie der Fensterhintergrund gezeichnet wird. Diese Farbe ist Weiß im voreingestellten Windows-Farbschema.
ss_WhiteRect	Gibt ein Rechteck an, das mit der Farbe gefüllt wird, die benutzt wird, um den Fensterhintergrund zu zeichnen. Diese Farbe ist Weiß im voreingestellten Windows-Farbschema.

Logische Objekte

Beschreibung

Diese Konstanten repräsentieren vordefinierte Zeichengeräte für die Funktion **GetStockObject**.

Konstante	Bedeutung
Black_Brush	Schwarzer Pinsel.
DkGray_Brush	Dunkelgrauer Pinsel.
Gray_Brush	Grauer Pinsel.
Hollow_Brush	Hohler Pinsel.
LtGray_Brush	Hellgrauer Pinsel.
Null_Brush	Kein Pinsel.
White_Brush	Weißer Pinsel.
Black_Pen	Schwarzer Stift
Null_Pen	Kein Stift
White_Pen	Weißer Stift
ANSI_Fixed_Font	ANSI feste Schriftneigung
ANSI_Var_Font	ANSI variable Schriftneigung
Device_Default_Font	Gerätabhängige Schrift
OEM_Fixed_Font	OEM-abhängige feste Schrift
System_Fixed_Font	feste Schriftneigung von alter Windows-Version.
System_Font	Windows Systemschrift mit variabler Schriftneigung
Default_Palette	Vorgabe.Farbpalette

StretchBlt-Modi

Beschreibung

Diese Konstanten sind Bitmap-Dehnungsmodi für **GetStretchBltMode** und **SetStretchBltMode**.

Konstante	Bedeutung
BlackOnWhite	Mit AND werden die Linien der Bitmap, aber nicht die schwarzen Flächen entfernt.
ColorOnColor	Linien werden entfernt, was immer auch ihr Inhalt ist. Information geht verloren.
WhiteOnBlack	Mit **OR** werden die Linien der Bitmap, aber nicht die weißen Flächen entfernt.

sw_xxx (ShowWindow)

Beschreibung

Diese Konstanten indizieren den Status, in dem die Funktion **ShowWindow** das Fenster darstellt.

Konstante	Bedeutung
sw_Hide	Verdeckt
sw_Maximize	Wie sw_ShowMaximized.
sw_Minimize	Verkleinert und nicht aktiv
sw_Normal	Wie sw_ShowNormal.
sw_OtherZoom	Anderes Fenster ist vergrößert. (Windows 2.0 kompatibel)
sw_OtherUnzoom	Anderes Fenster ist verkleinert. (Windows 2.0 kompatibel)
sw_Restore	Wie sw_ShowNormal.
sw_Show	In aktueller Größe und Position des Fensters.
sw_ShowMaximized	Vergrößert und aktiv
sw_ShowMinimized	Verkleinert und aktiv
sw_ShowMinNoActive	Verkleinert. Berührt Aktivierung nicht
sw_ShowNA	Im aktuellen Status. Berührt Aktivierung nicht
sw_ShowNoActivate	In aktueller Fenstergröße und -position. Berührt Aktivierung nicht
sw_ShowNormal	Restauriert und aktiv

sw_xxx

Beschreibung

Diese Bezeichner geben den Status des gezeigten Fensters an und werden in der Botschaft **wm_ShowWindow** weitergegeben.

Konstante	Bedeutung
sw_ParentClosing	Übergeordnetes Fenster ist verkleinert oder Popup-Fenster verdeckt.
sw_ParentOpening	Übergeordnetes Fenster des Fensters wird angezeigt oder ist Popup-Fenster

swp_xxx

Beschreibung

Diese Flags werden in den Funktionen **SetWindowPos** und **DeferWindowPos** weitergegeben, um eine oder mehrere Aktionen für das angegebene Fenster anzuzeigen.

Konstante	Bedeutung
swp_DrawFrame	Zeichnet einen Rahmen (definiert in der Klassenbeschreibung des Fensters) um das Fenster.
swp_HideWindow	Versteckt das Fenster.
swp_NoActivate	Aktiviert das Fenster nicht.
swp_NoMove	Behält die aktuelle Position bei (ignoriert die Parameter x und y).
swp_NoRedraw	Zeichnet Veränderungen nicht neu.
swp_NoSize	Behält die aktuelle Größe bei (ignoriert die Parameter cx und cy).
swp_NoZOrder	Behält die aktuelle Reihenfolge bei (ignoriert den Parameter hWndInsertAfter).
swp_ShowWindow	Zeigt das Fenster.

syspal_xxx

Beschreibung

Diese Konstanten sind für die Systempalette. In den Funktionen **GetSystemPaletteUse** und **SetSystemPaletteUse** verwendet.

Konstante	Bedeutung
syspal_NoStatic	Systempalette enthält keine statische Farbe außer schwarz und weiß.
syspal_Static	Systempalette enthält statische Farben, die sich nicht ändern, wenn ein Programm seine logische Palette anwendet.

ta_xxx

Beschreibung

Textausrichtung mit den Funktionen **TextOut** und **ExtTextOut**. Sie werden in den Funktionen **GetTextAlign** und **SetTextAlign** spezifiziert und zurückgegeben.

Konstante	Bedeutung
ta_BaseLine	Legt die Ausrichtung des Punktes an der Grundlinie der gewählten Schrift fest.
ta_Bottom	Legt die Ausrichtung des Punktes an der Unterseite des umschließenden Rechtecks fest.
ta_Center	Legt die Ausrichtung des Punktes am horizontalen Mittelpunkts des umschließenden Rechtecks fest.
ta_Left	Legt die Ausrichtung des Punktes an der linken Seite des umschließenden Rechtecks fest.
ta_NoUpdateCP	Legt fest, daß die aktuelle Position nicht nach jedem Aufruf der Funktionen TextOut oder ExtTextOut aktualisiert wird.
ta_Right	Legt die Ausrichtung des Punktes an der rechten Seite des umschließenden Rechtecks fest.
ta_Top	Legt die Ausrichtung des Punktes an der Oberseite des umschließenden Rechtecks fest.
ta_UpdateCP	Legt fest, daß die aktuelle Position nach jedem Aufruf der Funktionen TextOut oder ExtTextOut aktualisiert wird.

tc_xxx

Beschreibung

Diese Konstanten repräsentieren die Textdarstellung eines Geräts.

Konstante	Bedeutung
tc_cp_Stroke	Kann vektoriell mit Clipping ausgeben
tc_cr_90	Kann 90 Grad Rotation.
tc_cr_Any	Kann jede Zeichenrotation.
tc_ea_Double	Kann doppelt dichte Zeichen
tc_ia_Able	Kann kursive Zeichen
tc_op_Character	Kann Zeichen in Präzisionsdarstellung
tc_op_Stroke	Kann Strich in Präzisionsdarstellung
tc_ra_Able	Kann Rasterschriften
tc_sa_Contin	Kann multiple Skalierung
tc_sa_Double	Kann doppelte Zeichen-Skalierung
tc_sa_Integer	Kann Integer-Skalierung
tc_sf_X_YIndep	Kann Skalierung unabhängig von X und Y.
tc_so_Able	Kann durchgestrichene Zeichen.
tc_ua_Able	Kann unterstrichene Zeichen.
tc_va_Able	Kann Vektorschriften

tf_ForceDrive

Beschreibung

Kombiniert mit dem Argument DriveLetter in einem Aufruf der Funktion **GetTempFileName** stellt tf_ForceDrive sicher, daß auf dem angegebenen Laufwerk eine temporäre Datei angelegt wird. Ansonsten auf der ersten Festplatte oder im Verzeichnis der Variablen TEMP.

Rasteroperationen

Beschreibung

Diese Konstanten werden als Rastercodes verwendet.

Konstante	Bedeutung
Blackness	Darstellung völlig schwarz
DSTInvert	Invertierte Bitmap.
MergeCopy	Muster und Bitmap mit logischem **AND** kombiniert.
MergePaint	Ziel und invertierte Quell-Bitmaps mit logischem **OR** kombiniert.
NotSrcCopy	Quell-Bitmap invertiert und auf Ziel-Bitmap kopiert.
NotSrcErase	Ergebnis von MergePaint invertiert.
PatCopy	Kopiert Muster auf Bitmap.
PatInvert	Ziel-Bitmap und Muster mit logischem **XOR** kombiniert.
PatPaint	Quell-Bitmap mit Muster und logischem **OR**, dann mit Ziel-Bitmap und **OR** kombiniert.
SrcAnd	Quell- und Ziel-Bitmaps mit logischem **AND** kombiniert.
SrcCopy	Kopiert Quell-Bitmap auf Ziel-Bitmap
SrcErase	Quell- und invertierte Ziel-Bitmaps mit logischem **AND** kombiniert.
SrcInvert	Ziel und Quell-Bitmaps mit logischem **XOR** kombiniert.
SrcPaint	Ziel und Quell-Bitmaps mit logischem **OR** kombiniert.
Whiteness	Völlig weiße Ausgabe.

Diese Konstanten werden in folgenden Funktionen verwendet:

BitBlt
PatBlt
StretchBlt
StretchDIBits

vk_xxx

Beschreibung

Virtuelle Tastaturcodes sind Konstanten, die Standardtasten des Rechners darstellen. Tastatureingaben sollten mit diesen Codes bearbeitet werden, da verschiedene Computer verschiedene Tastaturen haben können.

Diese Codes dienen zur Definition von Tastenkürzeln oder zum Behandeln der Botschaften **wm_KeyDown**, **wm_KeyUp**, **wm_SysKeyDown**, und **wm_SysKeyUp**.

Virtueller Code	Taste oder Aktionsschalter
vk_A bis vk_Z	'A' bis 'Z'
vk_0 bis vk_9	'0' bis '9'
vk_NumPad0 bis vk_NumPad9	'0' bis '9' (numerischer Tastenblock)
vk_LButton	Linke Maustaste
vk_RButton	Rechte Maustaste
vk_Cancel	Control-break Verarbeitung
vk_MButton	Mittlere Maustaste
vk_Back	Backspace
vk_Tab	Tab
vk_Clear	Clear
vk_Return	Return
vk_Shift	Shift
vk_Control	Ctrl (Strg)
vk_Menu	Menütaste
vk_Pause	Pause
vk_Capital	Caps Lock
vk_Escape	Esc
vk_Space	Leertaste
vk_Prior	Page Up (Bild auf)
vk_Next	Page Down (Bild ab)
vk_End	End (Ende)
vk_Home	Home (Pos1)
vk_Left	Pfeil links
vk_Up	Pfeil auf
vk_Right	Pfeil rechts
vk_Down	Pfeil ab
vk_Select	Select
vk_Print	OEM spezifisch
vk_Execute	Execute
vk_SnapShot	Printscreen
vk_Insert	Insert (Einfg)
vk_Delete	Delete (Entf)
vk_Help	Help
vk_Multiply	Multiplizieren (grau '*')
vk_Add	Addieren (grau '+')
vk_Separator	Separator
vk_Subtract	Dividieren (grau '-')
vk_Decimal	Dezimal (numerische Tasten '.')
vk_Divide	Dividieren (grau '/')
vk_F1 bis vk_F16	Funktionstasten F1 bis F16
vk_NumLock	Num Lock

wep_xxx

Beschreibung

Diese Konstanten werden in der Variablen ExitCode verwendet, die von einer dynamischen Linkbibliothek zugänglich ist

Konstante	Bedeutung
wep_Free_DLL	Nur die DLL wird beendet und aus dem Speicher entfernt
wep_System_Exit	Windows selbst wird beendet

wf_xxx

Beschreibung

Diese Konstanten geben dei aktuelle Windows-Speicherkonfiguration an. Sie werden als Rückgabewerte von **GetWinFlags** verwendet.

Konstante	Bedeutung
wf_80x87	Intel Mathematik-Koprozessor
wf_CPU086	CPU ist 8086.
wf_CPU186	CPU ist 80186.
wf_CPU286	CPU ist 80286.
wf_CPU386	CPU ist 80386.
wf_CPU486	CPU ist 80486.
wf_Enhanced	Windows läuft im 386-Enhanced (protected) Modus.
wf_LargeFrame	System ist als large frame konfiguriert.
wf_PMode	Windows läuft im protected (386-Enhanced oder Standard) Modus.
wf_SmallFrame	System ist als small frame konfiguriert.
wf_Standard	Windows läuft im Standard (protected) Modus.
wf_Win286	Wie wf_Standard oder Standard Modus.
wf_Win386	Wie wf_Enhanced oder 386 Enhanced Modus.

wh_xxx

Beschreibung

Filterfunktion, die mit **SetWindowsHook** oder **UnhookWindowsHook** aus einer Kette von Filterfunktionen eingefügt oder gelöst werden muß.

Konstante	Bedeutung
wh_CallWndProc	Fensterfunktionsfilter.
wh_GetMessage	Botschaftssfilter.
wh_JournalPlayback	Mitschriftfilter.
wh_JournalRecord	Mitschriftaufzeichnungsfilter.
wh_Keyboard	Tastaturfilter.
wh_MsgFilter	Botschaftsfilter (nur SetWindowsHook.)
wh_SysMsgFilter	Systembotschaftsfilter (nur SetWindowsHook)

ws_xxx

Beschreibung

Diese Konstanten werden kombiniert verwendet, um den Stil von Fenstern beim Erstellen von Fenstern, Dialogfenstern und Elementen mit den Funktionen **CreateWindow** und **CreateWindowEx** zu bestimmen.

Konstante	Bedeutung
ws_Border	Erzeugt ein Fenster, das eine Umrandung hat.
ws_Caption	Erzeugt ein Fenster, das eine Titelleiste hat (hat den Stil ws_Border zur Folge). Dieser Stil kann nicht mit dem Stil ws_DlgFrame benutzt werden.
ws_Child	Erzeugt ein untergeordnetes Fenster. Kann nicht mit dem Stil ws_Popup benutzt werden.
ws_ChildWindow	Wie ws_Child.
ws_ClipChildren	Schließt den von untergeordneten Fenstern belegten Bereich aus, wenn innerhalb des übergeordneten Fensters gemalt wird. Wird benutzt, wenn das übergeordnete Fenster erzeugt wird.
ws_ClipSiblings	Schneidet untergeordnete Fenster relativ zueinander ab; das heißt, wenn ein bestimmtes untergeordnetes Fenster eine Malbotschaft erhält, schneidet der Stil ws_ClipSiblings alle anderen überlappten untergeordneten Fenster aus der Region des untergeordneten Fensters heraus, das aktualisiert werden soll. (Wenn ws_ClipSiblings nicht gesetzt ist und untergeordnete Fenster überlappen, ist es möglich, wenn man innerhalb des Client-Bereichs eines unterge- ordneten Fensters malt, auch innerhalb des Client- Bereichs eines benachbarten untergeordneten Fensters zu zeichnen.) Nur zum Gebrauch mit dem Stil ws_Child.
ws_Disabled	Erzeugt ein Fenster, das anfänglich nicht aktiviert ist.
ws_DlgFrame	Erzeugt ein Fenster mit einer doppelten Umrandung, aber ohne Titelleiste.
ws_Group	Gibt das erste Steuerelement aus einer Gruppe von Elementen an, in der der Benutzer sich von einem Steuerelement zum nächsten bewegen kann, indem er die Richtungstasten benutzt. Alle Steuerelemente, die nach dem ersten Element mit dem Stil ws_Group definiert wurden, gehören zur gleichen Gruppe. Das nächste Steuerelement mit dem Stil ws_Group beendet die Stilgruppe und beginnt die nächste Gruppe (das heißt, eine Gruppe hört da auf, wo die nächste beginnt). Nur Dialogfenster benutzen diesen Stil.
ws_HScroll	Erzeugt ein Fenster, das eine horizontale Bildlaufleiste hat.
ws_Iconic	Wie ws_Minimize.
ws_Maximize	Erzeugt ein Fenster von maximaler Größe.
ws_MaximizeBox	Erzeugt ein Fenster, das ein Feld zur Ausdehnung auf größtmöglichen Umfang hat.
ws_Minimize	Erzeugt ein Fenster von minimaler Größe.
ws_MinimizeBox	Erzeugt ein Fenster, das ein Feld zur Reduzierung auf minimale Größe hat.
ws_Overlapped	Erzeugt ein überlapptes Fenster. Ein überlapptes Fenster

	hat eine Überschrift und eine Umrandung.
ws_OverlappedWindow	Erzeugt ein überlapptes Fenster mit den folgenden Stilen: ws_Overlapped ws_Caption ws_SysMenu ws_ThickFrame ws_MinimizeBox ws_MaximizeBox
ws_Popup	Erzeugt ein Pop-up-Fenster. Kann nicht mit dem Stil ws_Child benutzt werden.
ws_PopupWindow	Erzeugt ein Pop-up-Fenster, das die Stile ws_Border, ws_Popup und ws_SysMenu hat. Der Stil ws_Caption muß mit dem Stil ws_PopupWindow kombiniert werden, um das Systemmenü sichtbar zu machen.
ws_SizeBox	Wie ws_ThickFrame.
ws_SysMenu	Erzeugt ein Fenster, das ein Systemmenü in seiner Titelleiste hat. Wird nur für Fenster mit Titelleisten benutzt.
ws_TabStop	Gibt eines der Steuerelemente an, durch die sich der Benutzer mit der TAB-Taste bewegen kann. Die TAB-Taste bewegt den Fokus zum nächsten Steuerelement, das durch den Stil ws_TabStop angegeben wird. Nur Dialogfenster verwenden diesen Stil.
ws_ThickFrame	Erzeugt ein Fenster mit einem dicken Rahmen, der benutzt werden kann, um die Größe des Fensters zu verändern.
ws_Tiled	Wie ws_Overlapped.
ws_TiledWindow	Wie ws_OverlappedWindow.
ws_Visible	Erzeugt ein Fenster, das anfänglich sichtbar ist.
ws_VScroll	Erzeugt ein Fenster, das eine vertikale Bildlaufleiste hat.

Stile, die nicht zusammen verwendet werden können:

ws_Border	&	ws_DlgFrame
ws_caption	&	ws_DlgFrame
ws_Child	&	ws_PopUp

Identische Stile:

ws_Child	&	ws_ChildWindow
ws_Iconic	&	ws_Minimize
ws_SizeBox	&	ws_ThickFrame
ws_Tiled	&	ws_Overlapped
ws_TiledWindow	&	ws_OverlappedWindow

ws_ex_xxx

Beschreibung

Diese Konstanten bestimmen den Stil beim Erzeugen von Fenstern, Dialogfenstern und Elementen mit den Funktionen **CreateWindow** und **CreateWindowEx**.

Konstante	Bedeutung
ws_ex_DlgModalFrame	Bestimmt ein Fenster mit einer doppelten Umrandung,das wahlweise mit einer Titelleiste erzeugt werden kann, indem man den Stil ws_Caption im Parameter Style angibt.
ws_ex_NoParentNotify	Gibt an, daß ein untergeordnetes Fenster, das mit diesem Stil erzeugt wird, keine Botschaft **wm_ParentNotify** an sein übergeordnetes Fenster sendet, wenn das untergeordnete Fenster erzeugt oder zerstört wird.

bm_GetCheck

Beschreibung

Diese Funktion ermittelt, ob ein Auswahlfeld oder ein Schaltfeld gewählt ist.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Auswahlfeld oder das Schaltfeld markiert ist. Andernfalls ist er 0. Die Botschaft gibt für einen Schalter immer 0 zurück.

bm_GetState

Beschreibung

Diese Botschaft bestimmt den Status eines Schalters, wenn der Benutzer eine Maustaste oder die Leertaste drückt.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Die Botschaft bm_getstate gibt einen Wert ungleich 0 zurück, wenn eine der folgenden Bedingungen erfüllt ist:

- Ein Schalter ist hervorgehoben.
- Der Benutzer drückt eine Maustaste oder die Leertaste, wenn ein Aktionsschalter den Fokus hat.

bm_SetCheck

Beschreibung

Diese Botschaft markiert ein Schaltfeld oder ein Auswahlfeld oder löscht die Auswahlmarkierung aus einem Schaltfeld oder einem Auswahlfeld.

Parameter

wParam	Gibt an, ob ein Schalter oder ein Feld markiert werden oder die Markierung gelöscht werden soll. Wenn der Parameter wParam ungleich 0 ist, wird eine Markierung gesetzt; ist er 0, dann wird die Markierung (falls vorhanden) gelöscht. Bei dreistufigen Schaltern wird eine Markierung neben den Schalter gesetzt, wenn wParam 1 ist. Ist wParam 2, wird der erste Schalter in den nicht aktivierbaren Zustand überführt. Ist wParam 0, wird der Schalter in den normalen Zustand zurückgesetzt (keine Markierung, aber aktivierbar).
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

bm_SetState

Beschreibung

Diese Botschaft ändert den Status eines Schalters oder eines Auswahlfeldes.

Parameter

wParam	Wenn der Parameter wParam ungleich 0 ist, dann wird der Schalter hervorgehoben (das Innere wird invertiert dargestellt). Wenn wParam 0 ist, dann wird der Schalter in seinem regulären Status gezeichnet.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

bm_SetStyle

Beschreibung

Diese Botschaft ändert den Stil eines Schalters. Wenn der Stil, der im Parameter wParam angegeben wird, vom vorgefundenen Stil abweicht, wird der Schalter in der neuen Form gezeichnet.

Parameter

wParam	Gibt den Wert für den Stil an. (siehe **bs_xxx-Konstanten**)
lParam	Gibt an, ob die Schalter neu gezeichnet werden sollen. Ist lParam 0, dann werden die Schalter nicht neu gezeichnet. Ist lParam ungleich 0, dann werden sie neu gezeichnet.

Rückgabewert

Nicht verwendet

cb_AddString

Beschreibung

Diese Botschaft fügt einen String zur Liste eines Kombinationsfensters hinzu.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf den null-terminierten String, der hinzugefügt werden soll.

Rückgabewert

Der Rückgabewert gibt bei erfolgreicher Ausführung den Index zum eingefügten String in der Liste an. Ansonsten ist er **cb_Err**, wenn ein Fehler auftritt. **cb_ErrSpace**, wenn zum Speichern des neuen Strings nicht genügend Speicher zur Verfügung steht.

Hinweise

Ist die Liste des Kombinationsfensters nicht sortiert, wird der String am Ende eingefügt. Wird das Kombinationsfenster mit einer der **cbs_xxx-Konstanten** cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable, jedoch ohne cbs_HasStrings erstellt:

- ist lParam ein 32-Bit-Wert, der anstelle eines String gespeichert wird
- die Botschaft **wm_CompareItem** wird ein- oder mehrmals an den Besitzer des Kombinationsfensters gesendet, so daß der neue Eintrag richtig in der Liste angeordnet werden kann.

cb_DeleteString

Beschreibung

Löscht einen String aus der Liste eines Kombinationsfensters.

Parameter

wParam	Enthält einen Index zu dem String, der gelöscht werden soll.
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Anzahl der Strings, die in der Liste verbleiben. Der Rückgabewert ist **cb_Err**, wenn wParam keinen gültigen Index bezeichnet.

Hinweise

Wird das Kombinationsfenster mit einer der **cbs_xxx-Konstanten** cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable, jedoch ohne cbs_HasStrings erstellt:

- ist lParam ein 32-Bit-Wert, der anstelle eines String gespeichert wird
- die Botschaft **wm_DeleteItem** wird ein- oder mehrmals an den Besitzer des Kombinationsfensters gesendet, so daß der neue Eintrag richtig in der Liste angeordnet werden kann.

cb_Dir

Beschreibung

Diese Botschaft fügt eine Liste der Dateien mit der gegebenen Dateispezifikation aus dem aktuellen Verzeichnis zur Liste hinzu.

Parameter

wParam	Enthält einen DOS-Attributwert.
lParam	Zeigt auf einen null-terminierten Dateispezifikationsstring. Der String kannJokerzeichen enthalten (z.B.: *.*).

Rückgabewert

Der Rückgabewert gibt den Index des letzten Eintrags der resultierenden Liste zurück..

Der Rückgabewert ist **cb_Err**, wenn ein Fehler auftritt; der Rückgabewert ist **cb_ErrSpace,** wenn zum Speichern des neuen Strings nicht genügend Speicherplatz vorhanden ist.

cb_FindString

Beschreibung

Diese Botschaft findet den ersten String in der Liste eines Kombinationsfensters, der mit dem angegebenen Präfixtext übereinstimmt.

Parameter

wParam — Enthält den Index zu dem Eintrag vor dem ersten zu suchenden Eintrag. Wenn die Suche das Ende einer Liste erreicht, dann wird sie vom Listenbeginn an bis zu dem Element, das wParam bezeichnet, fortgesetzt. Wenn der Parameter wParam -1 ist, dann wird die ganze Liste von Beginn an durchsucht.

lParam — Enthält einen Zeiger auf den null-terminierten Präfix-String

Rückgabewert

Der Rückgabewert ist der Index des übereinstimmenden Eintrags oder **cb_Err**, wenn die Suche nicht erfolgreich war.

Hinweise

Wird das Kombinationsfenster mit einer der **cbs_xxx-Konstanten** cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable**,**jedoch ohne cbs_HasStrings erstellt, ist lParam ein 32-Bit-Wert, der anstelle eines String gespeichert wird

cb_GetCount

Beschreibung

Diese Botschaft gibt die Anzahl der Einträge in der Liste eines Kombinationsfensters zurück.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Anzahl der Einträge in der Liste eines Kombinationsfensters.

cb_GetCurSel

Beschreibung

Diese Botschaft gibt den Index des aktuell ausgewählten Eintrags in der Liste eines Kombinationsfensters an, falls ein solcher vorhanden ist.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist der Index des aktuell ausgewählten Eintrags. Er ist **cb_Err**, wenn kein Eintrag ausgewählt ist.

cb_GetEditSel

Beschreibung

Diese Botschaft gibt die Start- und Endpositionen des ausgewählten Texts in das Editierfeld eines Kombinationsfensters zurück.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist ein LongInt-Wert, der die Startposition im niederwertigen Word und die Endposition im höherwertigen Word angibt. Wenn die Botschaft an ein Kombinationsfenster ohne Editierfeld gesendet wurde, dann ist der Rückgabewert **cb_Err**.

cb_GetItemData

Beschreibung

Diese Botschaft ruft den von der Anwendung bereitgestellten 32-Bit-Wert ab, der dem angegebenen Kombinationsfenstereintrag zugeordnet ist.

Parameter

wParam	The entry index
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist der 32-Bit-Wert, der dem Eintrag zugeordnet ist oder **cb_Err**, wenn ein Fehler auftritt.

cb_GetLBText

Beschreibung

Diese Botschaft kopiert einen String aus der Liste eines Kombinationsfensters in einen Puffer.

Parameter

wParam	Enthält den Index des Strings, der kopiert werden soll.
lParam	Zeigt auf einen Puffer, der den String empfangen soll. Der Puffer muß ausreichend Speicherplatz für den String und ein abschließendes 0zeichen haben.

Rückgabewert

Nicht verwendet

Hinweise

Wird das Kombinationsfenster mit einer der **cbs_xxx-Konstanten**cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable, jedoch ohne cbs_HasStrings erstellt, wird der 32-Bit-Wert für den Listeneintrag in den Puffer kopiert.

cb_GetLBTextLen

Beschreibung

Diese Botschaft gibt die Länge eines Strings in der Liste eines Kombinationsfensters zurück.

Parameter

wParam	Index des Strings
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Länge des Strings in Bytes, ohne Berücksichtigung des abschließenden Nullzeichens. Wenn der Parameter wParam keinen gültigen Index angibt, ist der Rückgabewert **cb_Err**.

cb_InsertString

Beschreibung

Diese Botschaft fügt einen String in die Liste eines Kombinationsfensters ein. Die Liste wird nicht sortiert.

Parameter

wParam	Enthält einen Index zu der Position, die den String empfangen soll. Wenn der Parameter wParam -1 ist, dann wird der String an das Ende der Liste angefügt.
lParam	Zeigt auf den null-terminierten String, der eingefügt werden soll.

Rückgabewert

Der Rückgabewert ist der Index der Position, an der der String eingefügt wurde. Der Rückgabewert ist **cb_Err**, wenn ein Fehler auftritt; der Rückgabewert ist **cb_ErrSpace**, wenn zum Abspeichern des neuen Strings nicht genügend Speicherplatz zur Verfügung steht.

cb_LimitText

Beschreibung

Diese Botschaft begrenzt die Länge (in Bytes) des Textes, den der Benutzer in das Editierfeld eines Kombinationsfensters eingeben kann.

Parameter

wParam	Gibt die Maximalzahl von Bytes an, die der Benutzer eingeben kann.
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Botschaft erfolgreich war; andernfalls ist er 0. Wird diese Botschaft an ein Kombinationsfenster ohne Editierfeld gesendet, dann ist der Rückgabewert **scb_Err**.

cb_ResetContent

Beschreibung

Diese Botschaft löscht alle Strings aus der Liste eines Kombinationsfensters.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Wird das Kombinationsfenster mit einer der **cbs_xxx-Konstanten** cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable, jedoch ohne cbs_HasStrings erstellt, wird eine Botschaft **wm_DeleteItem** für jeden String an den Besitzer des Kombinationsfensters gesendet.

cb_SelectString

Beschreibung

Diese Botschaft wählt den ersten String in der Liste eines Kombinationsfensters aus, der mit dem angegebenen Präfix übereinstimmt. Der Text im Editierfeld des Kombinationsfensters wird verändert, um die neue Auswahl wiederzuspiegeln.

Parameter

wParam	Enthält den Index zu dem Eintrag vor dem ersten Eintrag, ab dem gesucht werden soll. Wenn die Suche das Ende der Liste erreicht (wParam ist dann 0), wird sie vom Listenbeginn an bis zu dem vom Parameter wParam bezeichneten Eintrag fortgeführt. Wenn der Parameter wParam -1 ist, dann wird die ganze Liste von Beginn an durchsucht.
lParam	Zeigt auf den null-terminierten Präfix-String.

Rückgabewert

Der Rückgabewert ist der Index des neu ausgewählten Eintrags. Wenn die Suche nicht erfolgreich war, dann ist der Rückgabewert **cb_Err** und die aktuelle Auswahl wird nicht verändert.

Hinweise

Wird das Kombinationsfenster mit einer der **cbs_xxx-Konstanten**cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable, jedoch ohne cbs_HasStrings erstellt:, ist lParam ein 32-Bit-Wert, der mit jedem 32-Bit-Wert der Liste verglichen wird.

cb_SetCurSel

Beschreibung

Diese Botschaft wählt einen String aus der Liste eines Kombinationsfensters aus und rollt ihn ins Blickfeld, wenn die Liste sichtbar ist.

Parameter

wParam	Enthält den Index; falls -1, keine Auswahl
lParam	Nicht verwendet

Rückgabewert

Liefert den Index des gesuchten Eintrags. Wenn der von wParam bezeichnete Index nicht gültig ist oder wParam -1, dann ist der Rückgabewert **cb_Err** und die aktuelle Auswahl wird nicht verändert.

cb_SetEditSel

Beschreibung

Diese Botschaft wählt alle bezeichneten Zeichen im Editierfeld eines Kombinationsfensters aus.

Parameter

wParam	Nicht verwendet
lParamLo	Startposition der Zeichenauswahl
lParamHi	Endposition der Zeichenauswahl

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Botschaft erfolgreich war; andernfalls ist er 0. Wenn die Botschaft an ein Kombinationsfenster ohne Editierfeld gesendet wird, dann ist der Rückgabewert **cb_Err**.

cb_SetItemData

Beschreibung

Diese Botschaft setzt den 32-Bit-Wert, der dem angegebenen Eintrag in einem Kombinationsfenster zugeordnet ist.

Parameter

wParam	Enthält einen Index zu dem Eintrag.
lParam	Enthält den neuen Wert, der dem Eintrag zugeordnet werden soll.

Rückgabewert

cb_Err im Fehlerfall.

cb_ShowDropDown

Beschreibung

Diese Botschaft zeigt die Drop-Down-Liste eines Kombinationsfenster.

Parameter

wParam	Ist er ungleich 0, wird die Liste dargestellt, wenn sie nicht schon sichtbar ist. Ist er 0, wird die Liste ausgeblendet, wenn sie sichtbar ist.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft bezieht sich nur auf Dialogfenster, die mit den **cbs_xxx-Konstanten** cbs_DropDown oder cbs_DropDownList erstellt wurden.

dm_GetDefID

Beschreibung

ID des Vorgabe-Aktionsschalters eines Dialogfensters.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

dc_HasDefID im höherwertigen Word, die ID des Vorgabe-Aktionsschalters im niederwertigen. Gibt es im Dialogfenster keinen Vorgabe-Aktionsschalter, wird 0 im höherwertigen Word zurückgegeben.

dm_SetDefID

Beschreibung

Diese Botschaft wird von einer Anwendung benutzt, um die ID für den als Vorgabewert vorgesehenen Aktionsschalter eines Dialogfensters zu ändern.

Parameter

wParam	ID des neuen Vorgabe-Schalters
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

em_CanUndo

Beschreibung

Diese Botschaft bestimmt, ob eine Editierfeld korrekt auf eine Botschaft **em_Undo** reagieren kann.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn das Editierfeld auf die Botschaft em_Undo korrekt reagieren kann. Andernfalls ist er 0.

em_EmptyUndoBuffer

Besachreibung

Diese Botschaft weist ein Editierfeld an, seinen Widerruf-Puffer zu löschen. Dies nimmt dem Editierfeld die Fähigkeit, den letzten Editiervorgang rückgängig zu machen.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Der Puffer zum Widerrufen wird automatisch geleert, wenn das Editierfeld eine Botschaft **wm_SetText** oder **em_SetHandle** empfängt.

em_FmtLines

Beschreibung

Diese Botschaft weist ein Mehrzeilen-Editierfeld an, das Zeilenendezeichen aus umgebrochenen Zeilen zu entfernen oder hinzuzufügen.

Parameter

wParam	Zeigt die Verwendung von Zeilenendezeichen an. Wenn der Parameter wParam ungleich 0 ist, dann werden die Zeichen CR CR LF (0D 0D 0A hexadezimal) an das Ende der umgebrochenen Zeilen gesetzt. Ist wParam 0, dann werden die Zeilenendezeichen aus dem Text gelöscht.
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn eine Formatierung erfolgt. Andernfalls ist er 0.

Hinweise

Zeilen, die mit einem Hardreturn enden (einem Wagenrücklaufzeichen, das vom Benutzer eingegeben wurde), enthalten die Zeichen CR LF am Zeilenende. Diese Zeilen werden von der Botschaft nicht verändert.

em_GetHandle

Beschreibung

Diese Botschaft gibt das Daten-Handle des Puffers zurück, der den Inhalt des Steuerelementfensters aufgenommen hat. Das Handle ist immer ein lokales Handle auf eine Position im Datensegment der Anwendung.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist das Daten-Handle des Puffers, der den Inhalt des Editierfeldes enthält.

Hinweise

Eine Anwendung kann diese Botschaft nur an ein Steuerelement senden, wenn sie das Dialogfenster, welches das Steuerelement enthält, mit gesetztem Flag **ds_LocalEdit** erzeugt hat.

em_GetLine

Beschreibung

Diese Botschaft kopiert eine Zeile aus dem Editierfeld.

Parameter

wParam	Gibt die Zeilennummer der Zeile im Steuerelement an, wobei die Zeilennummer der ersten Zeile 0 ist.
lParam	Zeigt auf den Puffer, in dem die Zeile gespeichert werden soll. Das erste Word des Puffers gibt die maximale Anzahl von Bytes an, die in den Puffer kopiert werden sollen.

Rückgabewert

Der Rückgabewert ist die Anzahl von Bytes, die tatsächlich kopiert wurden. Diese Botschaft wird von Einzeileneditierfeldern nicht bearbeitet.

Hinweise

Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_GetLineCount

Beschreibung

Diese Botschaft gibt die Anzahl von Textzeilen im Editierfeld zurück.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Zahl der Textzeilen.

Hinweise

Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_GetModify

Beschreibung

Diese Botschaft gibt den aktuellen Wert des Veränderungsflags für ein gegebenes Editierfeld an. Das Flag wird vom Steuerelement gesetzt, wenn der Benutzer Text innerhalb des Steuerelements eingibt oder bearbeitet, oder wenn die Botschaft **em_SetModify** an das Editierfeld gesendet wird.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist der Wert des aktuellen Veränderungsflags für ein angegebenes Steuerelement oder 0, falls keine Änderung des Textes vorgenommen wurde.

em_GetRect

Beschreibung

Diese Botschaft ruft das formatierende Rechteck eines Steuerelements ab.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf eine Rect-Datenstruktur. Das Steuerelement kopiert die Abmessungen der Struktur.

Rückgabewert

Nicht verwendet

em_GetSel

Beschreibung

Diese Botschaft gibt die Position der Start- und Endzeichen der aktuellen Auswahl an.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert enthält die Startposition im niederwertigen Word. Er enthält im höherwertigen Word die Position des ersten nicht ausgewählten Zeichens nach dem Ende der Auswahl.

em_LimitText

Beschreibung

Diese Botschaft begrenzt die Länge des Textes, den der Benutzer eingeben kann (in Bytes).

Parameter

wParam	Bestimmt die maximale Anzahl von Bytes, die eingegeben werden können. Wenn der Benutzer versucht, mehr Zeichen einzugeben, wird ein Warnton ausgegeben und die Zeichen werden nicht angenommen. Wenn der Parameter wParam 0 ist, wird die Textgröße nur durch den verfügbaren Speicherplatz begrenzt.
lParam	Nicht verwendet

Rückgabewert

Im Erfolgsfall ungleich 0, sonst 0.

em_LineFromChar

Beschreibung

Diese Botschaft gibt die Zeilennummer der Zeile zurück, die das Zeichen enthält, dessen Position (gezählt vom Textanfang an) vom Parameter wParam vorgegeben wird.

Parameter

wParam	Enthält den Indexwert für das gewünschte Zeichen im Text des Editierfeldes (diese Indexwerte beginnen mit 0) oder -1.
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist eine Zeilennummer. Wenn wParam -1 ist, dann wird die Nummer der Zeile zurückgegeben, die das erste Zeichen der Auswahl enthält. Andernfalls enthält wParam den Index (oder die Position) des gewünschten Zeichens im Text des Editierfeldes und es wird die Nummer der Zeile, die das Zeichen enthält, zurückgegeben.

em_LineIndex

Beschreibung

Diese Botschaft gibt die Anzahl von Zeichenpositionen zurück, die vor dem ersten Zeichen einer angegebenen Zeile auftreten.

Parameter

wParam	Bezeichnet die gewünschte Zeilennummer, wobei die Zeilennummer der ersten Zeile 0 ist. Wenn der Parameter wParam -1 ist, dann wird die aktuelle Zeilennummer (die Zeile, die das Caret enthält) benutzt.
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Anzahl von Zeichenpositionen, die dem ersten Zeichen in der vorgegebenen Zeile vorangehen.

Hinweise

Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_LineLength

Beschreibung

Diese Botschaft gibt die Länge einer Zeile im Textpuffer eines Editierfeldes zurück (in Bytes).

Parameter

wParam	Gibt den Zeichenindex eines Zeichens in der angegebenen Zeile an oder -1.
lParam	Nicht verwendet

Rückgabewert

Ist der Parameter wParam -1, dann wird die Länge der aktuellen Zeile (der Zeile, die das Caret enthält), zurückgegeben - ohne Berücksichtigung eines ausgewählten Texts. Wenn die aktuelle Auswahl mehr als eine Zeile umfaßt, dann wird die gesamte Länge der Zeilen abzüglich des ausgewählten Texts zurückgegeben.

em_LineScroll

Beschreibung

Diese Botschaft rollt den Inhalt des Steuerelements um die angegebene Anzahl von Zeilen.

Parameter

wParam	Nicht verwendet
lParamLo	Anzahl vertikal zu verschiebender Zeilen.
lParamHi	Anzahl horizontal zu verschiebender Zeichen.

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_ReplaceSel

Beschreibung

Diese Botschaft ersetzt die aktuelle Auswahl durch neuen Text.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf einen null-terminierten String, der den Ersatztext enthält.

Rückgabewert

Nicht verwendet

em_SetHandle

Beschreibung

Diese Botschaft richtet den Textpuffer ein, der benutzt wird, um die Inhalte des Steuerelementfensters zu speichern.

Parameter

wParam	Ein lokales Handle zum Textpuffer des Editierfelds.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Anwendung muß das vorherige Puffer-Handle mit **em_GetHandle** abrufen, bevor sie das neue Handle setzt und das alte Handle mit der Funktion **LocalFree** freigeben.

Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_SetModify

Beschreibung

Diese Botschaft setzt das Veränderungsflag für ein gegebenes Editierfeld.

Parameter

wParam	Gibt einen neuen Wert für das Veränderungsflag an.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

em_SetPasswordChar

Beschreibung

Diese Botschaft setzt das Zeichen, das in einem mit dem Modus **es_Password** erzeugten Editierfeld angezeigt wird.

Parameter

wParam	Gibt das Zeichen an, das anstelle des tatsächlich vom Benutzer eingegebenen Zeichens dargestellt werden soll. Wenn wParam 0 ist, dann werden die tatsächlich vom Benutzer getippten Zeichen ausgegeben.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

em_SetRect

Beschreibung

Diese Botschaft setzt das formatierende Rechteck für ein Steuerelement. Der Text wird neu formatiert und neu dargestellt, um dem veränderten Rechteck zu entsprechen.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf eine Rect-Datenstruktur, die die neuen Abmessungen des Rechtecks vorgibt.

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_SetRectNP

Beschreibung

Diese Botschaft setzt das formatierende Rechteck für ein Steuerelement. Diese Botschaft ähnelt der Botschaft em_SetRect, allerdings wird das Steuerelement nicht neu gezeichnet. Alle darauffolgenden Änderungen bewirken, daß das Steuerelement neu gezeichnet wird, um dem geänderten, formatierenden Rechteck zu entsprechen.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf eine Rect-Datenstruktur, die die neuen Abmessungen des Rechtecks vorgibt.

Rückgabewert

Nicht verwendet

Hinweise

Verwenden Sie diese Botschaft anstatt **em_SetRect**, wenn der Text später sowieso restauriert wird. Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_SetSel

Beschreibung

Diese Botschaft wählt alle Zeichen in einem Editierfeld aus, die innerhalb der Start- und Endzeichenpositionen liegen, die der Parameter lParam vorgibt.

Parameter

wParam	Nicht verwendet
lParamLo	Startposition
lParamHi	Endposition

Rückgabewert

Nicht verwendet

em_SetTabStops

Beschreibung

Diese Botschaft setzt die Tabulatorpositionen in einem Mehrzeilen- Editierfeld.

Parameter

wParam	1, Zahl der Tabulatorpositionen, oder 0.
lParam	Ist ein Zeiger auf das erste Element eines Arrays von Integerwerten mit mindestens wParam Elementen, die die Tabulatorpositionen in Dialogeinheiten enthalten. Die Tabulatorpositionen müssen in aufsteigender Reihenfolge sortiert werden; Rückwärtstabulatoren sind nicht erlaubt. Sind wParam und lParam gleich 0, werden voreingestellte Tabulatorpositonen im Abstand von 32 Dialogeinheiten gesetzt. Ist wParam 1, werden im Editierfeld Tabulatoren an jeder Position gesetzt, die einem Vielfachen von lParam entsprechen.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn alle Tabulatoren gesetzt wurden. Andernfalls ist der Rückgabewert 0.

Hinweise

Eine Dialogeinheit ist ein horizontaler oder vertikaler Abstand. Eine horizontale Dialogeinheit ist gleich 1/4 und eine vertikale gleich 1/8 ihrer dazugehörigen Basiseinheit. Die Dialogbasiseinheiten werden aufgrund der Höhe und Breite der aktuellen Systemschrift gesetzt. Die Funktion **GetDialogBaseUnits** gibt die aktuellen Dialogbasiseinheiten in Pixeln zurück. Diese Botschaft gilt nur für mehrzeilige Editierfelder.

em_SetWordBreak

Beschreibung

Diese Botschaft wird an Mehrzeileneditierfelder gesendet, um das Editierfeld zu informieren, daß Windows die voreingestellte Wortumbruchfunktion durch eine von der Anwendung übergebene Wortumbruchfunktion ersetzt hat.

Parameter

wParam	Nicht verwendet
lParam	Adresse der Prozedurinstanz aus **MakeProcInstance**. Die Wortumbruchfunktion sollte auf folgende Weise deklariert sein: `function WordBreakFunction(EditText: PChar; CurrentWord, EditTextCount: Integer): PChar;`
WordBreakFunction	Die Funktion kann einen anderen Namen haben; muß jedoch einen Zeiger auf das erste Zeichen des nächsten Wortes im Text liefern.
EditText	Zeiger auf den Text des Editierfelds.
CurrentWord	Index des aktuellen Wortanfangs. Ist dies das letzte Wort im Text, sollte der Zeiger auf das nächste Zeichen nach dem letzten Zeichen im Text zeigen.
EditTextCount	Gesamtzahl von Bytes im Text.

Rückgabewert

Nicht verwendet

Hinweise

Der Rückgabewert zeigt auf das erste Byte des nächsten Wortes im Text des Editierfeldes. Wenn das aktuelle Wort das letzte Wort im Text ist, dann zeigt der Rückgabewert auf das erste Byte, das dem letzten Wort folgt.

em_Undo

Beschreibung

Diese Botschaft macht den letzten Bearbeitungsvorgang im Editierfeld rückgängig.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Widerrufen-Operation erfolgreich ist. Er ist 0, wenn die Operation fehlschlägt.

Hinweise

Wenn der Benutzer das Editierfeld verändert, dann wird die letzte Änderung in einem Widerrufenpuffer gespeichert, der dynamisch mit den Anforderungen wächst. Wenn für den Puffer ungenügend Platz verfügbar ist, dann schlägt der Versuch zu widerrufen fehl und das Editierfeld bleibt unverändert.

lb_AddString

Beschreibung

Diese Botschaft fügt einen String zur Liste eines Kombinationsfensters hinzu.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf den null-terminierten String, der hinzugefügt werden soll.

Rückgabewert

Der Rückgabewert ist der Index zu String und Liste. Der Rückgabewert ist **lb_Err**, wenn ein Fehler auftritt. Er ist **lb_ErrSpace**, wenn zum Speichern des neuen Strings nicht genügend Speicher zur Verfügung steht.

Hinweise

Wenn das Listenfenster nicht sortiert ist, wird der String an das Ende der Liste angehängt.

Wenn das Listenfenster mit einer der **lbs_xxx-Konstanten** lbs_OwnerDrawFixed oder lbs_OwnerDrawVariable, jedoch ohne lbs_HasStrings definiert ist:

- dann ist lParam ein 32-Bit-Wert und wird anstelle eines String gespeichert und
- die Botschaft **wm_CompareItem** wird ein- oder mehrmals an den Besitzer des Kombinationsfensters gesendet, so daß der neue Eintrag richtig in der Liste angeordnet werden kann.

lb_DeleteString

Beschreibung

Diese Botschaft löscht einen String aus der Liste.

Parameter

wParam	Enthält einen Index zu dem String, der gelöscht werden soll.
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Anzahl der Strings, die in der Liste verbleiben. Der Rückgabewert ist **lb_Err**, wenn wParam keinen gültigen Index bezeichnet.

Hinweise

Wenn das Kombinationsfenster besitzergezeichnet ist (siehe **lbs_xxx-Konstanten**) ist, aber ohne lbs_HasStrings erstellt wurde, dann wird eine Botschaft **wm_DeleteItem** an den Besitzer des Kombinationsfensters gesendet, so daß die Anwendung zusätzliche, dem Eintrag zugeordnete Daten freigeben kann.

lb_Dir

Beschreibung

Diese Botschaft fügt eine Liste der Dateien aus dem aktuellen Verzeichnis zur Liste hinzu. Es werden nur Dateien mit den Attributen hinzugefügt, die der Parameter wParam vorgibt und die der vom Parameter lParam gegebenen Dateispezifikation entsprechen.

Parameter

wParam	DOS-Attribut
lParam	Zeigt auf einen Dateispezifikationsstring.

Rückgabewert

Es wird der Index des letzten dargestellten Eintrags zurückgegeben. Der Rückgabewert ist **lb_Err**, wenn ein Fehler auftritt; oder **lb_ErrSpace**, wenn zum Speichern des neuen Strings nicht genügend Speicherplatz vorhanden ist.

lb_FindString

Beschreibung

Diese Botschaft findet den ersten String in der Liste eines Kombinationsfensters, der mit dem angegebenen Präfixtext übereinstimmt.

Parameter

wParam	Enthält den Index zu dem Eintrag vor dem ersten zu suchenden Eintrag. Wenn die Suche das Ende einer Liste erreicht, dann wird sie vom Listenbeginn an bis zu dem Element, das wParam bezeichnet, fortgesetzt. Wenn der Parameter wParam -1 ist, dann wird die ganze Liste von Beginn an durchsucht.
lParam	Zeigt auf den null-terminierten Präfixstring.

Rückgabewert

Der Rückgabewert ist der Index des übereinstimmenden Eintrags oder **lb_Err**, wenn die Suche nicht erfolgreich war.

Hinweise

Wenn das Kombinationsfenster besitzergezeichnet (siehe **lbs_xxx-Konstanten**) ist, aber ohne lbs_HasStrings erstellt wurde, dann gibt diese Botschaft den Index des Eintrags zurück, dessen LongInt-Wert mit dem Wert übereinstimmt, der als Parameter lParam von lbs_FindString übergeben wurde.

lb_GetCount

Beschreibung

Diese Botschaft gibt die Anzahl der Einträge in der Liste eines Kombinationsfensters zurück.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Anzahl der Einträge in der Liste eines Kombinationsfensters.

lb_GetCurSel

Beschreibung

Diese Botschaft gibt den Index des aktuell ausgewählten Eintrags in der Liste eines Kombinationsfensters an, falls ein solcher vorhanden ist.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist der Index des aktuell ausgewählten Eintrags. Er ist **lb_Err**, wenn kein Eintrag ausgewählt ist.

lb_GetHorizontalExtent

Beschreibung

Diese Botschaft ruft die Breite in Pixeln ab, um die in der Liste ein Bildlauf durchgeführt werden kann, wenn die Liste horizontale Bildlaufleisten hat.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Breite des Bildlaufbereichs in der Liste in Pixeln.

Hinweise

Die Liste muß mit dem Stil **ws_HScroll** definiert worden sein.

lb_GetItemData

Beschreibung

Diese Botschaft ruft den von der Anwendung bereitgestellten 32-Bit-Wert ab, der dem angegebenen Listeneintrag zugeordnet ist.

Parameter

wParam	Index zum Element
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist der 32-Bit-Wert, der dem Eintrag zugeordnet ist oder **lb_Err**, wenn ein Fehler auftritt.

lb_GetItemRect

Beschreibung

Diese Botschaft ruft die Abmessungen des Rechtecks ab, das einen Listeneintrag so umgrenzt, wie er aktuell im Listenfenster dargestellt wird.

Parameter

wParam	Enthält einen Index zu dem Element.
lParam	Enthält einen Zeiger auf eine Rect-Datenstruktur, die die Client-Koordinaten des Listeneintrags aufnimmt.

Rückgabewert

lb_Err im Fehlerfall.

lb_GetSel

Beschreibung

Diese Botschaft gibt den Auswahlstatus eines Eintrags zurück.

Parameter

wParam	Der Index zum Eintrag
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist eine positive Zahl, wenn ein Eintrag ausgewählt ist, andernfalls ist er 0. Der Rückgabewert ist **lb_Err**, wenn ein Fehler auftritt.

lb_GetSelCount

Beschreibung

Diese Botschaft gibt die Gesamtzahl ausgewählter Einträge in einer Liste mit Mehrfachauswahlmöglichkeit an.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Anzahl der ausgewählten Einträge in einer Liste. Wenn die Liste nur für Einzelauswahl vorgesehen ist, ist der Rückgabewert **lb_Err**.

lb_GetSelItems

Beschreibung

Diese Botschaft füllt einen Puffer mit einem Array von Integerwerten, die Indices von ausgewählten Einträgen in einer Liste mit Mehrfachauswahl angeben.

Parameter

wParam	Gibt die maximale Anzahl von ausgewählten Einträgen an, deren Indexeinträge im Puffer gespeichert werden sollen.
lParam	Enthält einen leiger auf einen Puffer, der für die Anzahl von Integerwerten, die der Parameter wParam angibt, ausreichend groß ist.

Rückgabewert

Der Rückgabewert ist die Anzahl der tatsächlich im Puffer gespeicherten Einträge. Wenn die Liste nur für Einzelauswahlen vorgesehen ist, ist der Rückgabewert **lb_Err**.

lb_GetText

Beschreibung

Diese Botschaft kopiert einen String aus der Liste in einen Puffer.

Parameter

wParam	Der Index zum Eintrag
lParam	Zeigt auf den Puffer, der den String empfangen soll. Der Puffer muß groß genug sein, um sowohl den String, als auch dessen abschließendes Null-Zeichen aufnehmen zu können.

Rückgabewert

Nicht verwendet

Hinweise

Wenn die Liste besitzergezeichnet ist (siehe **lbs_xxx-Konstanten**), aber ohne den Stil lbs_HasStrings erstellt wurde, empfängt der Puffer, auf den der Parameter lParam der Botschaft zeigt, den 32-Bit-Wert, der dem Eintrag zugeordnet ist.

lb_GetTextLen

Beschreibung

Diese Botschaft gibt die Länge eines Strings in der Liste zurück.

Parameter

wParam	Der Index zum Eintrag
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist die Länge des Strings (in Bytes) ohne Berücksichtigung des abschließenden Null-Zeichens. Der Rückgabewert ist **lb_Err**, wenn ein Fehler auftritt.

lb_GetTopIndex

Beschreibung

Diese Botschaft gibt den Index des ersten sichtbaren Eintrags in einer Liste zurück.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Index des ersten sichtbaren Eintrags in einer Liste.

Hinweise

Zu Beginn ist der erste sichtbare Eintrag 0. Wurde das Listenfenster verschoben, kann ein anderer Eintrag an oberster Stelle stehen.

lb_InsertString

Beschreibung

Diese Botschaft fügt einen String in eine Liste ein. Es wird keine Sortierung vorgenommen.

Parameter

wParam	Enthält einen Index zu der Position, die den String aufnehmen soll. Wenn der Parameter wParam -1 ist, wird der String an das Ende der Liste angefügt.
lParam	Zeigt auf den null-terminierten String, der eingefügt werden soll.

Rückgabewert

Der Rückgabewert ist der Index der Position, an der der String eingefügt wurde. Es wird **lb_Err** zurückgegeben, wenn ein Fehler auftritt, und **lb_ErrSpace**, wenn zum Abspeichern des neuen Strings nicht genügend Speicherplatz zur Verfügung steht.

lb_ResetContent

Beschreibung

Diese Botschaft löscht alle Strings aus einer Liste und gibt den gesamten für diese Strings reservierten Speicher frei.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Wenn die Liste besitzergezeichnet ist, aber ohne den Stil lbs_HasStrings erstellt wurde, empfängt der Besitzer der Liste eine Botschaft **wm_DeleteItem** für jedes Element in der Liste.

lb_SelectString

Beschreibung

Diese Botschaft setzt die aktuelle Auswahl auf den ersten String, der das angegebene Präfix hat.

Parameter

wParam	Enthält den Index des Eintrags vor dem ersten zu suchenden Eintrag. Wenn die Suche das Listenende erreicht, wird sie vom Listenbeginn an bis hin zu dem von lParam bezeichneten Eintrag fortgesetzt. Wenn der Parameter wParam -1 ist, wird die gesamte Liste von Beginn an abgesucht.
lParam	Zeigt auf den null-terminierten Präfix-String.

Rückgabewert

Der Rückgabewert ist der Index des ausgewählten Eintrags oder **lb_Err** , wenn ein Fehler auftritt.

Hinweise

Diese Botschaft darf nicht für Listen verwendet werden, die für Mehrfachauswahl vorgesehen sind.Ein String wird nur ausgewählt, wenn seine Anfangsbuchstaben den Zeichen im Präfix-String entsprechen. Wenn die Liste besitzergezeichnet ist (über die **lbs_xxx-Konstanten** lbs_OwnerDrawFixed oder lbs_OwnerDrawVariable verfügt), aber ohne den Stil lbs_HasStrings erstellt wurde, gibt diese Botschaft den Index des Eintrags zurück, dessen LongInt-Wert dem als Parameter lParam übergebenen Wert entspricht.

lb_SelItemRange

Beschreibung

Diese Botschaft wählt einen oder mehrere aufeinanderfolgende Einträge in einer Mehrfachauswahlliste aus oder hebt die Auswahl auf.

Parameter

wParam	Gibt an, wie die Auswahl gesetzt werden soll. Wenn der Parameter wParam ungleich 0 ist, wird der String ausgewählt und hervorgehoben; ist wParam 0, wird die Hervorhebung zurückgenommen und der String ist nicht mehr ausgewählt.
lParamLo	Index auf den ersten auszuwählenden Eintrag
lParamHi	Index auf den letzten auszuwählenden Eintrag

Rückgabewert

lb_Err im Fehlerfall.

Hinweise

Diese Botschaft sollte nur in Mehrfachauswahlfenstern benutzt werden.

lb_SetColumnWidth

Beschreibung

Diese Botschaft wird an eine Mehrspaltenliste gesendet, um die Breite der Spalten in der Liste (in Pixeln) zu setzen.

Parameter

wParam	Gibt die Breite aller Spalten in Pixeln an.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft betrifft nur Listenfenster mit dem Stil **lbs_MultiColumn**.

lb_SetCurSel

Beschreibung

Diese Botschaft wählt einen String aus und läßt ihn ins Blickfeld rollen, falls dies notwendig ist. Wenn der neue String ausgewählt ist, löscht die Liste die Hervorhebung des vorher ausgewählten Strings.

Parameter

wParam	Der Index zum Eintrag; Wenn wParam -1 ist, wird der Liste keine Auswahlmöglichkeit zugeteilt.
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist der Index auf den ausgewählen Eintrag, falls wParam -1 oder kein gültiger Index ist, wird **lb_Err** zurückgegeben.

lb_SetHorizontalExtent

Beschreibung

Diese Botschaft stellt die Breite in Pixeln ein, um die ein horizontaler Bildlauf durchgeführt werden kann.

Parameter

wParam	Gibt die Anzahl der Pixel an, um die die Liste verschoben werden kann.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Liste muß mit dem Stil **ws_HScroll** erstellt worden sein.
Ist die Liste größer oder gleich dem resultierenden Wert, wird die horizontale Bildlaufleiste deaktiviert.

lb_SetItemData

Beschreibung

Diese Botschaft setzt einen 32-Bit-Wert, der dem angegebenen Eintrag in einer Liste zugeordnet ist.

Parameter

wParam	Der Index zum Eintrag
lParam	Enthält den neuen Wert, der dem Eintrag zugeordnet werden soll.

Rückgabewert

lb_Err im Fehlerfall.

lb_SetSel

Beschreibung

Diese Botschaft wählt in einer Liste mit Mehrfachauswahl einen String aus.

Parameter

wParam	Gibt an, wie die Auswahl gesetzt werden soll. Wenn der Parameter wParam ungleich 0 ist, wird der String ausgewählt und hervorgehoben; ist der Parameter wParam 0, wird die Hervorhebung zurückgenommen und der String ist nicht mehr ausgewählt.
lParam	Das niederwertige Word des Parameters lParam ist ein Index, der angibt, welcher String gesetzt werden soll. Wenn lParam -1 ist, wird die Auswahl von allen Strings gelöscht oder zu allen hinzugefügt, abhängig vom Wert von wParam.

Rückgabewert

lb_Err im Fehlerfall.

Hinweise

Diese Botschaft sollte nur für Mehrfachauswahllisten verwendet werden.

lb_SetTabStops

Beschreibung

Diese Botschaft setzt die Tabulatoren in einer Liste.

Parameter

wParam	Ein Integer, der die Anzahl der Tabulatoren in der Liste angibt.
lParam	Wenn wParam und lParam 0 sind, beträgt die voreingestellte Tabulatorweite zwei Dialogeinheiten. Wenn wParam 1 ist, verfügt das Editierfeld über Tabulatoren mit dem von lParam bezeichneten Abstand. Wenn wParam weder 0 noch 1 ist, ist lParam ein Zeiger auf das erste Element eines Arrays von Integerwerten mit mindestens wParam Elementen, die die Tabulatorpositionen in Dialogeinheiten enthalten.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn alle Tabulatoren gesetzt wurden, ansonsten 0.

Hinweise

Eine Dialogeinheit ist ein horizontaler oder vertikaler Abstand. Eine horizontale Dialogeinheit ist gleich einem Viertel der zugehörigen Basiseinheit. Bei der Berechnung der Basisdialogeinheiten werden Breite und Höhe der aktuellen Systemschrift zugrundegelegt. Die Funktion **GetDialogBaseUnits** gibt die aktuellen Dialogbasiseinheiten in Pixeln zurück. Die Tabulatoren müssen in aufsteigender Ordnung sortiert werden, Rückwärtstabulatoren sind nicht zulässig.

Diese Botschaft sollte nur für Mehrfachauswahllisten verwendet werden.

lb_SetTopIndex

Beschreibung

Diese Botschaft macht das durch den Index bezeichnete Element zum ersten sichtbaren Eintrag in einer Liste.

Parameter

wParam	Der Index zum Eintrag
lParam	Nicht verwendet

Rückgabewert

lb_Err im Fehlerfall.

wm_Activate

Beschreibung

Diese Botschaft wird gesendet, wenn ein Fenster aktiviert oder deaktiviert wird.

Parameter

wParam	0, wenn das Fenster nicht aktiv ist. i 1, Das Fenster wurde nicht durch einen Mausklick aktiviert i 2, Das Fenster wurde durch einen Mausklick des Anwenders aktiviert.
lParamHi	lParam ist ungleich 0, wenn das Fenster verkleinert ist und in jedem anderen Fall 0
lParamLo	Hat wParam den Wert 0, ist das niederwertige Word des Parameters lParamLo das Handle des aktivierten Fensters. Ist wParam ungleich 0, enthält das niederwertige Word von lParamLo das Handle des soeben deaktivierten Fensters.

Rücgabewert

Nicht verwendet

Hinweise

Wenn ein Fenster nicht zum Symbol verkleinert ist und aktiviert wird, ordnet die Funktion **DefWindowProc** als Standardaktion dem Fenster den Eingabefokus zu.

wm_ActivateApp

Beschreibung

Diese Botschaft wird übergeben, wenn das zu aktivierende Fenster einer anderen Anwendung angehört als das aktive Fenster. Die Botschaft wird sowohl an die Anwendung des zu aktivierenden Fensters, als auch an die Anwendung des deswegen zu deaktivierenden Fensters übergeben.

Parameter

wParam	Der Wert 0 zeigt an, daß Windows ein Fenster deaktivieren wird. Andere Werte bedeuten, daß ein Fenster aktiviert wird.
lParam	Enthält das Task-Handle der Anwendung.

Rückgabewert

Nicht verwendet

wm_AskCBFormatName

Beschreibung

Fragt den Besitzer der Zwischenablage nach dem Format der Zwischenablagedaten.

Parameter

wParam	Bestimmt die maximale Anzahl der zu kopierenden Bytes.
lParam	Zeigt auf den Puffer, der die Kopie des Formatnamens aufnehmen soll.

Rückgabewert

Nicht verwendet

Hinweise

Folgende Botschaften werden übergeben, wenn die Zwischenablage ein Daten-Handle für das Format **cf_OwnerDisplay** enthält:

wm_AskCBFormatName
wm_HScrollClipboard
wm_PaintClipboard
wm_SizeClipboard
wm_VScrollClipboard

Das Format der Zwischenablagedaten wird durch die Funktion **SetClipboardData** gesetzt.

wm_CancelMode

Beschreibung

Diese Botschaft annuliert jeden Systemmodus, z.B. den, der die Maus innerhalb einer Bildlaufleiste verfolgt oder der ein Fenster bewegt.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft warnt das Programm, daß Maus- und Tastatureingaben in das Botschaftsfenster geleitet werden.

Prozesse, die den Tastatur- und Mausstatus und/oder die Mausposition überwachen, können nach dem Löschen des Botschaftsfensters ungültig geworden sein.

wm_ChangeCBChain

Beschreibung

Diese Botschaft teilt dem ersten Fenster in der Viewer-Kette der Zwischenablage mit, daß ein Fenster aus der Kette entfernt wurde.

Parameter

wParam	Enthält das Handle des aus der Viewer-Kette der Zwischenablage zu entfernenden Fensters.
lParamLo	Enthält im niederwertigen Word das Handle des Fensters, das dem zu entfernenden Fenster nachfolgt.
lParamHi	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Jedes Fenster, das diese Botschaft empfängt, sollte die Funktion **SendMessage** aufrufen, um die Botschaft an das folgende Fenster in der Viewer-Kette der Zwischenablage weiterzugeben.

Das Handle des nächsten Fensters der Kette ist zuerst der Rückgabewert der Funktion **SetClipboardViewer**, die das Fenster einfügt.

Das Fenster muß sich selbst aus dieser Kette vor der Rückkehr von der Botschaft **wm_Destroy** entfernen.

Siehe auch:

wm_DrawClipboard

wm_Char, wm_DeadChar, wm_KeyDown und wm_KeyUpen

Beschreibung

wm_Char enthält den Wert der gedrückten oder losgelassenen Taste.
wm_DeadChar bestimmt den Wert einer toten Taste.
wm_KeyDown enthält den Wert der gedrückten Taste
wm_KeyUp enthält den Wert der losgelassenen Taste

Parameter

wParam	Enthält den Wert der Taste.
lParamLo	Wiederholungszähler (dessen Wert ergibt sich aus der Zeit, während der der Anwender die Taste gedrückt hält).
lParamHi	Bits 0-7 sind der Scancode der Taste, OEM-abhängig. Bit 8 ist 1, wenn die Taste eine erweiterte ist Bit 13 ist 1, wenn die Taste Alt zugleich gedrückt wurde Bit 14 ist 1, wenn die Taste vor dieser Botschaft gedrückt wurde Bit 15 ist 1, wenn die Taste losgelassen wird, 0 wenn sie gedrückt wird.

Rückgabewert

Nicht verwendet

Hinweise

Eine Nicht-Systemtaste ist jede Taste, die ohne Drücken der Taste Alt gedrückt wird.

Tote Tasten sind Umlaute und Akzentzeichen. wm_DeadChar kann für RückBotschaften von Tasten verwendet werden, die nicht automatisch ein Zeichen ergeben.

Aufgrund der Tastenwiederholfunktion kann mehr als eine wm_KeyDown vor einer wm_KeyUp Botschaft gesendet werden.

Bei wm_KeyDown	lParamHi Bit 13 = 0 lParamHi Bit 15 = 0
Bei wm_KeyUp	lParamHi Bit 13 = 0 lParamHi Bit 15 = 1

Im allgemeinen werden nur lParamLo und die Bits 0-7 von lParamHi von Anwendungen benötigt.

Hat kein Fenster den Fokus, werden die Botschaften **wm_SysKeyDown**, **wm_SysChar** und **wm_SysKeyUp** anstatt wm_KeyDown, wm_Char und wm_KeyUp gesendet.

Siehe auch:

wm_SysDeadChar

wm_CharToItem

Beschreibung

Diese Botschaft wird von einer Liste des Typs lbs_WantKeyBoradInput an ihren Besitzer als Reaktion auf eine **wm_Char** -Botschaft übergeben.

Parameter

wParam	Enthält den Wert der vom Anwender gedrückten Taste.
lParamLo	Enthält das Handle der Liste im niederwertigen Word.
lParamHi	Enthält die aktuelle Position des Carets.

Rückgabewert

-2 zeigt an, daß die Anwendung die Auswahl eines Eintrags vollkommen bewältigt hat
-1 zeigt an, daß die Liste die dem Tastendruck entsprechende Aktion ausführen soll.
>=0 bestimmt den Index des Listeneintrags und zeigt an, daß die Liste die dem Tastendruck und dem vorgegebenen Eintrag entsprechende Standardaktion ausführen soll.

Hinweise

Diese Botschaft betrifft nur Listen mit dem Stil **lbs_WantKeyboardInput**.

Siehe auch:

wm_VKeyToItem

wm_ChildActivate

Beschreibung

Diese Botschaft wird an ein übergeordnetes Fenster übergeben, wenn die Funktion **SetWindowPos** das untergeordnete Fenster verschiebt.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_Clear

Beschreibung

Diese Botschaft löscht die aktuelle Auswahl.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_Close

Beschreibung

Diese Botschaft tritt beim Schließen eines Fensters auf.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Funktion **DefWindowProc** ruft die Funktion **DestroyWindow** auf, um das Fenster zu schließen.

wm_Command

Beschreibung

Diese Botschaft tritt auf, wenn der Anwender einen Menüeintrag auswählt, ein Steuerelement eine Botschaft an ihr übergeordnetes Fenster übergibt oder ein Akzeleratortastendruck übersetzt wird.

Parameter

wParam	Enthält den Menüeintrag, die ID eines Steuerelements oder die der Accelerator-Taste.
lParamLo	Ist 0, wenn die Botschaft von einem Menü stammt , lParamHi wird in diesem Fall nicht verwendet Ansonsten hängt die Interpretation von lParamLo von lParamHi ab.
lParamHi	Geht die Botschaft von einer Accelerator-Taste aus, enthält das höherwertige Word von lParamLo 1. Stammt die Botschaft von einem Steuerelement, enthält das höherwertige Word den Benachrichtigungscode. Das niederwertige Word von lParamLo enthält dann das Fenster-Handle des die Botschaft sendenden Steuerelements.

Rückgabewert

Nicht verwendet

Hinweise

Eine Accelerator-Taste, die für die Auswahl eines Eintrags aus dem Systemmenü definiert ist, wird in **wm_SysCommand** -Botschaften übersetzt, anstatt in wm_Command.

Wird ein Accelerator-Tastendruck durchgeführt, der einem Menüeintrag entspricht und ist das verwaltende Fenster des Menüs verkleinert, wird keine wm_Command-Botschaft übergeben. Wenn aber der Accelerator-Tastendruck keinem Menüeintrag des Fensters oder des Sytemmenüs entspricht, wird auch wenn das betreffende Fenster verkleinert ist, eine wm_Command-Botschaft übergeben.

Siehe auch:

bn_xxx-Konstanten
cbn_xxx-Konstanten
en_xxx-Konstanten
lbn_xxx-Konstanten

wm_Compacting

Beschreibung

Diese Botschaft wird an alle Hauptfenster übergeben, sobald Windows feststellt, daß mehr als 12,5 Prozent der Systemzeit innerhalb eines 30 oder 60 Sekunden währenden Intervalls zur Neuanordnung des Speichers verwendet wird. In diesem Fall reicht der Sytemspeicher nicht aus.

Parameter

wParam	65535 Mal ein Prozent der CPU-Zeit, die für die Speicherverwaltung aufgewendet wird.(ist z.B. wParam = 32768, entspricht 50% der CPU-Zeit).
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Erhält eine Anwendung diese Botschaft, sollte sie möglichst viel Speicher bereitstellen und dabei die eigenen Anforderungen sowie die Anzahl der unter Windows laufenden Anwendungen berücksichtigen.

Diese Anzahl kann über die Funktion **GetNumTasks** ermittelt werden.

wm_CompareItem

Beschreibung

Diese Botschaft bestimmt die relative Position eines neuen Eintrags in einem sortierten, besitzergezeichneten Kombinationsfenster oder einer Liste.

Parameter

wParam	Nicht verwendet
lParam	Der Parameter lParam ist ein Zeiger vom Typ long auf eine **TCompareItemStruct**-Datenstruktur, die Bezeichner und anwendungsunterstützte Daten zweier Einträge des Kombinationsfensters oder der Liste enthält.

Rückgabewert

-1: Eintrag 1 wird vor Eintrag 2 eingeordnet.
0:Eintrag 1 und 2 werden gleich eingeordnet.
1:Eintrag 1 wird nach Eintrag 2 eingeordnet.

Hinweise

Diese Botschaft gilt nur für folgende Stile:

die **cbs_xxx-Konstanten:**
cbs_Sort, cbs_OwnerDrawFixed und cbs_OwnerDrawVariable
die **lbs_xxx-Konstanten**:
lbs_Sort,, lbs_OwnerDrawFixed und lbs_OwnerDrawVariable

wm_Copy

Beschreibung

Diese Botschaft kopiert den aktuellen Auswahlbereich im **cf_Text** -Format in die Zwischenablage.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_Create

Beschreibung

Diese Botschaft teilt der Fensterprozedur mit, daß eine Initialisierung erfolgen sollte.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf eine **TCreateStruct**-Datenstruktur, in der eine Kopie der an die Funktion **CreateWindow** übergebenen Parameter abgelegt ist.

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft wird beim Aufruf der Funktion CreateWindow vor dem Öffnen des Fensters gesendet.

wm_CtlColor

Beschreibung

Diese Botschaft wird an das übergeordnete Fenster übergeben, wenn ein untergeordnetes Steuerelement oder ein Mitteilungsfenster gezeichnet werden soll. Dies ermöglicht es dem übergeordneten Fenster, Text- und Hintergrundfarben des untergeordneten Fensters unter Verwendung des im Parameter wParam vorgegebenen Handle des Bildschirmkontextes vorzugeben.

Parameter

wParam	Enthält das Handle des Bildschirmkontextes für das untergeordnete Fenster.
lParamLo	Enthält das Handle des untergeordneten Fensters.
lParamHi	Eine **ctlcolor_xxx-Konstante**; legt den Typ des Steuerelements fest.

Rückgabewert

Nicht verwendet

Hinweise

Die Funktion **DefWindowProc** wählt die Standardsystemfarben.

wm_Cut

Beschreibung

Diese Botschaft übergibt die aktuelle Auswahl für die Zwischenablage im Format **cf_Text** und löscht die Auswahl anschließend aus dem Steuerelementfenster.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_dde_Ack

Beschreibung

Informiert das Programm, daß eine andere DDE-Botschaft empfangen wurde.

Parameter

wParam	Handle des sendenden Fensters
lParamLo	Ist die Botschaft **wm_dde_Initiate**, enthält lParamLo ein Atom mit dem antwortenden Programm Ist die Botschaft **wm_dde_Execute**, enthält lParamLo einen Record mit dem Antwortstatus Bei allen anderen Botschaften enthält lParamLo einen Statusrecord.
lParamHi	Ist die Botschaft wm_dde_Initiate, enthält lParamHi ein Atom mit dem Thema der Übertragung Ist die Botschaft wm_dde_Execute, enthält lParamHi ein Handle zum Dateneintrag, das den Befehlsstring enthält. Bei allen anderen Botschaften enthält lParamHi ein Atom mit dem Dateneintrag, für das die Antwort gesendet wurde

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **SendMessage** gesendet werden.

Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Advise

Beschreibung

Wird von einem Client-Programm gesendet und fordert von der Server-Anwendung eine Aktualisierung, wenn sich der Dateneintrag ändert.

Parameter

wParam	Handle des sendenden Fensters
lParamLo	Eine **TDDEAdvise**-Datenstruktur, der angibt, wie die Daten gesendet werden sollen.
lParamHi	Ein Atom mit dem angefordeten Dateneintrag.

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **PostMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Data

Beschreibung

Wird von der Server-Anwendung übergeben, um den Wert eines Dateneintrags zu übertragen oder um der Client-Anwendung die Verfügbarkeit des Eintrags anzuzeigen.

Parameter

Param	Handle des sendenden Fensters.
lParamLo	Handle zum globalen Speicherblock, der die Daten in einer **TDDEData**-Datenstruktur enthält, oder 0, wenn eine Änderung angezeigt wird.
lParamHi	Ein Atom, das festlegt, welcher Dateneintrag übergeben wird.

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **PostMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Execute

Beschreibung

Wird von einer Client-Anwendung gesendet, um Befehlsfolge zu übertragen, die von der Server-Anwendung bearbeitet werden soll.

Parameter

wParam	Handle des sendenden Fensters
lParamLo	Reserviert
lParamHi	Handle auf ein globales Speicherobjekt, das die Befehlsfolge enthält.

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **PostMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Initiate

Beschreibung

Wird von Client- oder Server-Anwendungen zur Initialisierung eines Austauschs gesendet.
Antwortende Programme müssen die Botschaft **wm_dde_Ack** senden.

Parameter

wParam	Handle des sendenden Fensters
lParamLo	Ein Atom mit Namen des anfragenden Programms 0 für Austausch mit beliebigem Programm
lParamHi	Ein Atom mit dem Thema, über das Austausch erfolgen soll 0 für beliebiges Thema

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **SendMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Poke

Beschreibung

Wird von Client-Anwendungen an die Server-Anwendung gesendet, um zum Empfang unerwünschter Daten aufzufordern. Die Server-Anwendung antwortet mit **wm_dde_Ack**.

Parameter

wParam	Handle des sendenden Fensters
lParamLo	Handle zu einer **TDDEPoke**-Datenstruktur
lParamHi	Atom mit dem Dateneintrag

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **PostMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Request

Beschreibung

Wird von der Client-Anwendung gesendet, um den Wert eines bestimmten Dateneintrags zu erfragen.

Parameter

wParam	Handle des sendenden Fensters
lParamLo	Zwischenablagenformat-Nummer
lParamHi	Atom mit dem Dateneintrag

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **PostMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Terminate

Beschreibung

Wird von einer Anwendung zur Beendigung eines Austauschs gesendet.

Parameter

wParam	Handle des sendenden Fensters
lParam	Reserviert

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **PostMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_dde_Unadvise

Beschreibung

Wird von der Client-Anwendung an die Server-Anwendung gesendet, um anzuzeigen, daß die Aktualisierung eines bestimmten Dateneintrags oder Zwischenablageformats dieses Eintrags nicht länger nötig ist.

Parameter

wParam	Handle des sendenden Fensters
lParamLo	Zwischenablageformat-Nummer
lParamHi	Atom mit dem Dateneintrag

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **PostMessage** gesendet werden.
Der erste Parameter sollte das Handle des Fensters sein, das die Botschaft erhält.

wm_DeleteItem

Beschreibung

Zeigt an, daß ein Eintrag in einer besitzergezeichneten Liste oder einem Kombinationsfenster gelöscht wurde.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft gilt nur für

- Kombinationsfenster mit den **cbs_xxx-Konstanten** cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable, und
- Listenfenstern mit den **lbs_xxx-Konstanten** lbs_OwnerDrawFixed oder lbs_OwnerDrawVariable.

Diese Botschaft wird gesendet, wenn das Kombinationsfenster oder die Liste gelöscht oder der Eintrag mit einer der folgenden Botschaften gelöscht wird.

lb_DeleteString
lb_ResetContent
cb_DeleteString
cb_ResetContent

wm_Destroy

Beschreibung

Diese Botschaft meldet einem Fenster, daß es entfent wird.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Jedes Fenster der Zwischenablage-Kette muß sich mit der Funktion **ChangeClipboardChain** vor der Rückkehr von der Botschaft **wm_Destroy** aus ihr entfernen.

Diese Botschaft wird von der Funktion **DestroyWindow** übergeben, nachdem ein Fenster vom Bildschirm entfernt wurde. Ein Fenster erhält diese Botschaft, bevor ein untergeordnetes Fenster entfernt wird.

wm_DestroyClipboard

Beschreibung

Benachrichtigt den Besetzer der Zwischenablage, daß diese durch Funktionsaufruf von **EmptyClipboard** geleert wurde.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_DevModeChange

Beschreibung

Wird an alle Hauptfenster übergeben, wenn der Anwender Geräteeinstellungen ändert.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf den Gerätenamen

Rückgabewert

Nicht verwendet

Hinweise

Der Gerätename ist der String aus der Windows-Initialisierungsdatei WIN.INI.

wm_DrawClipboard

Beschreibung

Signalisiert einer Anwendung, daß die nächste Anwendung in der Kette über die Veränderungen des Inhalts der Zwischenablage benachrichtigt werden muß.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft muß mit der Funktion **SendMessage** an das nächste Fenster der Kette gesendet werden.

Das Handle des nächsten Fensters der Kette ist der Wert, der beim Einfügen eines Fensters mit **SetClipboardViewer** zurückgegeben wird. Neue Werte für dieses Handle werden mit **wm_ChangeCBChain** Botschaften gesendet.

Ein Fenster muß sich aus der Kette entfernen, wenn es die Botschaft **wm_Destroy** erhält.

wm_DrawItem

Beschreibung

Zeigt an, daß ein besitzergezeichnetes Steuerelement aktualisiert werden muß.

Parameter

wParam	Nicht verwendet
lParam	Zieger auf eine **TDrawItemStruct**-Datenstruktur, die Informationen über das Element und die nötige Zeichenoperationenthält.

Rückgabewert

Nicht verwendet

Hinweise

Alle Objekte zur Darstellung aus der Struktur TDrawItemStruct müssen vor Rückgabe von dieser Botschaft restauriert werden.

wm_Enable

Beschreibung

Wird übergeben, nachdem ein Fenster aktiviert oder deaktiviert wurde.

Parameter

wParam	0, Fenster wurde deaktiviert Ansonsten, Fenster wurde aktiviert
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_EndSession

Beschreibung

Diese Botschaft teilt einer Anwendung, die auf eine **wm_QueryEndSession**-Botschaft ungleich 0 zurückgegeben hat, mit, ob die Arbeitssitzung tatsächlich beendet wird.

Parameter

wParam	0: die Sitzung wird nicht beendet ungleich 0: die Sitzung wird beendet.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Ist wParam ungleich 0, kann Windows die Arbeitssitzung jederzeit nach Bearbeitung dieser Botschaft durch die Anwendungen beenden. Daher sollte eine Anwendung alle für die Beendigung notwendigen Aufgaben durchführen, bevor sie von der Bearbeitung dieser Botschaft zurückkehrt.
Ein Aufruf der Funktionen **DestroyWindow** und **PostQuitMessage** ist hier, im Unterschied zur Botschaft **wm_Destroy**. nicht nötig.

wm_EnterIdle

Beschreibung

Informiert ein Fenster, daß ein Dialogfenster oder Menü dargestellt wurde und auf Eingaben des Anwenders wartet.

Parameter

wParam	Wartet das System mit einem Dialogfenster, ist wParam gleich msgf_DialogBox; Wartet das System mit einem Menü, ist wParam gleich msgf_Menu.
lParamLo	Handle zu Dialogfenster oder Menü, wenn wParam msgf_DialogBox oder msgf_Menu ist.
lParamHi	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Das System liegt still, weil ein modales Dialogfenster oder ein dargestelltes Menü keine Botschaften in der Warteschlange hat, nachdem mindestens eine vorhergehende Botschaft verarbeitet wurde. Der Standardrückgabewert von **DefWindowProc** ist 0.

Siehe auch:

msgf_xxx-Konstanten

wm_EraseBkgnd

Beschreibung

Wird übergeben, wenn der Fensterhintergrund gelöscht werden muß.

Parameter

wParam	Handle zum Gerätekontext
lParam	Nicht verwendet

Rückgabewert

Ungleich 0, wenn ein Programm diese Botschaft bearbeitet und den Hintergrund löscht, sonst 0.

Hinweise

Die Vorgabeaktion der Funktion **DefWindowProc** ist Löschen des Hintergrunds mit dem Klassen-Hintergrund der Klassen-Struktur.
Ist das Zeichenwerkzeug das Klassen-Hintergrunds 0, sollte das Programm

- den Ausgangspunkt des Pinsels mit der Funktion **UnrealizeObject** ausrichten.
- einen Pinsel wählen
- den Hintergrund mit dem Pinsel löschen.

Windows nimmt den Abbildungsmodus **mm_Text** an. Wenn der Gerätekontext einen anderen Abbildungsmodus verwendet, kann der gelöschte Bereich die sichtbare Fläche des Client-Bereichs überschreiten.

wm_FontChange

Beschreibung

Wird Hauptfenstern übergeben, wenn sich der Bestand an Schriftressourcen ändert.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Jede Anwendung, die Schriften dem System zufügt oder aus ihm entfernt, sollte diese Botschaft mit der Funktion **SendMessage** an das Fenster der obersten Ebene senden.
Die Funktion **AddFontResource** fügt Schriften ins System ein.
Die Funktion **RemoveFontResource** entfernt Schriften aus dem System.

wm_GetDlgCode

Beschreibung

Wird an die zu einem Steuerelement gehörende Eingabeprozedur übergeben und ermöglicht einer Anwendung die Verwaltung aller Eingaben der Cursor-Tasten und der Tab-Taste.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

dlgc_xxx-Konstanten werden bitweise verglichen, je nach der Eingabe, die das Programm verarbeiten will.

Hinweise

Der Standardrückgabewert von **DefWindowProc** ist 0.

Die Fensterfunktionen für vordefinierte Steuerelement-Klassen können einen Code, der ungleich 0 ist, zurückgeben.

Diese Botschaft und die nicht vorgegebenen Rückgabewerte sind nur für anwenderdefinierte Dialogelemente oder unterklassifizierte Standardsteuerelemente sinnvoll.

wm_GetFont

Beschreibung

Gibt die aktuelle Schrift eines Dialogfensters zurück.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

0, wenn das Dialogfenster die Systemschrift verwendet, ansonsten ein Handle zur Schrift.

Siehe auch:

wm_SetFont

wm_GetMinMaxInfo

Beschreibung

Ruft die maximale Größe eines Fenster, die minimale und maximale Tracking-Größe und die Position des vergrößerten Fensters ab.

Parameter

wParam	Nicht verwendet
lParam	lParam zeigt auf ein Array von 5 POINT-Strukturen lParam[0] intern für Windows. lParam[1] ist die Maximalgröße lParam[2] ist die Position der oberen linken Fensterecke des vergößerten Fensters. lParam[3] ist die minimale Tracking-Größe des Fensters. lParam[4] ist die maximale Tracking-Größe des Fensters.

Rückgabewert

Elemente 1 bis 4 von lParam können verändert werden.

Hinweise

Die Tracking-Größen sind die erlaubten Minimal- und Maximalgrößen des Fensters.
Diese Botschaft dient zum Verändern der Fenstervorgabegrößen, bevor sie Windows verwendet.

wm_GetText

Beschreibung

Kopiert den einem Fenster zugeordneten Text in einen Puffer.

Parameter

wParam	Die größte kopierbare Anzahl Bytes, die in den Puffer lParam kopiert werden können.
lParam	Zeiger auf einen Puffer, der mindestens wParam Bytes lang sein muß.

Rückgabewert

cb_Err, wenn das Fenster ein Kombinationsfenster ohne Editierelement ist;
lb_Err, wenn das Fenster ein Listenfenster ohne gewähltes Element ist;
ansonsten wird die Zahl der kopierten Bytes samt dem null-terminierenden Zeichen zurückgeliefert.

Hinweise

Editierfelder	Der Text ist der Inhalt des Editierelements
Schalter	Der Text ist dr Name des Aktionsschalters
Listen	Der Text ist das gewählte Element, falls vorhanden
Kombinationsfenster	Der Text ist der Inhalt des Kombinationsfenster-Editierfelds
Alle übrigen Felder	Der Text ist die Fensterüberschrift

Siehe auch:

wm_GetTextLength
wm_SetText

wm_GetTextLength

Beschreibung

Ruft die Länge des einem Fenster zugeordneten Textes in Bytes ab.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Die Textlänge ohne null-terminierenden Zeichen.

Hinweise

Editierfelder	Der Text ist der Inhalt des Editierelements
Schalter	Der Text ist der Name des Aktionsschalters
Listen	Der Text ist das gewählte Element, falls vorhanden
Kombinationsfenster	Der Text ist der Inhalt des Kombinationsfenster-Editierfelds
Alle übrigen Fenster	Der Text ist die Fensterüberschrift

Siehe auch:

wm_GetText

wm_HScroll

Beschreibung

Wird übergeben, wenn der Benutzer die horizontale Bildlaufleiste mit der Maus anklickt.

Parameter

wParam	Ein Bildlaufleistencode, der die Wünsche des Anwenders angibt, kann den Wert einer der **sb_xxx-Konstanten** annehmen, die für waagrechte Bildlaufleistenelemente gelten.
lParamLo	Nicht verwendet
lParamHi	Handle des Bildlaufleistenelements; lParamHi ist 0, wenn das Bildlaufleistenelement zusammen mit dem Fenster mit dem Stil **ws_HScroll** erstellt wurde.

Rückgabewert

Nicht verwendet

Hinweise

Wenn die Anwendung Fenstertext verschiebt, sollte mit **SetScrollPos** auch die Positionsmarke der Bildlaufleiste aktualisiert werden.

Siehe auch:

sb_xxx-Konstanten

wm_HScrollClipboard

Beschreibung

Fordert horizontalen Bildlauf für das Format **cf_OwnerDisplay** an.

Parameter

Param	Handle auf das Zwischenablagefenster
lParamLo	Bildlaufleistencode, der das Anklicken beschreibt, kann eine der **sb_xxx-Konstanten** für waagrechte Bildlaufleisten sein.
lParamHi	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Der Besitzer der Zwischenablage muß das Fenster restaurieren oder die Funktion **InvalidateRect** aufrufen.

Die Position der Bildlaufleiste der Zwischenablage muß mit der Funktion **SetScrollPos** aktualisiert werden.

Folgende Botschaften werden an den Besitzer der Zwischenablage gesendet, wenn das Format der Zwischenablage **cf_OwnerDisplay** ist:

wm_AskCBFormatName
wm_HScrollClipboard
wm_PaintClipboard
wm_SizeClipboard
wm_VScrollClipboard

Daten und Format der Zwischenablage werden mit der Funktion **SetClipboardData** gesetzt.

Siehe auch:

sb_xxx-Konstanten

wm_IconEraseBkgnd

Beschreibung

Wird an ein verkleinertes Fenster übergeben, wenn der Hintergrund vor Zeichnung des Symbols ausgemalt werden muß.

Parameter

wParam	Gerätekontext des Symbols
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Ein Fenster empfängt diese Botschaft nur, wenn ein Klassensymbol für dieses Fenster definiert wurde, andernfalls wird die Botschaft **wm_EraseBkgnd** übergeben.
Die Vorgabeaktion der Funktion **DefWindowProc** ist das Ausfüllen des Symbol-Hintergrunds mit dem Pinsel des übergeordnetes Fensters.

wm_InitDialog message

Beschreibung

Diese Botschaft wird unmittelbar vor der Darstellung eines Dialogfensters übergeben und ermöglicht einer Anwendung Initialisierungen durchzuführen.

Parameter

wParam — ID des ersten Elements im Dialogfenster, das den Fokus erhalten kann

lParam — Der Wert InitParam, der an die Funktion weitergegeben wird, die das Dialogfenster erzeugte. Folgende Funktionen verwenden diesen Parameter:
CreateDialogIndirectParam **CreateDialogParam**
DialogBoxIndirectParam
DialogBoxParam

lParam ist 0, wenn das Dialogfenster mit einer der folgenden Funktionen erzeugt wurde:
CreateDialog
CreateDialogIndirect
DialogBox
DialogBoxIndirect

Rückgabewert

Ungleich 0, wenn das Programm den Fokus nicht vergibt;
0, wenn ein Element des Dialogfensters den Fokus erhält.

Hinweise

Mit dieser Botschaft kann ein Programm ein Dialogfenster initialisieren und einem Element den Fokus geben, bevor das Dialogfenster dargestellt wird.
Ist der Rückgabewert 0, gibt Windows den Fokus dem Element in wParam. wParam ist üblicherweise die ID des ersten Elements im Dialogfenster mit dem Stil **ws_TabStop**.
Am Besten wird die Schrift des Dialogfensters zu diesem Zeitpunkt mit **wm_SetFont** bestimmt.

wm_InitMenu

Beschreibung

Diese Botschaft ist die Aufforderung zur Initialisierung eines Menüs.

Parameter

wParam	Handle zum Menü
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Botschaft tritt auf, wenn der Anwender eine Menüleiste anklickt oder eine Menütaste betätigt. Windows übergibt diese Botschaft vor der Darstellung des Menüs. Dies ermöglicht es der Anwendung, den Status der Menüeinträge vorab einzustellen.

wm_InitMenuPopup

Beschreibung

Diese Botschaft wird unmittelbar vor Darstellung eines Pop-up-Menüs übergeben.

Parameter

wParam	Handle zum Pop-up-Menü
lParamLo	Index des Pop-up-Menüs im Hauptmenü
lParamHi	Ungleich 0, wenn das Pop-up-Menü das Systemmenü ist, sonst 0.

Rückgabewert

Nicht verwendet

Hinweise

Die Bearbeitung der Botschaft ermöglicht es der Anwendung, den Status der Pop-up-Menüeinträge vorab und ohne Auswirkung auf den gesamten Menüstatus zu verändern.

wm_KillFocus

Beschreibung

Diese Botschaft wird übergeben, unmittelbar bevor ein Fenster den Fokus verliert.

Parameter

wParam	Handle des Fensters, das den Fokus erhält
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Stellt eine Anwendung ein Caret dar, sollte dieses jetzt zerstört werden. wParam kann 0 sein

wm_LButtonDblClk, wm_MButtonDblClk und wm_RButtonDblClken

Beschreibung

wm_ LButtonDblClk: Diese Botschaft tritt auf, wenn der Anwender einen Doppelklick mit der linken Maustaste durchführt.
wm_ MButtonDblClk: Diese Botschaft tritt auf, wenn der Anwender einen Doppelklick mit der mittleren Maustaste durchführt.
wm_ RButtonDblClk: Diese Botschaft tritt auf, wenn der Anwender einen Doppelklick mit der rechten Maustaste durchführt.

Parameter

wParam	Enthält einen Wert, der anzeigt, welche virtuellen Tasten gedrückt werden. Möglich ist jede Kombination der **mk_xxx-Konstanten**
lParamLo	x-Koordinate des Mauszeigers
lParamHi	y-Koordinate des Mauszeigers

Rückgabewert

Nicht verwendet

Hinweise

Nur Fenster, deren Fensterklasse den Stil **cs_DblClks** hat, können Doppelklickbotschaften empfangen. Windows erzeugt eine Doppelklickbotschaft, wenn eine Maustaste gedrückt, losgelassen und innerhalb der vom System vorgegebenen Doppelklickzeit ein zweites Mal betätigt wird. Ein Doppelklick erzeugt vier Botschaften: Drücken der Taste, Loslassen der Taste, Doppelklickbotschaft, erneutes Loslassen der Taste.

Siehe auch:

wm_xxx-Botschaften

wm_LButtonDown, wm_LButtonUp, wm_MButtonDown, wm_MButtonUp, wm_RButtonDown, wm_RButtonUp und wm_MouseMoveen

Beschreibung

wm_LButtonDown tritt auf, wenn der Anwender die linke Maustaste drückt.
wm_LButtonUp tritt auf, wenn der Anwender die linke Maustaste losläßt.
wm_MButtonDown tritt auf, wenn der Anwender die mittlere linke Maustaste drückt.
wm_MButtonUp tritt auf, wenn der Anwender die linke Maustaste losläßt.
wm_RButtonDown tritt auf, wenn der Anwender die rechte Maustaste drückt.
wm_RButtonUp tritt auf, wenn der Anwender die rechte Maustaste losläßt.
wm_MouseMove tritt auf, wenn der Anwender die Maus innerhalb des Client-Bereichs des Fensters bewegte.

Parameter

wParam	Enthält einen Wert, der den Status verschiedener virtueller Tasten anzeigt. Jede Kombination der Werte der **mk_xxx-Konstanten** ist möglich:
lParamLo	Die x-Koordinate des Mauszeigers
lParamHi	Die y-Koordinate des Mauszeigers

Rückgabewert

Nicht verwendet

Hinweise

Diese Koordinaten sind relativ zur linken oberen Ecke des Fensters.

Siehe auch:

wm_Botschaften

wm_MDIActivate

Beschreibung

Eine Anwendung übergibt diese Botschaft an ein MDI-Client-Fenster (Multidokumentschnittstelle), um die Aktivierung eines untergeordneten MDI-Fensters zu veranlassen. Diese Botschaft muß von einem Client-Fenster mit dem entsprechenden Paramter wParam sowohl an das zu aktivierende als auch an das zu deaktivierende untergeordnete Fenster übergeben werden.

Parameter

wParam	Das MDI-Client-Fenster verwendet wParam nicht. Ein MDI-Child-Fenster wird deaktiviert, wenn wParam 0 ist, sonst aktiviert.
lParamLo	Fenster-Handle des zu aktivierenden untergeordneten Fensters.
lParamHi	Fenster-Handle des zu deaktivierenden untergeordneten Fensters

Rückgabewert

Nicht verwendet

Hinweise

Untergeordnete MDI-Fenster werden unabhängig vom MDI-Rahmenfenster aktiviert. Wird der Rahmen aktiv, erhält das zuletzt mit der wm_MDIActivate aktivierte, untergeordnete Fenster die Botschaft **wm_MDIActivate**, um Rahmen und Fenstertitel zu zeichnen. Weitere wm_MDIActivate-Botschaften werden nicht an das Fenster übergeben.

wm_MDICascade und wm_MDITileen

Beschreibung

wm_MDICascade überführt die einem MDI-Client-Fenster (Multidokumentschnittstelle) untergeordneten Fenster in das Format »Cascade« (überlappend).
wm_MDITile überführt die einem MDI-Client-Fenster (Multidokumentschnittstelle) untergeordneten Fenster in das Format »Tile« (nebeneinander).

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_MDICreate

Beschreibung

Diese Botschaft veranlaßt ein MDI-Client-Fenster (Multidokumentschnittstelle), ein untergeordnetensFenster zu erstellen.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf eine **TMDICreateStruct**-Datenstruktur

Rückgabewert

Der Rückgabewert enthält im niederwertigen Word den Bezeichner des neuen Fensters und 0 im höherwertigen Word.

Hinweise

Das Fenster wird mit den folgenden Stil-Bits, sowie den zusätzlichen Stil-Bits in der durch lParam bezeichneten lParam TMDICreateStruct -Datenstruktur erstellt:

ws_Child	ws_ClipSiblings
ws_ClipChildren	ws_SysMenu
ws_Caption	ws_ThickFrame
ws_MinimizeBox	ws_MaximizeBox

Windows fügt die Überschrift des neuen Fensters in das Fenstermenü des Rahmenfensters ein. Die Anwendung sollte alle dem Client-Fenster untergeordneten Fenster mit dieser Botschaft erzeugen.

Bei Erzeugung des untergeordneten MDI-Fensters übergibt Windows die Botschaft **wm_Create** an das Fenster. Der Parameter lParam dieser Botschaft enthält einen Zeiger auf eine **TCreateStruct**-Datenstruktur. Diese Datenstruktur enthält wiederum einen Zeiger auf die TMDICreateStruct -Datenstruktur, die mit der das untergeordnete MDI-Fenster erzeugenden Botschaft wm_MDICreate übergeben wird.

Eine Anwendung sollte keine zweite wm_MDICreate-Botschaft übergeben, während die vorhergehende noch bearbeitet wird. Es sollte zum Beispiel keine wm_MDICreate-Botschaft übergeben, während ein untergeordnetes MDI_Fenster seine wm_Create-Botschaft abarbeitet.

Siehe auch:

ws_xxx-Konstanten

wm_MDIDestroy

Beschreibung

Empfängt ein MDI-Client-Fenster (Multidokumentschnittstelle) diese Botschaft, wird ein untergeordnetes Fenster geschlossen.

Parameter

wParam	Fenster-Handle des untergeordneten Fensters.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft löscht die Überschrift des untergeordneten Fensters aus dem Rahmenfenster und schließt es dann. Anwendungen sollten alle untergeordneten MDI-Fenster mit Hilfe dieser Botschaft schließen.

wm_MDIGetActive

Beschreibung

Diese Botschaft liefert das aktive untergeordnete MDI-Fenster (Multidokumentschnittstelle) zusammen mit einem Flag, das anzeigt, ob das Fenster vergrößert ist.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert enthält das Handle des untergeordneten MDI-Fensters im niederwertigen Word. Das höherwertige Word enthält den Wert 1, wenn das Fenster vergrößert ist; andernfalls ist der Wert 0.

wm_MDIIconArrange

Beschreibung

Diese Botschaft wird an ein MDI-Client-Fenster (Multidokument- schnittstelle) übergeben, um die verkleinerten untergeordneten Dokumentenfenster anzuordnen. Es werden nur verkleinerte Fenster (Symbole) angeordnet.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Untergeordnete MDI-Fenster, die nicht zum Symbol verkleinert sind, sind nicht betroffen.

wm_MDIMaximize

Beschreibung

Diese Botschaft veranlaßt ein MDI-Client-Fenster (Multidokumentschnittstelle), ein untergeordnetes MDI-Fenster zu vergrößern.

Parameter

wParam	Die ID des untergeordneten MDI-Fensters
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Wenn ein untergeordnetes MDI-Fenster vergrößert wird, geschieht folgendes:

- Windows paßt die Größe des zu öffnenden Fensters an das Client-Fenster an, wenn es zu groß ist,
- das Systemmenü des untergeordneten Fensters wird in der Menüzeile des MDI-Rahmens positioniert, und
- und die Überschrift der Überschrift des MDI-Rahmenfensters hinzugefügt.

wm_MDINext

Beschreibung

Aktiviert das dem aktiven unmittelbar folgende, untergeordnete MDI-Fenster (Multidokumentschnittstelle).

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Das nächste untergeordnete Fenster ist das direkt hinter dem aktiven untergeordneten MDI-Fenster.
Das aktuell aktive untergeordnete Fenster wird hinter alle anderen untergeordneten Fenster plaziert.

wm_MDIRestore

Beschreibung

Diese Botschaft stellt die Normalgröße eines untergeordnetes MDI-Fensters (Multidokumentschnittstelle) ausgehend von der Vollbild- oder der Symboldarstellung wieder her.

Parameter

wParam	Die ID des untergeordneten MDI-Fensters
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_MDISetMenu

Beschreibung

Diese Botschaft tauscht das Menü eines MDI-Rahmenfensters (Multidokumentschnittstelle), das Pop-up-Menü des Fensters oder beide aus.

Parameter

wParam	Nicht verwendet
lParamLo	Enthält das Menü-Handle des neuen MDI-Rahmenfenstermenüs oder den Wert 0
lParamHi	Enthält das Menü- Handle des neuen Pop-up-Menüs oder den Wert 0

Rückgabewert

Der Rückgabewert ist das Handle des Rahmenfenstermenüs, das durch die Botschaft ersetzt wird.

Hinweise

Sind lParamLo oder lParamHi 0, wird das entsprechende Menü nicht geändert.

Nach Übergabe dieser Botschaft muß die Anwendung die Funktion **DrawMenuBar** zur Aktualisierung der Menüleistendarstellung aufrufen.

Tauscht die Botschaft ein Fenster-Pop-up-Menü aus, werden die Menüeinträge des untergeordneten MDI-Fensters aus dem vorhergehenden Fenstermenü gelöscht und zum neuen Pop-Up-Menü hinzugefügt.

Tauscht diese Botschaft ein MDI-Rahmenfenstermenü aus, während ein untergeordnetes MDI-Fenster vergrößert dargestellt wird, werden das Systemmenü und die Wiederherstellungssteuerelemente aus dem vorhergehenden Rahmenfenstermenü entfernt und in das neue Menü eingefügt.

wm_MeasureItem

Beschreibung

Diese Botschaft wird an den Besitzer eines besitzergezeichneten Schalters, Kombinationsfensters, Menüeintrages oder einer Liste übergeben, wenn das betreffende Feld erzeugt wird, um die Größe des betreffenden Elementes zu ermitteln.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf eine **TMeasureItemStruct**-Datenstruktur.

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft gilt nur für benutzerdefinierte Menüelemente und folgende Elemente:

- Aktionsschalter mit dem Stil **bs_OwnerDrawbs**
- Kombinationsfenster mit einer der **cbs_xxx-Konstanten**, cbs_OwnerDrawFixed oder cbs_OwnerDrawVariable
- Listen mit einer der Konstanten **lbs_xxx-Konstanten** lbs_OwnerDrawFixed oder lbs_OwnerDrawVariable

Die Struktur TMeasureItemStruct, auf welche lParam zeigt, muß mit den richtigen Werten des Elements gefüllt werden.

Werden die Stile lbs_OwnerDrawVariable oder cbs_OwnerDrawVariable zur Erstellung der Liste oder des Kombinationsfensters verwendet, empfängt der Besitzer eine Botschaft für jeden Eintrag im Steuerelement.

Besitzt ein Dialogfenster ein Kombinationsfenster mit dem Stil cbs_OwnerDrawFixed oder eine Liste mit dem Stil lbs_OwnerDrawFixed, erhält sie die Botschaft wm_MeasureItem vor **wm_InitDialog**.

wm_MenuChar

Beschreibung

Diese Botschaft wird übergeben, wenn der Anwender eine Tastenkombination betätigt, die keiner vordefinierten Tastenkombination des aktuellen Menüs entspricht. Die Botschaft wird an das Fenster übergeben, welches das Menü verwaltet.

Parameter

wParam	Enthält das vom Anwender gedrückte ASCII-Zeichen.
lParamLo	Enthält das Flag **mf_Popup**, wenn ein Pop-up-Menü, das Flag **mf_SysMenu**, wenn ein Systemmenü bearbeitet wird.
lParamHi	Enthält das Handle des ausgewählten Menüs.

Rückgabewert

- 0: Weist Windows an, das vom Benutzer gewählte Zeichen zu verwerfen, und erzeugt ein Warnsignal im Systemlautsprecher.
- 1: Weist Windows an, das aktuelle Menü zu schließen.
- 2: Informiert Windows, daß im niederwertigen Word des Rückgabewerts die Eintragsnummer eines bestimmten Eintrags abgelegt ist. Der betreffende Eintrag wird von Windows ausgewählt.

Hinweise

Werden Accelerator-Tasten verwendet, um in einem Menü positionierte Bitmaps auszuwählen, sollte die Anwendung diese Botschaft bearbeiten.

wm_MenuSelect

Beschreibung

Die Botschaft tritt nach Auswahl eines Menüeintrags durch den Anwender auf.

Parameter

wParam	Die zugehörige ID oder den Handle des Pop-up-Menüs.
lParamLo	Enthält enweder -1 oder eine Kombination der folgenden **mf_xxx-Konstanten**: mf_Bitmap mf_Checked mf_Disabled mf_Grayed mf_MouseSelect mf_OwnerDraw mf_Popup mf_SysMenu
lParamHi	Ein Handle zum Menü oder 0, wenn lParamLo -1 oder wenn es sich um das Systemmenü handelt.

Rückgabewert

Nicht verwendet

Hinweise

Enthält lParamLo den Wert -1 und lParamHi den Wert 0, wurde das Menü von Windows geschlossen, weil der Anwender die ESC-Taste gedrückt oder außerhalb des Menüs geklickt hat.

wm_MouseActivate

Beschreibung

Diese Botschaft tritt auf, wenn sich der Zeiger in einem nicht aktiven Fenster befindet und eine Maustaste gedrückt wird.

Parameter

wParam	Enthält das Handle des höchsten, dem zu aktivierenden Fenster übergeordneten Fensters.
lParamLo	Enthält eine der **ht-Konstanten**; diese Werte werden auch von der Botschaft **wm_NCHitTest** zurückgegeben.
lParamHi	Enthält die Nummer der Mausbotschaft.

Rückgabewert

ma_Activate, wenn das erste Fenster mit dieser Botschaft aktiviert wird
ma_NoActivate, wenn das erste Fenster mit dieser Botschaft nicht aktiviert wird
ma_ActivateAndEat, wenn das erste Fenster mit dieser Botschaft aktiviert wird und das Mausereignis entfernt wird

Hinweise

Keine weiteren Rückgabewerte sind legal.

Übergibt ein untergeordnetes Fenster die Botschaft an die Funktion **DefWindowProc**, wird sie von dieser Funktion vor jeder Bearbeitung an das übergeordnete Fenster weitergeleitet. Gibt das übergeordnete Fenster ungleich 0 zurück, wird die Bearbeitung gestoppt.

Siehe auch

ma_xxx-Konstanten

wm_Move

Beschreibung

Diese Botschaft wird nach dem Verschieben eines Fensters übergeben.

Parameter

wParam	Nicht verwendet
lParamLo	Enthält die neue x-Koordinate der oberen linken Ecke des Client-Bereichs des Fensters.
lParamHi	Enthält die neue y-Koordinate der oberen linken Ecke des Client-Bereichs des Fensters.

Rückgabewert

Nicht verwendet

Hinweise

Die neue Position wird für überlappende und Pop-up-Fenster in Form von Bildschirmkoordinaten angegeben.
Die Position eines untergeordneten Fensters wird in Abhängigkeit vom übergeordneten Fenster angegeben.

wm_NCActivate

Beschreibung

Diese Botschaft wird an ein Fenster übergeben, wenn dessen Nicht-Client-Bereich verändert werden muß, um den aktiven bzw. inaktiven Status anzuzeigen.

Parameter

wParam	0 bei inaktiven Titeln oder Symbolen, wParam ist ungleich 0, wenn ein aktiver Titel bzw. ein aktives Symbol dargestellt werden soll.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Funktion **DefWindowProc** zeichnet eine graue Titelleiste für inaktive und eine schwarze Titelleiste für aktive Fenster.

wm_NCCalcSize

Beschreibung

Diese Botschaft wird übergeben, wenn die Größe des Client-Bereichs eines Fensters berechnet werden muß.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf eine Rect -Datenstruktur, in der die Koordinaten des Fensterrechtecks (einschließlich Client-Bereich, Grenzlinien, Überschrift, Bildlaufleisten usw.) abgelegt sind.

Rückgabewert

Nicht verwendet

Hinweise

Der mit lParam definierter Client-Bereich umfaßt Grenzlinien, Überschrift, Bildlaufleisten usw

Die Funktion **DefWindowProc** berechnet die Größe des Client-Bereichs, ausgehend von den Daten des Fensters (Existenz von Bildlaufleisten, Menüs, usw.). Das Ergebnis wird in der lParam Rect -Datenstruktur abgelegt.

wm_NCCreate

Beschreibung

Diese Botschaft wird noch vor der Botschaft wm_Create beim Erzeugen eines Fensters übergeben.

Parameter

wParam	Enthält das Handle des zu erstellenden Fensters.
lParam	Zeigt auf die **TCreateStruct**-Datenstruktur des Fensters.

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn der Nicht-Client-Bereich erzeugt wird. Er ist 0, wenn ein Fehler auftritt.

Hinweise

Falls diese Botschaft 0 zurückgibt, ist auch der Rückgabewert der Funktion **CreateWindow** 0.

Die Standardaktion der Funktion **DefWindowProc** ist hier, Bildlaufleisten zu initialisieren (Setzen von Position und Anzeigebereich der Bildlaufleiste) und den Fenstertext zu setzen. Intern verwendeter Speicher für Erzeugung und Verwaltung des Fensters wird reserviert.

wm_NCDestroy

Beschreibung

Diese Botschaft teilt einem Fenster mit, daß sein Nicht-Client-Bereich zerstört wird.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Funktion **DestroyWindow** übergibt die Botschaft **wm_Destroy** im Anschluß an die wm_Destroy-Botschaft an das Fenster.

Diese Botschaft wird von der Funktion **DefWindowProc** verwendet, um den dem Fenster zugewiesenen Speicherbereich freizugeben.

wm_NCHitTest

Beschreibung

Diese Botschaft wird an das Fenster übergeben, das den Zeiger enthält

Parameter

wParam	Nicht verwendet
lParamLo	Die X-Koordinate des Mauszeigers
lParamHi	Die Y-Koordinate des Mauszeigers

Rückgabewert

Der Rückgabewert ist einer der **ht-Konstanten**, falls diese Botschaft der Funktion **DefWindowProc** übergeben wird.

Hinweise

Die Maus-Koordinaten beziehen sich auf die obere linke Ecke des Bildschirms.

Diese Botschaft wird an das Fenster übergeben, das den Zeiger enthält oder das die Funktion **GetCapture** verwendet hat, um die Mauseingabe auf sich zu ziehen. Die Botschaft wird bei jeder Mausbewegung übergeben.

Siehe auch:

wm_xxx-Botschaften

wm_NCLButtonDblClk, wm_NCMButtonDblClk und wm_NCRButtonDblClken

Beschreibung

wm_NCLButtonDblClk: Diese Botschaft wird an ein Fenster übergeben, wenn der Anwender einen Doppelklick mit der linken Maustaste innerhalb eines Nicht-Client-Bereichs des Fensters ausführt.
wm_NCMButtonDblClk: Diese Botschaft wird an ein Fenster übergeben, wenn der Anwender einen Doppelklick mit der mittleren Maustaste innerhalb eines Nicht-Client-Bereichs des Fensters ausführt.
wm_NCRButtonDblClk: Diese Botschaft wird an ein Fenster übergeben, wenn der Anwender einen Doppelklick mit der rechten Maustaste innerhalb eines Nicht-Client-Bereichs des Fensters ausführt.

Parameter

wParam	Einer der **ht-Konstanten**. Enthält den Code, der von **wm_NCHitTest** zurückgegeben wird.
lParamLo	X-Koordinate des Mauszeigers
lParamHi	Y-Koordinate des Mauszeigers

Rückgabewert

Nicht verwendet

Hinweise

Die Maus-Koordinaten werden relativ zur oberen linken Ecke des Bildschirms gebildet.
Die Standardaktion von **DefWindowProc** besteht unter anderem aus dem Senden der entsprechenden **wm_SysCommand**-Botschaft, in Abhängigkeit vom betroffenden Nicht-Client-Bereichs.

Ein Doppelklick erzeugt vier Botschaften: Drücken der Taste, Loslassen der Taste, Doppelklickbotschaft, erneutes Loslassen der Taste.

Siehe auch:

wm_xxx-Botschaften

wm_NCLButtonDown, wm_NCLButtonUp, wm_NCMButtonDown, wm_NCMButtonUp, wm_NCRButtonDown, wm_NCLButtonUp und wm_NCMouseMoveen

Beschreibung

wm_NCLButtonDown wird an ein Fenster übergeben, wenn der Anwender die linke Maustaste drückt, während sich der Zeiger im Nicht-Client-Bereich des Fensters befindet.
wm_NCLButtonUp wird an ein Fenster übergeben, wenn der Anwender die linke Maustaste losläßt, während sich der Zeiger im Nicht-Client-Bereich des Fensters befindet.
wm_NCMButtonDown wird an ein Fenster übergeben, wenn der Anwender die mittlere Maustaste drückt, während sich der Zeiger im Nicht-Client-Bereich des Fensters befindet.
wm_NCMButtonUp wird an ein Fenster übergeben, wenn der Anwender die mittlere Maustaste losläßt, während sich der Zeiger im Nicht-Client-Bereich des Fensters befindet.
wm_NCRButtonDown wird an ein Fenster übergeben, wenn der Anwender die rechte Maustaste drückt, während sich der Zeiger im Nicht-Client-Bereich des Fensters befindet.
wm_NCRButtonUp wird an ein Fenster übergeben, wenn der Anwender die rechte Maustaste losläßt, während sich der Zeiger im Nicht-Client-Bereich des Fensters befindet.
wm_NCLMouseMove wird an ein Fenster übergeben, wenn der Anwender den Mauszeiger bewegt, während sich der Zeiger im Nicht-Client-Bereich des Fensters befindet.

Parameter

wParam	Eine der **ht-Konstanten**; dieselben Werte werden von der Botschaft **wm_NCHitTest** zurückgegeben.
lParamLo	X-Koordinate des Mauszeigers
lParamHi	Y-Koordinate des Mauszeigers

Rückgabewert

Nicht verwendet

Hinweise

Diese Koordinaten werden relativ zur oberen linken Ecke des Bildschirms gebildet.
Die Standardaktion von **DefWindowProc** beinhaltet das Übergeben der entsprechenden **wm_SysCommand** -Botschaften, in Abhängigkeit vom betroffenen Nicht-Client-Bereich.

Siehe auch:

wm_xxx-Botschaften

wm_NCPaint

Beschreibung

Diese Botschaft wird übergeben, wenn der Fensterrahmen gemalt werden muß.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Funktion **DefWindowProc** malt den Fensterrahmen. Anwendungen können diese Botschaft abfangen und individuelle Fensterrahmen malen.
Der Clipping-Bereich eines Fensters ist immer rechteckig, auch wenn die Form des Rahmens hiervon abweicht.

wm_NextDlgCtl

Beschreibung

Ändert den Steuerelementfokus eines Dialogfensters.

Parameter

wParam	Ist wParam 0, erhält das nächste Steuerelement den Fokus, andernfalls das vorhergehende. Wenn lParam ungleich 0 ist, bezeichnet wParam das Steuerelement, das den Fokus erhält.
lParam	Enthält ein Flag, das anzeigt, wie Windows den Parameter wParam interpretiert. Ist lParam ungleich 0, stellt wParam ein Handle zum Steuerelement bereit, das den Fokus erhält. Andernfalls ist wParam ein Flag, das anzeigt, ob das nächste oder das vorhergehende Steuerelement den Fokus erhält.

Rückgabewert

Nicht verwendet

Hinweise

Im Gegensatz zur Funktion **SetFocus** ändert diese Botschaft den Rahmen des Vorgabe-Steuerelements.
Die Funktion **SendMessage** sollte diese Botschaft keinesfalls übergeben, wenn die Anwendung gleichzeitig andere Botschaften zum Setzen des Steuerelementfokus bearbeitet. Stattdessen kann die Funktion **PostMessage** verwendet werden.

wm_Paint

Beschreibung

Diese Botschaft wird übergeben, wenn Windows oder eine Anwendung die Aktualisierung eines Teils eines Anwendungsfensters anfordert.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Botschaft wird entweder beim Aufruf der Funktion **UpdateWindow** oder durch die Funktion **DispatchMessage** übergeben, wenn die Anwendung die Botschaft **wm_Paint** erhält.

Siehe auch:

BeginPaint
EndPaint

wm_PaintClipboard

Beschreibung

Die Botschaft wird an den Besitzer der Zwischenablage übergeben, um die Aktualisierung des Client-Bereichs (in Teilen oder komplett) der Zwischenablage anzufordern.

Parameter

wParam	Enthält ein Handle zum Fenster der Zwischenablage.
lParamLo	lParam bezeichnet eine **TPaintStruct**-Datenstruktur, die festlegt, welcher Teil des Client-Bereichs zu aktualisieren ist.
lParamHi	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Anwendungen müssen die Funktion **GlobalLock** verwenden, um den Speicherbereich, der die TPaintStruct -Datenstruktur enthält, zu sperren. Dieser Speicherbereich sollte mit Hilfe der Funktion **GlobalUnlock** vor Rück- oder Abgabe der Steuerung wieder freigegeben werden.

Um zu ermitteln, ob der gesamte Client-Bereich oder nur ein Teil aktualisiert werden muß, muß der Besitzer der Zwischenablage die Daten des zu zeichnenden Bereichs im Feld rcpaint der TPaintStruct - Datenstruktur mit den durch die Botschaft **wm_SizeClipboard** vorgegebenen vergleichen.

Die folgenden Botschaften werden an die Zwischenablage gesendet, wenn der Besitzer **cf_OwnerDisplay** ist:
wm_AskCBFormatName
wm_HScrollClipboard
wm_PaintClipboard
wm_SizeClipboard
wm_VScrollClipboard

Zwischenablagedaten und -Format werden mit der Funktion **SetClipboardData** eingestellt.

wm_PaintIcon

Beschreibung

Diese Botschaft wird an ein zum Symbol verkleinertes Fenster übergeben, wenn das Symbol gezeichnet werden soll.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Botschaft **wm_Paint** wird stattdessen gesendet, wenn es kein Klassen-Symbol für das Fenster gibt.
Die Standardaktion von **DefWindowProc** ist das Zeichnen des Fenster-Symbols mit dem Klassen-Symbol.

wm_PaletteChanged

Beschreibung

Diese Botschaft teilt allen Fenstern mit, daß das Fenster mit dem Fokus bei der Realisierung seiner virtuellen Farbpalette die Systemfarbpalette verändert hat.

Parameter

wParam	Enthält das Handle des Fensters, das die Systemfarbpalette verändert hat.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Dies erlaubt es Fenstern ohne Fokus ihre virtuelle Farbpalette zu realisieren und ihre Client-Bereiche zu aktualisieren.
Um die Programmierung einer Endlosschleife zu vermeiden, sollte ein Fenster vor Realisierung der eigenen Farbpalette sicherstellen, daß der Parameter wParam nicht sein eigenes Handle enthält.

wm_ParentNotify

Beschreibung

Diese Botschaft wird an das übergeordnete Fenster übergeben, wenn das untergeordnete Fenster erstellt oder zerstört wird, oder wenn der Anwender eine Maustaste gedrückt hat, während der Zeiger über einem untergeordneten Fenster liegt.

Parameter

wParam	Bestimmt das dem übergeordneten Fenster mitzuteilende Ereignis. Möglich sind folgende Werte: **wm_Create** **wm_Destroy** **wm_LButtonDown** **wm_MButtonDown** **wm_RButtonDown**
lParamLo	Wenn wParam wm_Create oder wm_Destroy ist, ist lParamLo das Handle des untergeordneten Fensters, sonst die X-Koordinate des Mauszeigers.
lParamHi	Wenn wParam wm_Create oder wm_Del ist, ist lParamHi ID des untergeordneten Fensters, sonst die Y-Koordinate des Mauszeigers.

Rückgabewert

Nicht verwendet

Hinweise

Die Botschaft wird an die übergeordneten Fenster aller untergeordneten Fenster übergeben, es sei denn, das untergeordnete Fenster hat den erweiterten Fensterstil **ws_ex_NoParentNotify**. Die Funktion **CreateWindow** oder **CreateWindowEx** erzeugt Fenster mit erweiterten Fensterstilen. In der Standardeinstellung sind alle einem Dialogfenster untergeordneten Fenster vom Stil ws_ex_NoParentNotify.

wm_Paste

Beschreibung

Diese Botschaft fügt die Daten aus der Zwischenablage in das Fenster an der aktuellen Zeiger-Position ein.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Daten werden nur dann eingefügt, wenn sie in der Zwischenablage im Format **cf_Text** vorliegen.

wm_QueryDragIcon

Beschreibung

Diese Botschaft wird an ein zum Symbol verkleinertes Fenster übergeben, das vom Anwender verschoben wird und das kein definiertes Klassensymbol besitzt.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert enthält im niederwertigen Word das Handle des Zeigers, den Windows während des Verschiebens darstellen soll. Er hat den Wert 0, wenn Windows den Standardsymbolzeiger darstellen soll. Der Standardrückgabewert ist 0.

Hinweise

Der Zeiger muß zur Auflösungsmöglichkeit des Gerätetreibers passen. Die Anwendung kann die Funktion **LoadCursor** aufrufen, um einen Zeiger aus den Ressourcen ihrer ausführbaren Datei zu laden und dessen Handle abzurufen.

wm_QueryEndSession

Beschreibung

Fragt jedes Programm, ob die Arbeitssitzung enden soll.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist ungleich 0, wenn die Anwendung problemlos geschlossen werden kann. Andernfalls ist der Rückgabewert 0.

Hinweise

Diese Botschaft wird übergeben, wenn der Anwender den Befehl zum Beenden der Arbeitssitzung anwählt. Gibt eine Anwendung 0 zurück, wird die Arbeitssitzung nicht beendet. In diesem Augenblick wird auch die weitere Übergabe von wm_QueryEndSession-Botschaften durch Windows beendet und allen Anwendungen, die einen Wert ungleich 0 zurückgegeben haben, wird die Botschaft **wm_EndSession** mit dem auf 0 gesetzten Parameter wParam übergeben.
Die Standardaktion von **DefWindowProc** soll einen Wert ungleich 0 zurückgeben.

wm_QueryNewPalette

Beschreibung

Diese Botschaft teilt einem Fenster mit, daß es den Fokus erhalten wird.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Realisiert das Fenster beim Empfang des Fokus eine eigene virtuelle Farbpalette, sollte es den Wert ungleich 0, andernfalls den Wert 0 zurückgeben.

wm_QueryOpen

Beschreibung

Diese Botschaft wird an ein Symbol übergeben, wenn der Anwender dessen Öffnung zu einem Fenster anfordert.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Der Rückgabewert ist 0, wenn die Anwendung die Öffnung des Symbols verweigert. Er ist ungleich 0, wenn das Symbol geöffnet werden kann.

Hinweise

Die Standardaktion der Funktion **DefWindowProc** gibt einen Wert ungleich 0 zurück.

wm_Quit

Beschreibung

Diese Botschaft zeigt die Aufforderung zum Beenden einer Anwendung an.

Parameter

wParam	Enthält den im Funktionsaufruf von **PostQuitMessage** vorgegebenen Endecode.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Nach dem Senden dieser Botschaft gibt **GetMessage** 0 zurück.
Der Endecode in wParam muß gespeichert und für das Programm verwendet werden.

wm_RenderAllFormats

Beschreibung

Diese Botschaft wird an die zu zerstörende Anwendung übergeben, wenn diese die Zwischenablage besitzt.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Anwendung sollte die Daten der Zwischenablage mit allen Formaten zurückgeben, die sie erzeugen kann. Dabei muß ein Handle für jedes Format an die Funktion **SetClipboardData** übergeben werden. Dies stellt sicher, daß die Daten in der Zwischenablage auch nach Beendigung der Anwendung bearbeitet werden können.

Siehe auch:

wm_RenderFormat

wm_RenderFormat

Beschreibung

Durch diese Botschaft wird der Besitzer der Zwischenablage aufgefordert, die letzten in die Zwischenablage kopierten Daten im angegebenen Format zu formatieren.

Parameter

wParam	Bestimmt das Datenformat. Möglich sind die bei der Funktion **SetClipboardData** beschriebenen Formate.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Daten sollten im angegebenen Format formatiert werden. Anschließend wird das Handle der formatierten Daten mit SetClipboardData an die Zwischenablage übergeben.

Siehe auch:

wm_RenderAllFormats

wm_SetCursor

Beschreibung

Diese Botschaft tritt auf, wenn Mauseingaben nicht ausgewertet werden, der Mauszeiger aber gleichzeitig in einem Fenster bewegt wird.

Parameter

wParam	Enthält das Handle des Fensters, in dem sich der Mauszeiger befindet.
lParamLo	Einer der **ht-Konstanten**; derselbe Wert wird auch von der Botschaft **wm_NCHitTest** zurückgegeben.
lParamHi	Die Nummer der Mausbotschaft

Rückgabewert

0, wenn **DefWindowProc** mit der Standardaktion fortfährt, sonst keine Aktion.

Hinweise

Die Standardaktion von **DefWindowProc** ist, dem Zeiger die Pfeilform zuzuweisen, wenn er sich in keinem Client-Bereich befindet. Andernfalls wird die registrierte klassenspezifische Form zugewiesen.
Die Botschaft wird nicht gesendet, wenn die Mauseingabe mit **SetCapture** aufgefangen wird.
Hat das niederwertige Word des Parameters lParam den Wert **htError** und enthält das höherwertige Word eine Maustastendruckbotschaft, wird die Funktion **MessageBeep** aufgerufen.

lParamHi ist 0, wenn das Fenster Menümodus erhält.

wm_SetFocus

Beschreibung

Diese Botschaft wird übergeben, wenn ein Fenster den Fokus erhält.

Parameter

wParam	Enthält das Handle des Fensters, das den Fokus abgibt (kann 0 sein).
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Um ein Caret darzustellen, sollte die Anwendung an dieser Stelle die entsprechenden Funktionen aufrufen.

wm_SetFont

Beschreibung

Diese Botschaft bestimmt die zur Textdarstellung verwendete Schrift eines Dialogfensters. Sets the font

Parameter

wParam	Enthält das Handle der Schrift. Hat dieser Parameter den Wert 0, stellt das Steuerelement den Text in der Standardsystemschrift dar.
lParam	Legt fest, ob das Steuerelement unmittelbar nach dem Setzen der Schrift neu gespeichert werden soll. Wird lParam auf ungleich 0 gesetzt, wird das Steuerelement neu gezeichnet.

Rückgabewert

Nicht verwendet

Hinweise

Eine Schrift muß mit **DeleteObject** gelöscht werden, wenn sie nicht mehr benötigt wird. Die Größe des Dialogfensters sollte vor dem Ändern der Schrift verändert werden.

Bevor Windows ein Dialogfenster des Stils **ds_SetFont** erzeugt, wird die Botschaft an das übergeordnete Fenster übergeben.
Auch nach Empfang der Botschaft **wm_InitDialog** sollte diese Botschaft gesendet werden.

wm_SetRedraw

Bfeschreibung

Setzt oder löscht das Neuzeichnen-Flag des Fensters.

Parameter

wParam	Bestimmt den Status des Aktualisierungs-Flags. Ist der Parameter wParam ungleich 0, ist das Flag gesetzt, d.h. Neuzeichnen ist nicht möglich.
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_SetText

Beschreibung

Diese Botschaft wird verwendet, um den Text eines Fensters zu setzen.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf den null-terminierten String mit dem Fenstertext.

Rückgabewert

cb_Err	Bei Kombinationsfenster ohne Editor
cb_ErrSpace	Wenn der verfügbare Platz nicht ausreicht, um den Text in das Editierfeld zu setzen.
lb_Err	Bei Listenfenstern ohne gewähltem Element
lb_ErrSpace	Wenn der verfügbare Platz nicht ausreicht, um den Text für das Listenfenster zu setzen.

Hinweise

Editierfelder	Der Text ist der Inhalt des Editierelements
Schalter	Der Text ist dr Name des Aktionsschalters
Listen	Der Text ist das gewählte Element, falls vorhanden
Kombinationsfenster	Der Text ist der Inhalt des Kombinationsfenster-Editierfeldes
Alle übrigen Fenster	Der Text ist die Fensterüberschrift

Durch diese Botschaft wird die aktuelle Auswahl in der Liste oder im Kombinationsfenster nicht verändert. Mit Hilfe der Botschaft **cb_SelectString** kann eine Anwendung den Eintrag auswählen, der dem Text im Editierfeld entspricht.

Siehe auch:

wm_GetText

wm_ShowWindow

Beschreibung

Diese Botschaft wird übergeben, wenn ein Fenster angezeigt oder verborgen wird.

Parameter

wParam	Bestimmt, ob ein Fenster gezeigt wird. Ist der Wert 0, wird das Fenster verborgen. Ist er ungleich 0, wird es angezeigt.
lParam	Bestimmt den Status des zu zeigenden Fensters. Der Parameter lParam hat den Wert 0, wenn die Botschaft aufgrund eines Aufrufs der Funktion **ShowWindow** übergeben wird. Andernfalls sind folgende Werte möglich: sw_ParentClosin.: Es wird ein übergeordnetes Fenster geschlossen (Symbol) oder ein Pop-up-Fenster verborgen. sw_ParentOpening: Es wird ein übergeordnetes Fenster geöffnet oder ein Pop-up-Fenster gezeigt.

Rückgabewert

Nicht verwendet

Hinweise

Ein Fenster wird angezeigt oder verborgen nach Aufruf der Funktion ShowWindow, nach Aktualisierung oder Vergrößerung eines überlappenden Fensters oder nach dem Schließen (Symbol) bzw. Öffnen (Bildschirmdarstellung) eines überlappenden bzw. eines Pop-up-Fensters.

Wird ein überlappendes Fenster geschlossen, werden alle zugehörigen Pop-up-Fenster verborgen.

Die Standardaktion von **DefWindowProc** ist das Zeigen oder Verdecken des Fensters.

Siehe auch:

sw_xxx-Konstanten

wm_Size

Beschreibung

Diese Botschaft wird nach Änderung der Größe eines Fensters übergeben.

Parameter

wParam	Eine der **Size-Konstanten**
lParamLo	Neue Breite des Client-Bereichs des Fensters
lParamHi	Neue Höhe des Client-Bereichs des Fensters

Rückgabewert

Nicht verwendet

Hinweise

Wenn die Funktion **SetScrollPos** oder **MoveWindow** beim Bearbeiten dieser Botschaft verwendet wird, sollten die Parameter Redraw (für SetScrollPos) oder Repaint (für MoveWindow) nicht 0 sein, damit das Fenster aktualisiert wird.

wm_SizeClipboard

Beschreibung

Diese Botschaft wird übergeben, wenn die Zwischenablage die Größe geändert hat.

Parameter

wParam	Bezeichnet das Fenster der Zwischenablage.
lParamLo	Eine Rect-Datenstruktur, die den Bereich bestimmt, der vom Besitzer der Zwischenablage darzustellen ist.
lParamHi	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Die Botschaft wird mit einem Rechteck mit den Koordinaten (0,0,0,0) als neuer Größe übergeben, wenn die Zwischenablage zerstört oder verkleinert werden soll. Dies erlaubt es dem Besitzer der Zwischenablage, seine Bildschirm-Ressourcen freizugeben.

Anwendungen müssen die Funktion **GlobalLock** verwenden, um den Speicher zu sperren, der die **TPaintStruct**-Datenstruktur enthält. Sie sollten den Speicher mit Hilfe der Funktion **GlobalUnlock** wieder freigeben, bevor sie die Steuerung ab- oder zurückgeben.

Rect in lParamLo sollte für die nächste Botschaft **wm_PaintClipboard** kopiert werden. Die Botschaften werden mit dem Zwischenablageformat **cf_OwnerDisplay** gesendet:
wm_AskCBFormatName
wm_HScrollClipboard
wm_PaintClipboard
wm_SizeClipboard
wm_VScrollClipboard

Das Format der Zwischenablage wird mit der Funktion **SetClipboardData** eingestellt.

wm_SpoolerStatus

Beschreibung

Diese Botschaft wird vom Druck-Manager übergeben, wenn ein Auftrag aus der Schlange des Druck-Managers entfernt oder in diese eingefügt wird.

Parameter

wParam	Setzt **sp_JobStatus**
lParamLo	Bestimmt im niederwertigen Word die Anzahl der in der Schlange des Druck-Managers verbleibenden Aufträge.
lParamHi	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Diese Botschaft gilt nur als Information.

wm_SysChar, wm_SysDeadChar, wm_SysKeyDown und wm_SysKeyUpen

Beschreibung

wm_SysChar meldet dem Fenster mit Fokus, daß eine Systemtaste gedrückt wurde, oder dem aktiven Fenster, wenn keines den Fokus hat, daß eine Taste gedrückt wurde.
wm_SysDeadChar meldet dem Fenster ein totes Systemzeichen.
wm_SysKeyDown meldet dem Fenster, wenn keines den Fokus hat, daß eine Systemtaste gedrückt wurde.
wm_SysKeyUp meldet dem Fenster mit Fokus, daß eine Systemtaste losgelassen wurde, oder dem aktiven Fenster, wenn keines den Fokus hat, daß eine Taste losgelassen wurde.

Parameter

wParam	Tastenwert
lParamLo	Zahl der Wiederholungsdrücke, weil die Taste festgehalten wurde.
lParamHi	Bits 0-7 von lParamHi Scan-Code (OEM-abhängiger Wert). Bit 8 ist 1, wenn eine zusätzliche Taste, z.B. Funktionstaste oder eine Taste des numerischen Tastenblocks verwendet wird. Bit 13 ist 1, wenn Alt zugleich gedrückt wurde Bit 14 ist 1, wenn die Taste vor dieser Botschaft gedrückt wurde Bit 15 ist 1, wenn die Taste losgelassen, 0, wenn sie gedrückt wurde (nur wm_SysDeadChar: lParamHi = Zahl der Wiederholungsdrücke, weil die Taste festgehalten wurde).

Rückgabewert

Nicht verwendet

Hinweise

Bei wm_SysChar, wm_SysKeyDown und wm_SysKeyUp ist Bit 13 von lParam 1, wenn eine Systemtaste gedrückt wurde, oder 0, wenn kein Fenster den Fokus hat.
Bei wm_SysChar und wm_SysKeyDown ist Bit 15 von lParamHi 0.
Bei wm_SysKeyUp, lParamHi ist Bit 15 von lParamHi 1.

Wenn Bit 13 von lParam 0 ist, kann diese Botschaft an die Funktion **TranslateAccelerator** weitergegeben werden, damit auch ohne Fokus hier Tastenkürzel verwendet werden können.
Hat kein Fenster den Fokus, werden wm_SysKeyDown, wm_SysChar und wm_SysKeyUp Botschaften anstatt **wm_KeyDown**, **wm_Char** und **wm_KeyUp** Botschaften gesendet.

Tote Tasten sind Umlaute und Akzente.
Eine tote Systemtaste ist eine solche zusammen mit Alt.
wn_SysDeadChar kann für Systemtasten verwendet werden, die nicht notwendigerweise in einem Zeichen resultieren.
Es kann wegen der Tastenwiederholfunktion mehr als einmal wm_SysKeyDown vor wm_SysKeyUp gesendet werden.

Siehe auch:

wm_DeadChar

wm_SysColorChange

Beschreibung

Diese Botschaft bestimmt den Wechsel einer oder mehrerer Systemfarben.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Windows übergibt eine **wm_Paint**-Botschaft an alle Fenster, die von der Änderung der Systemfarben beeinflußt werden.
Anwendungen, die Pinsel in den existierenden Systemfarben besitzen, sollten diese Pinsel löschen und mit den neuen Systemfarben neu erstellen.

wm_SysCommand

Beschreibung

Diese Botschaft wird übergeben, wenn der Anwender einen Befehl aus dem Systemmenü oder das Symbol für Vollbild- bzw. Symboldarstellung anwählt.

Parameter

wParam	Bestimmt den Typ des angeforderten Systembefehls. Möglich ist einer der folgenden Werte:**sc_xxx-Konstanten** Die niederwertigen vier Bits von wParam werden von Windows intern verwendet; sie sollten vor Verwendung von wParam auf 0 gesetzt werden
lParamLo	X-Koordinate der Maus, oder 0, wenn die Maus nicht verwendet wurde
lParamHi	Y-Koordinate der Maus, oder 0, wenn die Maus nicht verwendet wurde

Rückgabewert

Nicht verwendet

Hinweise

Accelerator-Tasten für Elemente des Systemmenüs werden zu wm_SysCommand anstatt zu **wm_Command**-Botschaften übersetzt.
wm_Command wird für Accelerator-Tasten nur gesendet, wenn

1. Das Fenster nicht verkleinert ist, oder
2. das Fenster verkleinert ist und kein Tastenkürzel zu einem Menüelement paßt.

System-Menüelemente können mit folgenden Funktionen geändert werden:
GetSystemMenu
AppendMenu
InsertMenu
ModifyMenu

Ein Programm muß alle geänderten System-Menüelemente verwalten.

Alle von einem Programm nicht behandelten Botschaften müssen an **DefWindowProc** weitergegeben werden.

wm_SysCommand können jederzeit an DefWindowProc gesendet werden, um Systembefehle auszuführen.

wm_TimeChange

Beschreibung

Diese Botschaft tritt auf, wenn eine Anwendung die Systemzeit (auch mehrfach) verändert.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Alle Anwendungen, die dies tun, sollten diese Botschaft mit **SendMessage** an alle Hauptfenster übergeben.

wm_Timer

Beschreibung

Diese Botschaft tritt auf, wenn die Zeitbegrenzung des vorgegebenen Timers überschritten wurde.

Parameter

wParam	Timer-ID
lParam	Zeigt auf eine Funktion, die bei der Erstellung des Timers an die Funktion **SetTimer** übergeben wurde. Windows ruft die festgelegte Funktion direkt auf, wenn der Parameter lParam den Wert 0 hat. Andernfalls wird die Botschaft direkt an die Fensterfunktion übergeben.

Rückgabewert

Nicht verwendet

wm_Undo

Beschreibung

Diese Botschaft macht die vorhergehende Operation rückgängig.

Parameter

wParam	Nicht verwendet
lParam	Nicht verwendet

Rückgabewert

Nicht verwendet

wm_VKeyToItem

Beschreibung

Diese Botschaft wird als Reaktion auf eine **wm_KeyDown**-Botschaft von einer Liste an den Besitzer der Liste übergeben.

Parameter

wParam	Taste
lParamLo	Handle zum Listenelement
lParamHi	Position des Carets

Rückgabewert

-2, wenn das Programm alles bearbeitet hat
-1, wenn die Liste die Standardaktion durchführen soll
>=0, wenn die Liste auf dem Indexelement die Standardaktion durchführen soll

Hinweise

Diese Botschaft gilt nur für Listen mit dem Stil **lbs_WantKeyboardInput**.

Siehe auch:

wm_CharToItem

wm_VScroll

Beschreibung

Diese Botschaft wird übergeben, wenn der Anwender die vertikale Bildlaufleiste anklickt.

Parameter

wParam	Bildlaufleistencode mit dem Effekt des Anklickens; kann den Wert einer der **sb_xxx-Konstanten** haben, die für vertikale Bildlaufleisten zutreffen.
lParamLo	Nicht verwendet
lParamHi	Handle der Bildlaufleiste; lParamHi ist 0, wenn diese zusammen mit dem Fenster mit dem Stil **ws_VScroll** erzeugt wurde.

Rückgabewert

Nicht verwendet

Hinweise

Führt eine Anwendung einen Bildlauf in einem Fenster durch, muß sie die Positionsmarke mit Hilfe der Funktion **SetScrollPos** zurücksetzen.

Siehe auch:

sb_xxx-Konstanten

wm_VScrollClipboard

Beschreibung

Diese Botschaft wird übergeben, wenn die Zwischenablage ein Daten-Handle des Formats **cf_OwnerDisplay** (d.h. der Besitzer der Zwischenablage sollte den Inhalt der Zwischenablage darstellen) enthält und ein Ereignis in der vertikalen Bildlaufleiste der Zwischenablage auftritt.

Parameter

wParam	Enthält das Handle des Fensters der Zwischenablage.
lParamLo	Enthält einen der folgenden Bildlaufleistencodes im **sb_xxx-Konstanten**
lParamHi	Nicht verwendet

Rückgabewert

Nicht verwendet

Hinweise

Der Besitzer der Zwischenablage sollte die Funktion **InvalidateRect** verwenden oder die Zwischenablage wie gewünscht neu darstellen. Die Bildlaufleistenposition sollte ebenfalls mit **SetScrollPos** zurückgesetzt werden.

Diese Botschaften werden an den Besitzer der Zwischenablage gesendet, wenn das Format **cf_OwnerDisplay** ist:

wm_AskCBFormatName
wm_HScrollClipboard
wm_PaintClipboard
wm_SizeClipboard
wm_VScrollClipboard

Daten und Format der Zwischenablage werden mit der Funktion **SetClipboardData** gesetzt.

Siehe auch:

sb_xxx-Konstanten

wm_wininichange

Beschreibung

Diese Botschaft wird übergeben, wenn die Initialisierungsdatei WIN.INI verändert wird.

Parameter

wParam	Nicht verwendet
lParam	Zeigt auf einen String, der den Namen des geänderten Abschnitts angibt.

Rückgabewert

Nicht verwendet

Hinweise

Die Anwendung kann die Funktion **SendMessage** verwenden, um die Botschaft an alle Hauptfenster zu übergeben. Dabei muß der Parameter Wnd auf $FFF gesetzt sein.

Unit System

Die Unit System (SYSTEM.TPU) ist die Laufzeitbibliothek von Turbo Pascal. Sie enthält die Unterstützung für alle Low-level-Laufzeit-Routinen, wie Datei-Ein-/Ausgabe, String-Verarbeitung, Gleitkomma und dynamische Speicherzuweisung.

Alle Units und Programme verwenden die Unit System. Deshalb brauchen Sie nicht in einer **uses** Klausel darauf zu verweisen.

Siehe auch
Prozeduren und Funktionen
Variablen und typisierte Konstanten

Index der System-Prozeduren und -Funktionen

Dieser Index ist eine alphabetische Liste aller Prozeduren und Funktionen in der Unit System (SYSTEM.TPU).

Sie können diese Prozeduren und Funktionen auch im **kategorischen Index** nachschlagen, wo sie nach Funktionskategorien aufgelistet sind.

Abs	Funktion	Liefert den absoluten Wert des Arguments zurück.
Addr	Funktion	Liefert die Adresse des angegebenen Objekts.
Append	Prozedur	Öffnet eine exisierende Datei für das Anhängen weiterer Daten.
ArcTan	Funktion	Liefert den Arcustangens des Arguments zurück.
Assign	Prozedur	Ordnet einer Datei-Variablen eine externe Datei zu.
BlockRead	Prozedur	Liest einen oder mehrere Records in eine Puffervariable.
BlockWrite	Prozedur	Schreibt einen oder mehrere Records aus einer Puffervariablen.
Chr	Funktion	Liefert das Zeichen, dessen ASCII-Code dem Wert von x entspricht.
Close	Prozedur	Schließt eine offene Datei.
Concat	Funktion	Verbindet zwei oder mehrere Stringteile.
Copy	Funktion	Liefert einen Teil eines Strings zurück.
Cos	Funktion	Liefert den Cosinus des Arguments zurück. (x ist der Winkel im Bogenmaß).
CSeg	Funktion	Liefert die Adresse des aktuellen Code-Segments (CS-Register) zurück.
Dec	Prozedur	Zählt eine Variable um einen bestimmten Wert herunter.
Delete	Prozedur	Löscht count Zeichen ab der Position index im String s.
Dispose	Prozedur	Gibt den einer Zeigervariablen zugeordneten Speicherplatz auf dem Heap wieder frei.
DSeg	Funktion	Liefert die Adresse des Daten-Segments (d.h. den Inhalt des DS-Registers) zurück.
Eof	Funktion	Prüft, ob das Ende der Datei erreicht ist.
Eoln	Funktion	Prüft, ob das Zeilenende in einer Textdatei erreicht ist.
Erase	Prozedur	Löscht eine externe Datei.
Exit	Prozedur	Verläßt den momentanen Block mit sofortiger Wirkung.
Exp	Funktion	Liefert den Exponenten des Arguments zurück.
FilePos	Funktion	Liefert die aktuelle Position innerhalb einer Datei.
FileSize	Funktion	Liefert die Größe einer Datei zurück.
FillChar	Prozedur	Füllt eine bestimmte Anzahl (count) aufeinanderfolgender Speicherzellen mit dem Wert ch (vom Typ Byte oder Char).
Flush	Prozedur	Erzwingt das physikalische Schreiben des Puffers einer Textdatei, die für Ausgaben geöffnet wurde.
Frac	Funktion	Liefert den nicht-ganzzahligen Teil des Arguments zurück.
FreeMem	Prozedur	Gibt einen Speicherbereich bestimmter Größe auf dem Heap wieder frei.
GetDir	Prozedur	Ermittelt das momentan gesetzte Verzeichnis eines Laufwerks.
GetMem	Prozedur	Belegt einen Bereich von size Bytes auf dem Heap und weist die Startadresse dieses Bereichs der als p übergebenen Variablen zu, erzeugt also eine dynamische Variable.
Halt	Prozedur	Bricht die Ausführung des Programms ab und führt

einen Rücksprung zur DOS-Kommandoebene bzw. dem Programm durch, von dem aus dieses Programm gestartet wurde.

Hi	Funktion	Liefert das höherwertige Byte des Arguments zurück.
Inc	Prozedur	Erhöht eine Variable um 1 bzw. um den angegebenen Wert.
Insert	Prozedur	Fügt den als "source" übergebenen Stringausdruck ab der Position "index" in die als "s" übergebene Stringvariable ein.
Int	Funktion	Liefert den ganzzahligen Anteil des Arguments zurück.
IOResult	Funktion	Liefert den Status der letzten Ein-/Ausgabe-Operation als Integer zurück.
Length	Funktion	liefert die aktuelle Länge eines Stringausdrucks zurück.
Ln	Funktion	Liefert den natürlichen Logarithmus des Arguments.
Lo	Funktion	Liefert das niederwertige Byte des Arguments zurück.
MaxAvail	Funktion	Liefert die Umfang des größten freien Blocks auf dem Heap in Bytes zurück.
MemAvail	Funktion	Liefert den Gesamtumfang des freien Platzes auf dem Heap in Bytes.
MkDir	Prozedur	Erzeugt ein Unterverzeichnis.
Move	Prozedur	Kopiert Bytes von der Quelle zum Ziel.
New	Prozedur	erzeugt eine dynamische Variable und setzt eine Zeigervariable auf die Startadresse dieses Bereichs.
Odd	Funktion	Prüft, ob das Argument eine ungerade Zahl darstellt.
Ofs	Funktion	Liefert den Offset-Anteil der Adresse des angegebenen Objekts zurück.
Ord	Funktion	Liefert die Ordinalzahl des Arguments zurück.
ParamCount	Funktion	Liefert die Anzahl der Kommandozeilen-Parameter zurück, die dem Programm übergeben wurden.
ParamStr	Funktion	Liefert den Kommandozeilen-Parameter [Index] zurück.
Pi	Funktion	Liefert den Wert der mathematischen Konstanten Pi zurück.
Pos	Funktion	Sucht den als s übergebenen String-Ausdruck nach dem ersten Vorkommen des String-Ausdrucks substr ab.
Pred	Funktion	Liefert den Vorgänger des Arguments zurück.
Ptr	Funktion	Konvertiert zwei Angaben für Segment und Offset in einen Wert des Typs Pointer.
Random	Funktion	Liefert eine Zufallszahl.
Randomize	Prozedur	Initialisiert den "Zufallsgenerator" mit einem Wert, der sich aus Datum und Uhrzeit des Systems ergibt.
Read	Prozedur	Bei typisierten Dateien: Liest eine oder mehrere Komponenten in eine Variable ein. Bei Textdateien: Liest einen oder mehrere Werte in ein oder mehrere Variablen ein.
ReadLn	Prozedur	Führt einen Aufruf von Read aus und springt dann zum Anfang der nächsten Zeile.
Rename	Prozedur	Gibt einer externen Datei einen neuen Namen.
Reset	Prozedur	Öffnet eine existierende Datei.
Rewrite	Prozedur	Erzeugt eine neue Datei und öffnet sie.
Round	Funktion	Rundet das (Real-)Argument auf einen ganzzahligen Wert.
RunError	Prozedur	Erzeugt einen Laufzeitfehler, der das Programm definiert abbricht.
Seek	Prozedur	Speichert die aktuelle Position einer Datei in einer bestimmten Komponente.
SeekEof	Funktion	Prüft, ob sich zwischen der momentanen Position und dem Dateiende noch "lesbare" Daten befinden.
SeekEoln	Funktion	Prüft, ob sich zwischen der momentanen Position und dem nächsten Zeilenende "lesbare" Daten befinden.

Seg	Funktion	Liefert die Segmentadresse des angegebenen Objekts zurück.
SetTextBuf	Prozedur	Weist einer Textdatei-Variablen einen Puffer zu.
Sin	Funktion	Liefert den Sinus des Arguments.
SizeOf	Funktion	Liefert die Anzahl von Bytes zurück, die das Argument an Speicherplatz belegt.
SPtr	Funktion	Liefert den momentanen Wert des Stackzeigers (SP-Register) zurück.
Sqr	Funktion	Liefert das Quadrat des Arguments zurück.
Sqrt	Funktion	Liefert die Quadratwurzel des Arguments zurück.
SSeg	Funktion	Liefert die Adresse des Stack-Segments zurück, d.h. den Wert des SS-Registers.
Str	Prozedur	Konvertiert einen numerischen Wert in eine Zeichenfolge.
Succ	Funktion	Liefert den Nachfolger des Arguments zurück.
Swap	Funktion	Vertauscht das niederwertige und das höherwertige Byte des Arguments.
Trunc	Funktion	Wandelt einen Realwert durch Abschneiden der Nachkommastellen in einen Integerwert um.
Truncate	Prozedur	Schneidet eine Datei an der aktuellen Position ab.
UpCase	Funktion	Konvertiert Klein- in Großbuchstaben. Die deutschen Umlaute werden nicht berücksichtigt.
Val	Prozedur	Wandelt einen String-Wert in seine numerische Entsprechung um.
Write	Prozedur	Bei typisierten Dateien: Schreibt eine Variable in eine Datei-Komponente. Bei Textdateien: Schreibt einen oder mehrere Werte in die Datei.
WriteLn	Prozedur	Ruft Write (für Textdateien) auf und schreibt dann einen Zeilenvorschub in die Datei oder auf den Bildschirm.

Index der System-Prozeduren und -Funktionen (kategorisch)

Arithmetische Funktionen

- Abs (Funktion)
- ArcTan (Funktion)
- Cos (Funktion)
- Exp (Funktion)
- Frac (Funktion)
- Int (Funktion)
- Ln (Funktion)
- Pi (Funktion)
- Sin (Funktion)
- Sqr (Funktion)
- Sqrt (Funktion)

Prozeduren und Funktionen zur dynamischen Reservierung von Speicherplatz

- Dispose (Prozedur)
- FreeMem (Prozedur)
- GetMem (Prozedur)
- New (Prozedur)
- MaxAvail (Funktion)
- MemAvail (Funktion)

Prozeduren zur Flußsteuerung

- Exit (Prozedur)
- Halt (Prozedur)
- RunError (Prozedur)

Prozeduren und Funktionen zur Ein-/Ausgabe

- Assign (Prozedur)
- ChDir (Prozedur)
- Close (Prozedur)
- Eof (Funktion)
- Erase (Prozedur)
- FilePos (Funktion)
- FileSize (Funktion)
- GetDir (Prozedur)
- IOResult (Funktion)
- MkDir (Prozedur)
- Rename (Prozedur)
- Reset (Prozedur)
- Rewrite (Prozedur)
- RmDir (Prozedur)
- Seek (Prozedur)
- Truncate (Prozedur)

Verschiedene Prozeduren und Funktionen

- FillChar (Prozedur)
- Hi (Funktion)
- Lo (Funktion)
- Move (Prozedur)
- ParamCount (Funktion)
- ParamStr (Funktion)
- Random (Funktion)
- Randomize (Prozedur)

SizeOf (Funktion)
Swap (Funktion)
UpCase (Funktion)

Ordinale Prozeduren und Funktionen
Dec (Prozedur)
Inc (Prozedur)
Odd (Funktion)
Pred (Funktion)
Succ (Funktion)

Zeiger- und Adreßfunktionen
Addr (Funktion)
CSeg (Funktion)
DSeg (Funktion)
Ofs (Funktion)
Ptr (Funktion)
Seg (Funktion)
SPtr (Funktion)
SSeg (Funktion)

String-Prozeduren und Funktionen
Concat (Funktion)
Copy (Funktion)
Delete (Prozedur)
Insert (Prozedur)
Length (Funktion)
Pos (Funktion)
Str (Prozedur)
Val (Prozedur)

Textdatei-Prozeduren und -Funktionen
Append (Prozedur)
Eoln (Funktion)
Flush (Prozedur)
Read (Prozedur)
Readln (Prozedur)
SeekEof (Funktion)
SeekEoln (Funktion)
SetTextBuf (Prozedur
Write (Prozedur)
Writeln (Prozedur)

Transfer-Funktionen
Chr (Funktion)
Ord (Funktion)
Round (Funktion)
Trunc (Funktion)

Prozeduren für untypisierte Dateien
BlockRead (Prozedur)

BlockWrite (Prozedur)

System Unit, Variablen und typisierte Konstanten

Variablen (nicht-initialisiert)

Variable	Typ	Beschreibung
Input	Text	Standard-Eingabedatei
Output	Text	Standard-Ausgabedatei

Konstanten (initialisierte Variablen)

Variable	Typ	Anfangswert	Beschreibung
CmdLine	PChar	**nil**	Kommandozeilenzeiger
CmdShow	Integer	0	CmdShow Parameter für **CreateWindowErrorAddr** Zeiger **nil** Laufzeitfehleradresse
ExitCode	Integer	0	Exit-Code
ExitProc	Zeiger	**nil**	Exit-Prozedur
FileMode	Byte	2	Dateiöffnungsmodus
HeapBlock	Word	8192	Heap-Blockgröße
HeapError	Word	**nil**	Heap-Fehlerfunktion
HeapLimit	Word	1024	Heap-Grenze
HeapList	Word	0	Heap-Segmentliste
HInstance	Word	0	Handle dieser Instanz
HPrevInst	Word	0	Handle der vorigen Instanz
InOutRes	Integer	0	I/O-Ergebnispuffer
PrefixSeg	Word	0	Programmsegmentpräfix
RandSeed	LongInt	0	Startzahl des Zufallsgenerators

Input und Output (Variablen) **(Unit System)** Input und Output sind Standard-I/O-Dateien, die jede Pascal-Implementation benötigt.

CmdLine (Variable) **(Unit System)** In einem Programm enthält CmdLine einen Zeiger auf einen null-terminierten String, der Kommandozeilenargumente enthält, die beim Start des Programms spezifiziert wurden.

In einer Bibliothek ist CmdLine undefiniert.

CmdShow (Variable) **(Unit System)** In einem Programm enthält CmdShow den Parameterwert, den Windows für ShowWindow erwartet, wenn das Programm sein Hauptfenster erstellt.

In einer Bibliothek ist CmdShow immer 0.

ErrorAddr, ExitCode und ExitProc (Variablen) (Unit System)

Die Variablen ExitProc, ExitCode und ErrorAddr werden in Exit-Prozeduren verwendet.

Die Zeigervariable ExitProc installiert die Exit-Prozedur, die immer zur Beendigung des Programms aufgerufen wird.

Eine Exit-Prozedur erhält keine Parameter und muß als Far-Prozedur Compileranweisung compiliert werden, damit sie gezwungen ist, das Aufrufmodell **far** zu verwenden.

Wenn eine Exit-Prozedur richtig implementiert ist, wird sie Teil einer Kette solcher Prozeduren. Diese Kette von Prozeduren wird in umgekehrter Folge ihrer Installation ausgeführt.

Sie müssen den aktuellen Inhalt von ExitProc sichern, bevor Sie die Adresse der eigenen Exit-Prozedur übergeben, damit die Kette nicht durchbrochen wird.

Die erste Anweisung Ihrer Exit-Prozedur muß den Wert von ExitProc wiederherstellen.

Eine Exit-Prozedur kann die Ursache des Programmendes durch die Integervariable ExitCode und die Zeigervariable ErrorAddr bestimmen.

- Bei normaler Beendigung des Programms ist ExitCode gleich 0 und ErrorAddr ist **nil**.
- Wurde das Programmende durch einen Aufruf von **Halt** hervorgerufen, enthält ExitCode den an Halt übergebenen Wert, und ErrorAddr ist **nil**.
- Wird das Programmende durch einen Laufzeitfehler hervorgerufen, enthält ExitCode den Fehlercode, und ErrorAddr die Adresse der fehlerhaften Anweisung.

Die letzte Exit-Prozedur (die der Laufzeitbibliothek) schließt die Dateien Input und Output. Ist ErrorAddr nicht **nil**, liefert sie eine Laufzeitfehlermeldung.

Wenn die Laufzeitbibliothek alle Exit-Prozeduren aufgerufen hat, kehrt sie zu Windows zurück und gibt den Wert in ExitCode zurück.

FileMode (Variable) (Unit System)

Die Variable FileMode bestimmt den Zugriffscode, der an DOS weitergegebene wird, wenn typisierte und untypisierte Dateien mit der Prozedur **Reset** geöffnet werden.

Der Vorgabewert von FileMode ist 2. Zuweisen eines anderen Wertes an FileMode bedeutet, daß alle folgenden Aufrufe der Prozedur Reset diesen Modus verwenden.

Der Gültigkeitsbereich der FileMode-Werte hängt von der verwendeten DOS-Version ab. Bei allen Versionen gelten diese Modi:

0 Read only (Nur Lesen)
1 Write only (Nur Schreiben)
2 Read/Write (Lesen und Schreiben)

DOS Version 3.x und neuere Versionen definieren weitere Modi, die vor allem gemeinsamen Zugriff auf Dateien in Netzen betreffen.

HeapBlock, HeapError, HeapLimit und HeapList (Variablen) (Unit System)

Der Heap-Manager verwendet HeapList, HeapLimit, HeapBlock und HeapError, um die dynamische Speicherzuweisung von Turbo Pascal zu implementieren.

Die Variable HeapLimit definiert die Schwelle zwischen "small" und "large" Heap-Blöcken. Die Variable HeapBlock definiert die Größe, die der Heap-Manager verwendet, wenn für eine Speicherreservierung der normale Heap-Bereich nicht ausreicht.

Sie sollten die Werte von HeapLimit und HeapBlock nicht ändern müssen. Falls Sie dies doch tun, sollte HeapBlock mindestens viermal so groß wie HeapLimit sein.

Mit den Variablen HeapError installieren Sie eine Fehlerfunktion für den Heap, die immer aufgerufen wird, wenn der Heap-Manager seine Aufgaben nicht ausführen kann.

HeapError ist ein Zeiger, der auf eine Funktion mit folgendem Kopf zeigt:

```
function HeapFunc (Size: Word): Integer; far;
```

Die Heap-Fehlerfunktion wird installiert, indem der Variablen HeapError eine Adresse zugewiesen wird:

```
HeapError := @@HeapFunc;
```

Die Fehlerfunktion wird immer aufgerufen, wenn ein Aufruf von **New** oder **GetMem** nicht ausgeführt werden kann.

Der Parameter Size enthält die Größe des Blocks, der nicht zugewiesen werden konnte. Die Heap-Fehlerfunktion sollte einen Block von mindestens dieser Größe frei machen.

Vor dem Aufruf der Heap-Fehlerfunktion versucht der Heap-Manager, den Block innerhalb seines freien Sicherheitsbereichs oder durch Aufruf der Windows-Funktion **GlobalAlloc** zuzuweisen.

Die Funktion HeapError gibt folgendes zurück:

0 Scheitern, sofortige Auslösung eines Laufzeitfehlers
1 Scheitern; New oder GetMem geben einen **nil** Zeiger zurück
2 Erfolg; ein neuer Versuch wird gestartet (der wieder die Heap-Fehlerfunktion aufrufen kann).

HInstance und HPrevInst (Variablen) (Unit System)

HInstance enthält das Handle der Instanz des Programms aus der Windows-Umgebung.

In einem Programm enthält HPrevInst das Handle der vorigen Instanz oder 0, falls es keine vorige Instanz gibt.

In einer Bibliothek ist HPrevInst immer 0.

InOutRes (Variable) **(Unit System)** Die integrierten I/O-Routinen speichern mit InOutRes den Wert, der beim nächsten Aufruf von **IOResult** zurückgeben wird.

PrefixSeg (Variable) **(Unit System)** In einem Programm enthält die Variable PrefixSeg den Selektor (Segmentadresse) des Programmsegment-Präfixes (PSP), das von DOS und Windows beim Ablauf des Programms erstellt wurde.

In einer Bibliothek ist PrefixSeg immer 0.

Eine vollständige Beschreibung des PSP finden Sie im Handbuch von DOS und Windows.

RandSeed (Variable) **(Unit System)**RandSeed speichert die Startzahl des Zufallsgenerators.

Durch einen spezifischen Wert für RandSeed kann die Funktion **Random** dazu gebracht werden, eine bestimmte Sequenz von Zufallszahlen immer wieder zu generieren, z.B. für Datenverschlüsselung, Statistik und Simulationen.

Unit Strings

Die Unit Strings unterstützt eine neue Klasse von Zeichenstrings, die **null-terminierten Strings**.

Mit der erweiterten Syntax von Turbo Pascal und der Unit Strings können Ihre Programme null-terminierte Strings verwenden, die vom Windows Application Programming Interface (API) (= Programmieroberfläche) verwendet werden.

Funktionen der Unit Strings

StrCat	Funktion	Hängt die Kopie eines Strings an einen anderen an und gibt den verknüpften String zurück.
StrComp	Funktion	Vergleicht zwei Strings.
StrCopy	Funktion	Kopiert einen String.
StrECopy	Funktion	Kopiert einen String und gibt einen Zeiger auf das Stringende der Kopie zurück.
StrEnd	Funktion	Gibt einen Zeiger auf das Ende eines String zurück.
StrIComp	Funktion	Vergleicht zwei Strings, ohne auf Groß- oder Kleinschreibung zu achten.
StrLCat	Funktion	Hängt Zeichen aus einem String an das Ende eines anderen Strings an und gibt den Ergebnisstring zurück.
StrLComp	Funktion	Vergleicht Strings ohne auf Groß- oder Kleinschreibung zu achten.
StrLCopy	Funktion	Kopiert Zeichen eines Strings in einen anderen.
StrDispose	Funktion	Entfernt den zuvor zugewiesenen String.
StrLen	Funktion	Gibt Zeichenanzahl eines Strings Str zurück.
StrLIComp	Funktion	Vergleicht Strings bis zu einer Maximallänge, ohne auf Groß- oder Kleinschreibung zu achten.
StrLower	Funktion	Konvertiert einen String in Kleinbuchstaben.
StrMove	Funktion	Kopiert Zeichen aus einem String in einen anderen.
StrNew	Funktion	Reserviert Platz für einen String auf dem Heap.
StrPas	Funktion	Konvertiert einen String mit Null am Ende in einen Pascal-String.
StrPos	Funktion	Gibt einen Zeiger auf das erste Vorkommen eines Strings in einem anderen String zurück.
StrPCopy	Funktion	Kopiert einen Pascal-String in einen null-terminierten String.
StrRScan	Funktion	Gibt einen Zeiger auf das erste Vorkommen eines Strings in einem String zurück.
StrScan	Funktion	Gibt einen Zeiger auf das erste Vorkommen eines Zeichens in einem String zurück.
StrUpper	Funktion	Konvertiert einen String in Großbuchstaben.

Null-terminierte Strings

Was ist ein null-terminierter String?

Ein null-terminierter String besteht aus
- einer Folge von Zeichen (nicht Null) gefolgt von NULL (#0).
- maximal 65535 Zeichen.

Verwenden von null-terminierten Strings

Ein null-terminierter String wird als Array von Zeichen mit auf Null basierendem Integer-Indextyp gespeichert, z.B.:

```
array[0..X] of Char
```

wobei
X = ein positiver Integerwert (nicht Null).

Dies wird ein auf Null basierendes Zeichen-Array genannt.

Der Compilerbefehl zur Verwendung der erweiterten Syntax (**$X**) muß eingeschaltet sein, wenn Sie die Unit Strings in Ihrem Programm verwenden.

Unit WinCrt

Die Unit WinCrt implementiert in einem Fenster ein Textfenster, das einem herkömmlichen Textterminal entspricht. Sie brauchen keinen "Windows-spezifischen" Code zu schreiben, wenn Ihr Programm WinCrt verwendet.

WinCrt wird wie andere Units einfach in die Klausel uses eingeschlossen.

Prozeduren und Funktionen
Variablen

WinCrt Variablen

Die Unit WinCrt deklariert verschiedene typisierte Konstanten (die Anfangswerte haben) und eine Variable.

WindowOrg (TPoint)
WindowSize (TPoint)
ScreenSize (TPoint)
Cursor (TPoint)
Origin (TPoint)
InactiveTitle (PChar)
AutoTracking (Boolean)
CheckEOF (Boolean)
CheckBreak (Boolean)
WindowTitle (array [0..79] of Char)

Typisierte Konstanten kann man als Variablen verstehen, da Sie den Anfangswert der meisten verändern können.

WinCrt Funktionen und Prozeduren

AssignCrt	Prozedur	Ordnet einer Datei-Variablen eine externe Datei zu.
ClrEol	Prozedur	Löscht alle Zeichen von der Cursorposition bis zum Zeilenende.
ClrScr	Prozedur	Löscht den Bildschirm bzw. das Textfenster durch Überschreiben mit der momentan gesetzten Hintergrundfarbe, und setzt den Cursor in die obere linke Ecke.
DoneWinCrt	Prozedur	Löscht das CRT-Fenster.
GotoXY	Prozedur	Bewegt den Cursor im virtuellen Bildschirm zu den gegebenen Koordinaten.
InitWinCrt	Prozedur	Erzeugt das CRT-Fenster.
KeyPressed	Funktion	Gibt True zurück, wenn eine Taste gedrückt wurde.
ReadBuf	Funktion	Liest eine Zeile vom CRT-Fenster.
ReadKey	Funktion	Liest ein Zeichen von der Tastatur.
ScrollTo	Prozedur	Verschiebt das CRT-Fenster zur angegebenen Stelle.
TrackCursor	Prozedur	Verschiebt das CRT-Fenster, damit der Cursor sichtbar bleibt.
WhereX	Funktion	Liefert die Spaltenposition des Cursors (relativ zu einem eventuell gesetzten Textfenster) zurück.
WhereY	Funktion	Liefert die Zeilenposition des Cursors (relativ zu einem eventuell gesetzten Textfenster) zurück.
WriteBuf	Prozedur	Schreibt eine Zeichenblock ins CRT-Fenster.
WriteChar	Prozedur	Schreibt ein Zeichen ins CRT-Fenster.

WindowOrg (Variable) (Unit WinCrt)

Legt die Anfangsposition des CRT-Fensters fest.

```
const WindowOrg: TPoint = (X: cw_UseDefault; Y: cw_UseDefault);
```

Die voreingestellte Position ermöglicht es Windows, eine geeignete Position für das CRT-Fenster zu suchen.

Sie können die Anfangsposition verändern, indem Sie den X- und Y-Koordinaten neue Werte zuweisen, bevor das CRT-Fenster erzeugt wird.

WindowSize (Variable) (Unit WinCrt)

Legt die Anfangsgröße des CRT-Fensters fest.

```
const WindowSize: TPoint = (X: cw_UseDefault; Y: cw_UseDefault);
```

Die voreingestellte Größe ermöglicht es Windows, eine geeignete Größe für das CRT-Fenster zu wählen.

Sie können die Anfangsgröße verändern, indem Sie den X- und Y-Koordinaten neue Werte zuweisen bevor das CRT-Fenster erzeugt wird.

ScreenSize (Variable) (Unit WinCrt)

Legt die Breite und Höhe (in Zeichen) des virtuellen Bildschirms im CRT-Fenster fest.

```
const ScreenSize: TPoint = (X: 80; Y: 25);
```

Die voreingestellte Bildschirmgröße ist 80 Spalten x 25 Zeilen.

Sie können die Größe des virtuellen Bildschirms verändern, indem Sie den X- und Y-Koordinaten von ScreenSize andere Werte zuweisen bevor das CRT-Fenster erzeugt wird.

Das Ergebniss von ScreenSize.X multipliziert mit ScreenSize.Y darf 65 520 nicht überschreiten.

Cursor (Variable) (Unit WinCrt)

Enhält die aktuelle Position des Cursors im virtuellen Bildschirm.

```
const Cursor: TPoint = (X: 0; Y: 0);
```

Die obere linke Ecke entspricht (0, 0). Cursor ist eine Read-Only-Variable; weisen Sie ihr also keine Werte zu.

Origin (Variable) (Unit WinCrt)

Enthält die Koordinaten des virtuellen Bildschirms der Zeichenzelle, die in der oberen linken Ecke des CRT-Fensters angezeigt wird.

```
const Origin: TPoint = (X: 0; Y: 0);
```

Origin ist eine Read-Only Variable; weisen Sie ihr also keine Werte zu.

InactiveTitle (Variable) (Unit WinCrt)

Zeigt auf einen null-terminierten String, der verwendet wird, um den Titel eines inaktiven CRT-Fensters zu erstellen.

```
const InactiveTitle: PChar = '(Inactive %s)';
```

Der String wird als der Format-Kontrollparameter eines Aufrufs an die Windows-Funktion **WVSPrintF** verwendet. Die Formatanweisung %s gibt, falls vorhanden, an, wo der existierende Fenstertitel eingefügt werden muß.

AutoTracking (Variable) (Unit WinCrt)

Aktiviert bzw. inaktiviert das automatische Rollen des Fensters, damit der Cursor sichtbar bleibt.

```
const AutoTracking: Boolean = True;
```

Wenn AutoTracking als True definiert ist, wird das CRT-Fenster automatisch gerollt, um sicherzustellen, daß der Cursor nach jedem Aufruf von **Write** und **Writeln** sichtbar bleibt.

Wenn AutoTracking als False, definiert ist, wird das CRT-Fenster nicht automatisch gerollt. Ins Fenster geschriebener Text ist möglicherweise nicht immer sichtbar.

CheckEOF (Variable) (Unit WinCrt)

Aktiviert bzw. inaktiviert das Dateiende-Zeichen (EOF).

```
const CheckEOF: Boolean = False;
```

Wenn CheckEOF als True definiert ist, wird eine Dateiendemarkierung erzeugt, wenn der Benutzer die Tasten Strg+Z drückt, während er eine Datei liest, die dem CRT-Fenster zugewiesen ist.

Wenn CheckEOF als False, definiert ist, hat das Drücken von Strg+Z keine Auswirkungen.

CheckBreak (Variable) (Unit WinCrt)

Aktiviert bzw. inaktiviert die Möglichkeit des Benutzers, eine Anwendung abzubrechen.

```
const CheckBreak: Boolean = True;
```

Wenn CheckBreak als True, definert ist, kann der Benutzer eine Anwendung jederzeit abbrechen, indem er bzw. sie:
- den Befehl Schließen im Systemmenü des CRT-Fensters wählt oder
- zweimal auf das Systemmenüsymbol des Fensters klickt oder
- Alt+F4 drückt.

Der Benutzer kann auch jederzeit die Tasten Strg+C oder Strg+Untbr drücken, um eine Anwendung zu unterbrechen und das CRT-Fenster zu deaktivieren.

Diese Funktionen sind deaktiviert, wenn CheckBreak als False definiert ist.

WindowTitle (Variable) (Unit WinCrt)

Legt den Titel des CRT-Fensters fest.

```
var WindowTitle: array[0..79] of Char;
```

Der voreingestellte Wert ist die vollständige Pfadangabe der .EXE-Datei des Programms.

Sie können den Titel verändern, indem Sie in WindowTitle einen neuen String speichern, bevor das CRT-Fenster erzeugt wird.

Hier ein Beispiel:

```
StrCopy(WindowTitle, 'Hallo Erdling!');
```

Unit WinDos

Die Unit WinDos implementiert Routinen für das Betriebssystem und zur Dateiverarbeitung. Keine dieser Routinen ist in Standard-Pascal definiert.

Konstanten
Prozeduren und Funktionen
Typen
Variablen

Index der WinDos-Prozeduren und Funktionen (alphabetisch)

Dieser Index ist eine alphabetische Liste aller Prozeduren und Funktionen der Unit WinDos.

Sie finden die WinDos-Routinen nach deren Funktionalität geordnet im **kategorischen Index der WinDos-Prozeduren und -Funktionen**.

CreateDir	Prozedur	Erzeugt ein neues Unterverzeichnis.
DiskFree	Funktion	Liefert den freien Speicherplatz auf dem angegebenen Laufwerk in Bytes.
DiskSize	Funktion	Liefert die Gesamtkapazität des angegebenen Laufwerks in Bytes.
DosVersion	Funktion	Liefert die Versionsnummer von DOS in Form eines Word-Wertes.
FileExpand	Funktion	Expandiert einen Dateinamen.
FindFirst	Prozedur	Sucht das momentane bzw. durch Path angegebene Verzeichnis nach einem festgelegten Attribut ab.
FindNext	Prozedur	Setzt eine mit FindFirst begonnene Suche fort.
FileSearch	Funktion	Sucht nach einer Datei.
FileSplit	Prozedur	Teilt einen Dateinamen in seine drei Komponenten auf.
GetArgCount	Funktion	Gibt die Zahl der Parameter zurück, die aus der Kommandozeile dem Programm übergeben wurden.
GetArgStr	Funktion	Gibt die Kommandozeilenparameter aus Index zurück.
GetCBreak	Prozedur	Ermittelt, bei welchen Operationen DOS auf Ctrl-Break prüft.
GetCurDir	Funktion	Gibt das aktuelle Verzeichnis des Laufwerks zurück.
GetDate	Prozedur	Ermittelt das momentan gesetzte Kalenderdatum des Systems.
GetEnvVar	Funktion	Gibt einen Zeiger auf den Wert einer bestimmten Umgebungsvariablen zurück.
GetFAttr	Prozedur	Liefert die Attribute einer Datei zurück.
GetFTime	Prozedur	Liefert Datum und Uhrzeit der letzten Veränderung einer Datei.
GetIntVec	Prozedur	Ermittelt den Zeiger auf die Interrupt-Behandlungsroutine des Interrupts IntNo.
GetTime	Prozedur	Ermittelt die momentan gesetzte Uhrzeit des Systems.
GetVerify	Prozedur	Kopiert das DOS-Flag Verify in die übergebene Variable Verify.
Intr	Prozedur	Führt den durch IntNo angegebenen Software-Interrupt aus.
MsDos	Prozedur	Führt einen DOS-Funktionsaufruf aus.
PackTime	Prozedur	Konvertiert einen Record des Typs DateTime in das gepackte Datums-/Uhrzeitformat von DOS.
RemoveDir	Prozedur	Entfernt ein leeres Unterverzeichnis.
SetCBreak	Prozedur	Setzt das Break-Flag von DOS mit dem als Break übergebenen Wert.
SetCurDir	Prozedur	Ändert das aktuelle Verzeichnis zum angegebenen Pfad.
SetDate	Prozedur	Setzt das Kalenderdatum des Betriebssystems.
SetFAttr	Prozedur	Setzt die Attribute einer Datei.
SetFTime	Prozedur	Setzt Datum und Uhrzeit der letzten Veränderung einer Datei.
SetIntVec	Prozedur	Setzt einen angegebenen Interrupt-Vektor des Systems auf die Adresse, auf die der als Vector übergebene Zeiger zeigt.
SetTime	Prozedur	Setzt die Uhrzeit des Systems.
SetVerify	Prozedur	Setzt das Verify-Flag von DOS.
UnpackTime	Prozedur	Wandelt ein Longint-Wert in einen Record um.

WinDos-Konstanten

Flag-Konstanten
Dateimodus-Konstanten
Dateiattribute-Konstanten
Dateinamenskomponenten Stringlängen
FileSplit-Return-Flags

WinDos-Typen

- Datei-Record-Typen
- TRegisters
- TDateTime
- TSearchRec

WinDos Variablen

DosError

Index der WinDos-Prozeduren und -Funktionen (kategorisch)

Dieser Index ist eine funktionale (kategorische) Liste aller Prozeduren und Funktionen der Unit WinDos.

Um einzelne WinDos-Routinen namentlich zu suchen, schlagen Sie im **alphabetischen Index der WinDos-Prozeduren und -Funktionen** nach.

Datum- und Zeitprozeduren

GetDate (Prozedur)
GetFTime (Prozedur)
GetTime (Prozedur)
PackTime (Prozedur)
SetDate (Prozedur)
SetFTime (Prozedur)
SetTime (Prozedur)
UnpackTime (Prozedur)

Prozeduren und Funktionen für den Zugriff auf Verzeichnisse

CreateDir (Prozedur)
GetCurDir (Funktion)
RemoveDir (Prozedur)
SetCurDir (Prozedur)

Festplattenstatus-Funktionen

DiskFree (Funktion)
DiskSize (Funktion)

Umgebungsfunktionen

GetArgCount (Funktion)
GetArgStr (Funktion)
GetEnvVar (Funktion)

Prozeduren und Funktionen für Dateien

FileExpand (Funktion)
FileSearch (Funktion)
FileSplit (Funktion)
FindFirst (Prozedur)
FindNext (Prozedur)
GetFAttr (Prozedur)
SetFAttr (Prozedur)

Interrupt-Support-Prozeduren

GetIntVec (Prozedur)
Intr (Prozedur)
MsDos (Prozedur)
SetIntVec (Prozedur)

Verschiedene Prozeduren und Funktionen

DosVersion (Funktion)
GetCBreak (Prozedur)
GetVerify (Prozedur)
SetCBreak (Prozedur)
SetVerify (Prozedur)

Flag-Konstanten (Unit WinDos)

Die Flag-Konstanten testen einzelne Flag-Bits des Flag-Registers nach einem Aufruf von **Intr** oder **MsDos**.

Konstante	Wert
fCarry	$0001
fParity	$0004
fAuxiliary	$0010
fZero	$0040
fSign	$0080
fOverflow	$0800

Dateimodus-Konstanten (Unit WinDos)

Prozeduren für den Zugriff auf Dateien benutzen fmXXX-Konstanten, wenn Dateien auf Festplatten geöffnet oder geschlossen werden.

Die Mode-Felder der Turbo-Pascal-Variablen enthalten einen dieser Werte:

Konstante	Wert
fmClosed	$D7B0
fmInput	$D7B1
fmOutput	$D7B2
fmInOut	$D7B3

Dateiattribute-Konstanten (Unit WinDos)

Die faXXX-Konstanten testen, belegen und löschen Dateiattribute-Bits in Verbindung mit den Prozeduren **GetFAttr**, **SetFAttr**, **FindFirst** und **FindNext.**

Die faXXX-Konstanten wirken additiv. Die Konstante faAnyFile ist die Summe aller Attribute.

Konstante	Wert
faReadOnly	$01
faHidden	$02
faSysFile	$04
faVolumeID	$08
faDirectory	$10
faArchive	$20
faAnyFile	$3F

Dateinamen-Komponenten Stringlängen-Konstanten (Unit WinDos)

Die fsXXX-Konstanten bezeichnen die maximale Stringlänge der Dateinamens-komponenten, die von den Funktionen **FileSearch** und **FileExpand** verwendet werden.

Konstante	Wert
fsPathName	79
fsDirectory	67
fsFileName	8
fsExtension	4

FileSplit-Return-Flag-Konstanten (Unit WinDos)

Die fcXXX-Konstanten werden von der Funktion **FileSplit** verwendet.

Der Rückgabewert ist eine Kombination der Bitmasken fcDirectory, fcFileName und fcExtension. Der Wert zeigt an, welche Komponenten im Pfad enthalten sind. Wenn die Dateierweiterung Jokerzeichen (* oder ?) enthält, wird das Flag fcWildcards gesetzt.

Konstante	Wert
fcExtension	$0001
fcFileName	$0002
fcDirectory	$0004
fcWildcards	$0008

Datei-Record (Typ) (Unit WinDos)

Auch von Turbo Pascal intern genutzte Record-Definitionen sind in der Unit WinDos deklariert.

TFileRec wird sowohl für typisierte als auch untypisierte Dateien verwendet.

TTextRec ist das interne Format einer Variablen vom Typ text.

```
type
{ Typisierte und untypisierte Dateien }
  TFileRec = record
    Handle: Word;
    Mode: Word;
    RecSize: Word;
    Private: array[1..26] of Byte;
    UserData: array[1..16] of Byte;
    Name: array[0..79] of Char;
  end;

{ Textdatei-Record }
  PTextBuf = ^TTextBuf;
  TTextBuf = array[0..127] of Char;
  TTextRec = record
    Handle: Word;
    Mode: Word;
    BufSize: Word;
    Private: Word;
    BufPos: Word;
    BufEnd: Word;
    BufPtr: ^TTextBuf;
    OpenFunc: Pointer;
    InOutFunc: Pointer;
    FlushFunc: Pointer;
    CloseFunc: Pointer;
    UserData: array[1..16] of Byte;
    Name: array[0..79] of Char;
    Buffer: TTextBuf;
  end;
```

TRegisters (Typ) (Unit WinDos)

Die Prozeduren **Intr** und **MsDos** verwenden Variablen vom Typ TRegisters, um den Inhalt des Eingabe-Registers anzugeben und den des Ausgabe-Registers eines Software-Interrrupts zu untersuchen.

```
type
  TRegisters = record
    case Integer of
    0: (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags: Word);
    1: (AL,AH,BL,BH,CL,CH,DL,DH: Byte);
  end;
```

Beachten Sie, daß ein Variantenrecord verwendet wird, um das 8-Bit-Register auf seine 16-Bit-Äuquivalente abzubilden.

TDateTime (Typ) (Unit WinDos)

Variablen des Typs TDateTime werden zusammen mit den Prozeduren **UnpackTime** und **PackTime** verwendet, um gepackte 4-Byte Datum-und Zeitwerte für die Prozeduren **GetFTime**, **SetFTime**, **FindFirst** und **FindNext** zu untersuchen und aufzubauen.

```
type
  TDateTime = record
    Year,Month,Day,Hour,Min,Sec: Word;
  end;
```

Gültige Wertebereiche:

Year	1980..2099
Month	1..12/
Day	1..31
Hour	0..23
Min	0..59
Sec	0..59

TSearchRec (Typ) (Unit WinDos)

Die Prozeduren **FindFirst** und **FindNext** benutzen Variablen vom Typ TSearchRec, um Verzeichnisse zu durchsuchen.

```
type
  TSearchRec = record
    Fill: array[1..21] of Byte;
    Attr: Byte;
    Time: Longint;
    Size: Longint;
    Name: array[0..12] of Char;
  end;
```

Informationen über jede Datei, die von einer dieser Prozeduren gefunden worden ist, wird in TSearchRec zurückgegeben.

- Attr enthält die Dateiattribute (aufgebaut aus Dateiattribute-Konstanten)
- Time enthält gepackt Datum und Zeit (benutzen Sie **UnpackTime** zum konvertieren)
- Size enthält die Größe in Bytes
- Name enthält den Namen
- Fill ist von DOS reserviert und sollte nie geändert werden.

DosError (Variable) (Unit WinDos)

DosError wird von vielen Routinesn der Unit WinDos verwendet, um Fehler zu melden.

```
var DosError: Integer;
```

Die in DosError gespeicherten Werte sind DOS-Fehlercodes.

Der Wert 0 zeigt an, daß kein Fehler aufgetreten ist. Andere mögliche Fehlercodes:

Code	Bedeutung
2	Datei nicht gefunden
3	Pfad nicht gefunden
5	Zugriff verweigert
6	Ungültiger Handle
8	Nicht genug Speicher
10	Ungültige Umgebung
11	Ungültiges Format
18	Keine weiteren Dateien

Unit WObjects

Alle Elemente von ObjectWindows werden in der Unit WObjects deklariert.

Die Unit WObjects ist in jedem ObjectWindows-Programm nötig.

Konstanten
Objekte
Prozeduren und Funktionen
Records
Typen
Variablen

Felder eines Objekts oder einer Methode werden im Hilfebildschirm dieses Objekts behandelt.

Siehe auch

Unit WinTypes
Unit WinProcs

ObjectWindows Konstanten

bf_XXXX Konstanten
cm_XXXX Konstanten
coXXXX Konstanten
em_XXXX Konstanten
id_XXXX Konstanten
nf_XXXX Konstanten
stXXXX Konstanten
tf_XXXX Konstanten
wb_XXXX Konstanten
wm_XXXX Konstanten

ObjectWindows Prozeduren und Funktionen

Abstract (Prozedur)
AllocMultiSel (Funktion)
FreeMultiSel (Prozedur)
LongDiv (Funktion)
LongMul (Funktion)
LowMemory (Funktion)
MemAlloc (Funktion)
RegisterType (Prozedur)

ObjectWindows Records

- **LongRec**
- **PtrRec**
- **TDialogAttr**
- **TMessage**
- **TMultiSelRec**
- **TStreamRec**
- **TWindowAttr**
- **WordRec**

ObjectWindows Typen

- **PString**
- **TByteArray**
- **TItemList**
- **TWordArray**

Variablen

Application
EmsCurHandle
EmsCurPage
MaxCollectionSize
SafetyPoolSize
StreamError

Application (Variable) (Unit WObjects)

Deklaration

```
Application: PApplication = nil;
```

Bemerkungen

Die Variable Application wird am Beginn von **TApplication.Init** auf @Self gesetzt und von **TApplication.Done**.als **nil** deklariert.

Während der Ausführung eines ObjectWindows-Programms zeigt Application daher auf das Objekt Application.

bf_XXXX Konstanten (Unit WObjects)

Aktionsschalter, Markierungsfelder und Schaltfeld-Objekte definieren mit bf_ Konstanten ihre drei möglichen Zustände.

Werte

Es gibt diese definierten Aktionsschalter-Flag-Konstanten:

Konstante	Wert	Bedeutung
bf_Unchecked	0	Element nicht gewählt
bf_Checked	1	Element gewählt
bf_Grayed	2	Element ist grau

cm_XXXX Konstanten (Unit WObjects)

ObjectWindows definiert Konstanten, die Bereiche von Befehlsmeldungs-Konstanten definieren.

Werte

Folgende Befehls-Konstanten sind definiert:

Konstante	Wert	Bedeutung
cm_First	$A000	Beginn der Befehlsmeldungen
cm_Count	$6000	Zahl der Befehlsmeldungen
cm_Internal	$FF00	Beginn der Befehlsmeldungen für internen Gebrauch
cm_InternalOffset	cm_Internal - cm_First	

cm_ Konstanten sind für drei Standardmenüs definiert:File, Edit, und Window:

Konstante	Wert	Menüäqivalent
cm_EditCut	cm_InternalOffset	Edit|Cut
cm_EditCopy	cm_InternalOffset + 1	Edit|Copy
cm_EditPaste	cm_InternalOffset + 2	Edit|Paste
cm_EditDelete	cm_InternalOffset + 3	Edit|Delete
cm_EditClear	cm_InternalOffset + 4	Edit|Clear
cm_EditUndo	cm_InternalOffset + 5	Edit|Undo
cm_FileNew	cm_InternalOffset + 6	File|New
cm_FileOpen	cm_InternalOffset + 7	File|Open
cm_MDIFileNew	cm_InternalOffset + 8	File|New
cm_MDIFileOpen	cm_InternalOffset + 9	File|Open
cm_FileSave	cm_InternalOffset + 10	File|Save
cm_FileSaveAs	cm_InternalOffset + 11	File|Save As
cm_ArrangeIcons	cm_InternalOffset + 12	Window|Arrange Symbole
cm_TileChildren	cm_InternalOffset + 13	Window|Tile
cm_CascadeChildren	cm_InternalOffset + 14	Window|Cascade
cm_CloseChildren	cm_InternalOffset + 15	Window|Close All

coXXXX Konstanten (Unit WObjects)

Die coXXXX Konstanten werden als Code Parameter an die Methode **TCollection.Error** weitergegeben, wenn **TCollection** einen Fehler entdeckt.

Werte

Folgende Standard-Fehlercodes sind für alle ObjectWindows-Kollectionen definiert:

Fehlercode	Wert	Bedeutung
coIndexError	-1	Index ungültig Der Info Parameter der Methode Error enthält den ungültigen Index.
coOverflow	-2	Kollektionsüberlauf. **TCollection.SetLimit** erweiterte die Kollektion nicht auf die nötige Größe. Der Info Parameter der Methode Error enthält die ungültige Größe.

em_XXXX Konstanten (Unit WObjects)

Diverse Standard-Fehlerbedingungen werden durch ObjectWindows Konstanten, die mit em_ beginnen, angezeigt.

Werte

Folgende Fehler-Flags sind definiert:

Konstante	Wert	Bedeutung
em_OutOfMemory	-1	Speicherzuweisung ging in den safety pool.
em_InvalidClient	-2	MDI-Client-Fenster konnte nicht erstellt werden
em_InvalidChild	-3	Ein oder mehrere untergeordnete Fenster nicht gültig
em_InvalidWindow	-4	Fenster ist ungültig, weil Create erfolglos war
em_InvalidMainWindow	-5	Hauptfenster konnte nicht erstellt werden

EmsCurHandle (Variable) (Unit WObjects)

Deklaration

```
EmsCurHandle: Word = $FFFF;
```

Bemerkungen

Enthält das aktuelle EMS Handle, das in die physikalische EMS page 0 durch **TEmsStream** angelegt wurde.

TEmsStream vermeidet aufwendiges EMS-remapping durch caching des EMS-Status.

Wenn Ihr Programm EMS für andere Zwecke verwendet, müssen Sie EmsCurHandle und **EmsCurPage** auf $FFFF setzen, bevor Sie TEmsStream verwenden -- dies zwingt TEmsStream, sein mapping zu restaurieren.

Siehe auch

TEmsStream.Handle

EmsCurPage (Variable) (Unit WObjects)

Deklaration

```
EmsCurPage: Word = $FFFF;
```

Bemerkungen

Enthält die logische aktuelle EMS page-Nummer, wie sie in die physikalische EMS page 0 durch **TEmsStream** angelegt wurde.

TEmsStream vermeidet aufwendiges EMS-remapping durch caching des EMS-Status.

Wenn Ihr Programm EMS für andere Zwecke verwendet, müssen Sie EmsCurHandle und **EmsCurPage** auf $FFFF setzen, bevor Sie TEmsStream verwenden -- dies zwingt TEmsStream, seine Abbildung zu restaurieren.

Siehe auch

TEmsStream.PageCount

id_XXXX Konstanten (Unit WObjects)

ObjectWindows definiert Konstanten, die Bereiche von ID-Meldungen der untergeordneten Fenster definieren.

Werte

Es gibt folgende Child-ID-Botschaftskonstanten:

Konstante	Wert	Bedeutung
id_First	$8000	Beginn von Child-ID Meldungen
id_Count	$1000	Zahl der Child-ID Meldungen
id_Internal	$8F00	für internen Gebrauch
id_InternalOffset	id_Internal - id_First;	Interner offset
id_FirstMDIChild	id_InternalOffset + 1	Basis für Child-ID Nummern
id_MDIClient	id_InternalOffset + 2	Child-ID-Nummer des MDI-Client -Fensters

LongRec (Record) (Unit WObjects)

Deklaration

```
LongRec = record
   Lo, Hi: Word;
end;
```

Bemerkungen

Recordtyp für Variablen der Länge double word.

MaxCollectionSize (Variable) (Unit WObjects)

Deklaration

```
MaxCollectionSize = 65520 div SizeOf(Pointer);
```

Bemerkungen

MaxCollectionSize bestimmt die Maximalzahl von Elementen einer Kollektion, die auch die Zahl von Zeigern ist, welche in ein 64K Speichersegment passen.

nf_XXXX Konstanten (Unit WObjects)

ObjectWindows definiert Konstanten für Bereiche von Informationsmeldungen.

Werte

Folgende Konstanten sind definiert:

Konstante	Wert	Bedeutung
nf_First	$9000	Beginn der Nachricht
nf_Count	$1000	Zahl der Nachrichten
nf_Internal	$9F00	Beginn der Nachrichten zum internen Gebrauch

PString (Typ) (Unit WObjects)

Deklaration

```
PString = ^String;
```

Bemerkungen

Definiert einen Zeiger auf einen Pascal String.

PtrRec (Record) (Unit WObjects)

Deklaration

```
PtrRec = record
   Ofs, Seg: Word;
end;
```

Bemerkungen

Ein Record mit Offset- und Segment-Werten eines Zeigers.

SafetyPoolSize (Variable) (Unit WObjects)

Deklaration

```
SafetyPoolSize: Word = 8192;
```

Bemerkungen

Definiert die Größe des Sicherheitsbereichs im Heap.

Der Sicherheitsbereich ist ein Puffer am oberen Ende des Heap, der sicherstellt, daß Speicherzuweisungen nicht scheitern.

Siehe auch

LowMemory
MemAlloc
TApplication.ValidWindow

stXXXX Konstanten (Unit WObjects)

Es gibt zwie Konstantengruppen mit "st", die für das ObjectWindows Streamssystem verwendet werden.

Werte

Die Konstanten für den Streamzugriffmodus werden von **TDosStream** und **TBufStream** verwendet, um den Zugriffsmodus einer für einen ObjectWindows Stream geöffneten Datei zu bestimmen:

Konstante	Wert	Bedeutung
stCreate	$3C00	Erzeuge neue Datei
stOpenRead	$3D00	Öffne existierende Datei, nur Lesen
stOpenWrite	$3D01	Öffne existierende Datei, nur Schreiben
stOpen	$3D02	Öffne existierende Datei, Lesen und Schreiben

Folgenden Werte werden von **TStream.Error** im Feld **TStream.ErrorInfo** zurückgegeben, wenn ein Stream-Fehler auftritt:

Fehlercode	Wert	Bedeutung
stOk	0	Kein Fehler
stError	-1	Zugriffsfehler
stInitError	-2	Kann Stream nicht initialisieren
stReadError	-3	Über Ende des Stream hinaus gelesen
stWriteError	-4	Kann Stream nicht expandieren
stGetError	-5	Get von unregistriertem Objekttyp
stPutError	-6	Put von unregistriertem Objekttyp

Siehe auch

TStream

StreamError (Variable) (Unit WObjects)

Deklaration

```
StreamError: Pointer = nil;
```

Bemerkungen

Wenn nicht **nil**, zeigt StreamError auf eine Prozedur, die von der Methode Error des Streams bei einem Fehler aufgerufen wird.

Die Prozedur muß eine **far** Prozedur mit einem **var** Parameter sein, der ein **TStream** ist. Die Prozedur muß so deklariert sein:

```
procedure MyStreamErrorProc(var S: TStream); far;
```

StreamError ermöglicht, alle Stream-Fehlerverarbeitung zu überschreiben.

Um die Fehlerverarbeitung für einen bestimmten Stream-Typ zu überschreiben, sollten Sie die Methode Error dieses Stream-Typs überschreiben.

TByteArray (Array-Typ) (Unit WObjects)

Deklaration

```
TByteArray = array[0..32767] of Byte;
```

Bemerkungen

Ein Byte-Array-Typ zum allgemeinen Gebrauch bei Typumwandlungen.

TDialogAttr (Record) (Unit WObjects)

Deklaration

```
TDialogAttr = record
   Name: PChar;
   Param: Longint;
end;
```

Bemerkungen

TDialog-Objekte speichern ihre Attribut-Werte in einem Record vom Typ TDialogAttr.

Siehe auch

TDialog.Attr

tf_XXXX Konstanten (Unit WObjects)

Die Methode Transfer verwendet Flag-Konstanten mit tf_.

Werte

Es gibt folgende Transfer Konstanten:

Konstanten	Wert	Bedeutung
tf_SizeData	0	Finde die Datenmenge, die vom Objekt übertragen wurde
tf_GetData	1	Hole Daten vom Objekt ein
tf_SetData	2	Sende Daten, um den Wert des Objekts zu setzen

TItemList (Array-Typ) (Unit WObjects)

Deklaration

```
TItemList = array[0..MaxCollectionSize - 1] of Pointer;
```

Bemerkungen

Array von generischen Zeigern, die intern von **TCollection** Objekten verwendet werden.

TMessage (Record) (Unit WObjects)

Deklaration

```
TMessage = record
   Receiver: HWnd;
   Message: Word;
   case Integer of
     0: (WParam: Word;
         LParam: Longint;
         Result: Longint);
     1: (WParamLo: Byte;
         WParamHi: Byte;
         LParamLo: Word;
         LParamHi: Word;
         ResultLo: Word;
         ResultHi: Word);
   end;
```

Bemerkungen

Die Meldungsschleife in **TApplication** packt Windows-Meldungen in TMessage Records, bevor die Information der passenden Antwortmethode übergeben wird.

Siehe auch

TApplication.MessageLoop

TMultiSelRec (Record) (Unit WObjects)

Deklaration

```
TMultiSelRec = record
   Count: Integer;
   Selections: array[0..0] of Integer;
end;
```

Bemerkungen

TMultiSelRec enthält eine Liste von gewählten Elementen, die zu oder von einem Listenfenster mit mehrfacher Auswahl übertragen werden.

- Count Zahl an gewählten Elementen
- Selections Array von Integern mit offenem Ende

Mit **AllocMultiSel** können Sie einen Record zuweisen, der alle gewählten Elemente eines Listenfensters aufnehmen kann.

Siehe auch

FreeMultiSel

TStreamRec (Record) (Unit WObjects)

Deklaration

```
TStreamRec = record
   ObjType: Word;
   VmtLink: Word;
   Load: Pointer;
   Store: Pointer;
   Next: Word;
end;
```

Bemerkungen

Ein ObjectWindows Objekttyp muß einen registrierten TStreamRec haben, bevor seine Objekte in ein **TStream** Objekt geladen oder gespeichert werden können.

Die Routine **RegisterType** registriert einen Objekttyp durch Einrichten eines TStreamRec Records.

Die Felder des Registrier-Records sind so definiert:

Feld	Inhalt
ObjType	Eindeutige numerische ID für den Objekttyp
VmtLink	Link zum Tabelleneintrag der virtuellen methode des Objekttyps
Load	Zeiger auf den Konstruktor Load des Objekttyps
Store	Zeiger auf die Methode Store des Objekttyps
Next	Zeiger auf den nächsten TStreamRec

ObjectWindows reserviert ID-Werte für Objekttypen (ObjType) 0 bis 999 zum eigenen Gebrauch. Programmierer können ihre Werte im bereich von 1,000 bis 65,535 definieren.

TStreamRec für einen Txxxx Objekttyp wird Rxxxx genannt. TStreamRec für TCalculator wird RCalculator genannt. Siehe den folgenden Code:

```
type
   TCalculator = object(TDialog)
     constructor Load(var S: TStream);
     procedure Store(var S: TStream);
     ...
   end;

const
   RCalculator: TStreamRec = (
     ObjType: 2099;
     VmtLink: Ofs(TypeOf(TCalculator)^);
     Load: @TCalculator.Load;
     Store: @TCalculator.Store);

begin
   RegisterType(RCalculator);
   ...
end;
```

TWindowAttr (Record) (Unit WObjects)

Deklaration

```
TWindowAttr = record
   Title: PChar;
   Style: Longint;
   ExStyle: Longint;
   X, Y, W, H: Integer;
   Param: Pointer;
   case Integer of
     0: (Menu: HMenu);      { window menu's handle or... }
     1: (Id: Integer);      { control's child identifier }
   end;
```

Bemerkungen

TWindow Objekte definieren ihre Attribute in TWindowAttr Records.

Siehe auch

TWindow.Attr

TWordArray (Array-Typ) (Unit WObjects)

Deklaration

```
TWordArray = array[0..16383] of Word;
```

Bemerkungen

Word-Array-Typ zum allgemeinen Gebrauch.

wb_XXXX Konstanten (Unit WObjects)

Das Feld Flags in **TWindowsObject** ist ein Bbitmap-Feld. Auf die Bits kann mit Konstanten mit wb_ zugegriffen werden.

Werte

Folgende Werte sind definiert:

Konstante	Wert	Bedeutung, wenn gesetzt
wb_KeyboardHandler	$01	Fenster behandelt Tastaturereignisse wie Dialog
wb_FromResource	$02	Dialog aus Ressource
wb_AutoCreate	$04	Fenster mit übergeordnetem Fenster erstellt
wb_MDIChild	$08	Fenster ist MDI des untergeordneten Fensters
wb_Transfer	$10	Fenster nimmt im Transfer Mechanismus teil. Als Vorgabe ist diese Bit durch InitResource gesetzt und durch Init gelöscht.

Siehe auch

TWindowsObject.Flags.

wm_XXXX Konstanten (Unit WObjects)

ObjectWindows definiert Konstanten in Bezug auf Windows-Meldungen, die Bereiche für Meldungen von Windows definieren.

Werte

Folgende Fenster-Konstanten sind definiert:

Konstante	Wert	Bedeutung
wm_First	$0000	Beginn der Windows Meldungen
wm_Count	$8000	Zahl der Windows Meldungen

WordRec (Record) (Unit WObjects)

Deklaration

```
WordRec = record
   Lo, Hi: Byte;
end;
```

Bemerkungen

Ein Record, der Zugriff auf Lo und Hi Bytes eines Word gestattet.

Siehe auch

LongRec

AllocMem

Information not available as of 2/11/91.

Unit WinTypes

Die Unit WinTypes definiert Turbo-Pascal-Versionen aller Typen, welche von Windows-API-Routinen verwendet werden, wie **Typen** und **Datenstrukturen (Records)** und alle **Windows-Konstanten**, wie Stile, Botschaften und Flags.

Windows API Records (Unit WinTypes)

TBitmap
TBitmapCoreHeader
TBitmapCoreInfo
TBitmapFileHeader
TBitmapInfo
TBitmapInfoHeader
TCatchBuf
TClientCreateStruct
TCompareItemStruct
TComStat
TCreateStruct
TDCB
TDDEAck
TDDEAdvise
TDDEData
TDDEPoke
TDeleteItemStruct
TDevMode
TDrawItemStruct
THandleTable
TLogBrush
TLogFont
TLogPalette
TLogPen
TMDICreateStruct
TMeasureItemStruct
TMenuItemTemplateHeader
TMetaFilePict
TMetaHeader
TMetaRecord
TMsg
TMultiKeyHelp
TOFStruct
TPaintStruct
TPaletteEntry
TPoint
TRect
TRGBQuad
TRGBTriple
TTextMetric
TWndClass

Windows API Typen (Unit WinTypes)

Bool
HBitmap
HBrush
HCursor
HDC
HFont
HIcon
HMenu
HPalette
HPen
HRgn
HStr
HWnd
LPHandle
LPVoid
MakeIntAtom
MakeIntResource
PBool
PByte
PHandle
PInteger
PLongint
PStr
PWord
TAtom
TColorRef
TFarProc
TGlobalHandle
THandle
TLocalHandle
TPattern

Bool (Typ) (Unit WinTypes)

Deklaration

```
Bool = System.WordBool;
```

Hinweise

Bool entspricht dem Standardtyp WordBool. Er dient zu Kompatibilität mit Windows-Code aus anderen Sprachen.

Dialogfenstervorlage

Eine Dialogfenstervorlage ist eine Windows-Datenstruktur namens DLGTEMPLATE oder TDlgTemplate. Sie ist wie folgt zusammengesetzt:
- einer Kopfzeile (im Beispiel DialogTemplateHeader) (vorgeschrieben)
- Informationen zur im Dialogfenster verwendeten Schrift (optional)
- Informationen zu den einzelnen Elementen des Dialogfensters (optional)

Ein Zeiger auf diese Struktur wird verwendet von:
CreateDialogIndirect
CreateDialogIndirectParam
DialogBoxIndirect
DialogBoxIndirectParam

Kopfzeile der Dialogfenstervorlage

Dialogfenstervorlagen beginnen mit einer Kopfzeile, die wie folgt zusammengesetzt ist:

dtStyle	Stil des Dialogfensters (long integer). Hat den Wert einer oder einer beliebigen Kombination der **ds_xxx Konstanten**.
dtItemCount	Anzahl von Elementen im Dialogfenster (Byte).
dtX, dtY	Koordinaten der oberen linken Ecke des Dialogfensters, relativ zum Ursprung des Klientbereich des übergeordnetes Fensters. (Falls dem Dialogfenster der Stil ds_AbsAlign zugeordnet ist, beziehen sich die Koordinaten auf den Ursprung des Bildschirmfensters.)
dtCX, dtCY	Breite und Höhe des Dialogfensters.

Beispiel für eine Kopfzeile einer Dialogfenstervorlage:

```
type
  DialogTemplateHeader = record
    dtStyle: Longint;
    dtItemCount: Byte;
    dtX, dtY: Integer;
    dtCX, dtCY: Integer;
  end;
```

Nach den Koordinaten und der Größe des Dialogfensters werden drei null-terminierte Strings angegeben:

dtMenuName	Name des Dialogfenstermenüs
dtClassName	Name der Fensterklasse des Dialogfensters
dtCaptionText	Titel des Dialogfensters.

Schriftnformation

Falls dtStyle den Wert ds_SetFont hat, müssen nach dtCaptionText zwei Einträge mit folgenden Informationen zur Schrift folgen:
- die Punktgröße der Schrift (short integer)
- der Name der Schrift (null-terminnierter String)

Vorlagen für die Dialogfensterelemente

Vorlagen für die Dialogfensterelemente werden nach den Schriftinformationen angeben. Zu jedem Steuerelement des Dialogfensters wird eine Vorgage spezifiziert. Die Anzahl der Elemente wird durch dtItemCount bestimmt.

Die Vorlage eines Dialogfensterelements ist wie folgt zusammengesetzt:

dtilX, dtilY, dtilCX, dtilCY	Position und Größe des Steuerelements, relativ zum Ursprung des Dialogfensters (integer)
dtilID	Integerwert, der die ID des Dialogfensterelements angibt
dtilStyle	Stil des Dialogfensterelements (long integer)

Einer der folgenden Werte: BUTTON, EDIT, STATIC, LISTBOX, SCROLLBAR, COMBOBOX	Name der Windows-Klasse des Element (null-terminierter String)
dtilText	Text (null-terminierter String)
dtilInfo	Wert, der die Anzahl von Bytes mit zusätzlichen Informationen angibt (Byte). 0 beim Fehlen zusätzlicher Informationen.
dtilData	Informationen, die im Feld CreateParams einer **TCreateStruct**-Struktur an **CreateWindow** übergeben werden. Wird nur verwendet, falls dtilInfo ungleich 0 ist.

Siehe auch

GetDialogBaseUnits

HBitmap (Typ) (Unit WinTypes)

Deklaration

```
HBitmap = THandle;
```

Hinweise

HBitmap ist ein Handle-Typ für Bitmap-Handles.

HBrush (Typ) (Unit WinTypes)

Deklaration

```
HBrush = THandle;
```

Hinweise

HBrush ist ein Handle-Typ für Zeichenwerkzeuge.

HCursor (Typ) (Unit WinTypes)

Deklaration

```
HCursor  = THandle;
```

Hinweise

HCursor definiert einen Handle-Typ für Cursor-Handles.

HDC (Typ) (Unit WinTypes)

Deklaration

```
HDC = THandle;
```

Hinweise

HDC definiert einen Handle-Typ für Gerätekontext-Handles.

Der Bildschirmkontext ist eine Art von Gerätekontext, deshalb sind Bildschirmkontext-Handles immer in Variablen vomTyp HDC gespeichert.

HFont (Typ) (Unit WinTypes)

Deklaration

```
HFont = THandle;
```

Hinweise

HFont definiert einen Handle-Typ zum Schriftenmalen.

HIcon (Typ) (Unit WinTypes)

Deklaration

```
HIcon = THandle;
```

Hinweise

HIcon definiert einen Handle-Typ für Symbol-Handles.

HMenu (Typ) (Unit WinTypes)

Deklaration

```
HMenu = THandle;
```

Hinweise

HMenu definiert einen Handle-Typ für Menüressourcen.

HPalette (Typ) (Unit WinTypes)

Deklaration

```
HPalette = THandle;
```

Hinweise

HPalette definiert einen Handle-Typ für Paletten-Handles.

HPen (Typ) (Unit WinTypes)

Deklaration

```
HPen = THandle;
```

Hinweise

HPen definiert einen Handle-Typ für Stift-Zeichenwerkzeuge.

HRgn (Typ) (Unit WinTypes)

Deklaration

```
HRgn = THandle;
```

Hinweise

HRgn definiert einen Handle-Typ für Bereichs-Handles.

HStr (Typ) (Unit WinTypes)

Deklaration

```
HStr = THandle;
```

Hinweise

HStr definiert einen Handle-Typ für String-Handles.

HWnd (Typ) (Unit WinTypes)

Deklaration

```
HWnd = THandle;
```

Hinweise

HWnd definiert einen Handle-Typ für Fenster-Handles.

Sie werden normalerweise von ObjectWindows-Objekten zur Kontrolle zugehöriger Windows-Elemente verwendet.

Viele API-Funktionsaufrufe benötigen ein Fenster-Handle zur Bestimmung des Zielfensters.

LPHandle (Typ) (Unit WinTypes)

Deklaration

```
LPHandle = PHandle;
```

Hinweise

LPHandle definiert einen Long-Zeiger auf ein Handle.

Wird von ObjectWindows nicht verwendet, sondern soll lediglich die Kompatibilität mit Windows-Code aus anderen Sprachen sicherstellen.

LPVoid (Typ) (Unit WinTypes)

Deklaration

```
LPVoid = Pointer;
```

Hinweise

LPVoid definiert einen Long-Zeiger. Er wird von ObjectWindows nicht verwendet, sondern soll lediglich die Kompatibilität mit Windows-Code aus anderen Sprachen sicherstellen.

MakeIntAtom (Typ) (Unit WinTypes)

Deklaration

```
MakeIntAtom = PStr;
```

Hinweise

MakeIntAtom wird zur Typumwandlung von Integern in Atome verwendet.

Entspricht der Typumwandlung in den Turbo Pascal-Typ PChar.

MakeIntResource (Typ) (Unit WinTypes)

Deklaration

```
MakeIntResource = PStr;
```

Hinweise

MakeIntResource wird zur Typumwandlung von Integerzahlen in Ressourcen-Namen verwendet.

Entspricht der Typumwandlung in den Turbo Pascal-Typ PChar.

MakePoint (Typ) (Unit WinTypes)

Deklaration

```
MakePoint = TPoint;
```

Hinweise

MakePoint wird zur Typumwandlung von Long Integer-Werten in **TPoint** -Records verwendet.

PBool (Typ) (Unit WinTypes)

Deklaration

```
PBool = ^WordBool;
```

Hinweise

PBool definiert einen Zeiger auf einen 16-Bit-Wert vom Typ Boolean.

PByte (Typ) (Unit WinTypes)

Deklaration

```
PByte = ^Byte;
```

Hinweise

PByte definiert einen Zeiger auf einen 8-Bit-Wert ohne Vorzeichen.

PHandle (Typ) (Unit WinTypes)

Deklaration

```
PHandle = ^THandle;
```

Hinweise

PHandle definiert einen Zeiger aufein generisches Windows-Handle.

PInteger (Typ) (Unit WinTypes)

Deklaration

```
PInteger = ^Integer;
```

Hinweise

PInteger definiert einen Zeiger auf eine 16-Bit Integerzahl mit Vorzeichen.

PLongint (Typ) (Unit WinTypes)

Deklaration

```
PLongint = ^Longint;
```

Hinweise

PLongint definiert einen Zeiger auf eine 32-Bit-Integerzahl.

PStr (Typ) (Unit WinTypes)

Deklaration

```
PStr = PChar;
```

Hinweise

PStr definiert einen Zeiger auf einen null-terminierten String.

Er entspricht dem Turbo-Pascal-Typ PChar und dient zur Kompatibilität mit Code aus anderen Sprachen.

PWord (Typ) (Unit WinTypes)

Deklaration

```
PWord = ^Word;
```

Hinweise

PWord definiert einen Zeiger auf eine 16-Bit-Integerzahl ohne Vorzeichen.

TAtom (Typ) (Unit WinTypes)

Deklaration

```
TAtom = Word;
```

Hinweise

TAtom definiert eine 16-Bit-Zahl, die ein Atom oder eine Meldung identifiziert, welche zwischen DDE-Programmen gesendet werden.

TBitmap (Record) (Unit WinTypes)

Deklaration

```
TBitmap = record
   bmType: Integer;
   bmWidth: Integer;
   bmHeight: Integer;
   bmWidthBytes: Integer;
   bmPlanes: Byte;
   bmBitsPixel: Byte;
   bmBits: Pointer;
end;
```

Hinweise

Der Record TBitmap wird von den Funktionen **CreateBitmapIndirect** und **GetObject** verwendet, um Größe, Farben und Bitwerte eines Bitmap zu beschreiben.

Felder

bmType

Das Feld bmType definiert den Bitmap-Typ. Der Wert 0 steht für ein logisches Bitmap.

bmWidth und bmHeight

bmWidth und bmHeight geben die Breite in Pixeln und die Höhe in Rasterzeilen des Bitmap an. Beide müssen größer als 0 sein.

bmWidthBytes

Das Feld bmWidthBytes definiert die Anzahl von Bytes in jeder Rasterzeile und muß eine gerade Zahl sein.

bmPlanes und bmBitsPixel

bmPlanes und bmBitsPixel gibt die Zahl der Farbebenen und die benachbarten Farb-Bits jeder Ebene an.

bmBits

Das Feld bmBits ist ein Zeiger auf die Bits, aus denen das Bitmap besteht. Die Bits sind als Array von Bytes dargestellt.

TBitmapCoreHeader (Record) (Unit WinTypes)

Deklaration

```
TBitmapCoreHeader = record
   bcSize: Longint;     { used to get to color table }
   bcWidth: Word;
   bcHeight: Word;
   bcPlanes: Word;
   bcBitCount: Word;
end;
```

Hinweise

TBitmapCoreHeader definiert Größe und Farben eines gerätunabhängigen Bitmap. TBitmapCoreHeader-Records werden als Teil von **TBitmapCoreInfo** benutzt, um ein geräteunabhängiges Bitmap vollständig zu definieren.

Felder

bcSize

bcSize enthält die Anzahl von Bytes im Record TBitmapCoreHeader.

bcWidth und bcHeight

bcWidth und bcHeight geben Höhe und Breite des Bitmap in Pixeln an.

bcPlanes

Das Feld bcPlanes enthält die Anzahl der Ebenen im Zielgerät und muß auf 1 gesetzt werden.

bcBitCount

Das Feld bcBitCount enthält die Anzahl von Bits pro Pixel. Zulässige Werte für bcBitCount sind 1, 4, 8 und 24.

Die Bedeutung der einzelnen Bits von bcBitCount ist wie folgt definiert:

- bcBitCount ist 1, das Bitmap ist monochrom, die Farbtabelle muß zwei Einträge enthalten, jedes Bit des Bitmapsentspricht einem Pixel. Das freie Bit stellt die erste Farbe, das gesetzte Bit die zweite Farbe des Bitmaps dar.
- bcBitCount ist 4, das Bitmap hat bis zu 16 Farben, numeriert 0 bis 15, jedes Pixel dersBitmaps entspricht vier Farbbits, die seine Farbe darstellen. Die Farbtabelle enthält 16 Einträge. Jedes Byte des Bitmaps repräsentiert daher zwei Pixel, eines das höherwerttige, das zweite das niederwertige halbe Byte.
- bcBitCount ist 8, das Bitmap hat bis zu 256 Farben, zur Darstellung eines Pixel wird daher jeweils ein Byte benötigt. Jedes Byte des Bitmaps repräsentiert einen Index zwischen 0 und 255 auf die Farbtabelle.
- bcBitCount ist 24, das Bitmap hat bis zu 224 Farben. Es gibt keine Farbtabelle, jedes Pixel wird durch drei Bytes dargestellt, die die Intensität von Rot, Grün und Blau des Pixels enthalten.

TBitmapCoreInfo (Record) (Unit WinTypes)

Deklaration

```
TBitmapCoreInfo = record
   bmciHeader: TBitmapCoreHeader;
   bmciColors: array[0..0] of TRGBTriple;
end;
```

Hinweise

TBitmapCoreInfo-Records kombinieren Größe- und Farbinformationen eines **TBitmapCoreHeader**-Record mit einer Farbtabelle, um ein geräteunabhängiges Bitmap vollständig zu definieren.

Felder

bmciHeader

Das Feld bmciHeader ist ein TBitmapCoreHeader Record, der Größe- und Farbinformationen des Bitmaps enthält.

Das Feld bcSize von bmciHeader kann einen Offset zum Feld bmciColors enthalten.

bmciColors

bmciColors ist ein Array von **TRGBTriple** Records. Die Zahl der Elemente in diesem Array wird durch das Feld bcBitCount von bmciHeader bestimmt.

bmciColors kann entweder ein Array von RGB-Farb-Records oder ein Array von Indices in die aktuelle logische Palette sein.

Die Interpretation des Feldes hängt vom Parameter Usage ab, der an die Funktion übergeben wird, die auf die geräteunabhängige Bitmap zugreift.

Die Angabe von RGB-Werten für das Feld bmciColors ist sehr ratsam, außer das Bitmap wird nur von einem einzelnen Programm verwendet. Wird das Bitmap in einer Datei gespeichert oder in ein anderes Programm kopiert, sollte kein Palettenindex benutzt werden.

Siehe auch

DIB_ Farbtabellenbezeichner.

TBitmapFileHeader (Record) (Unit WinTypes)

Deklaration

```
TBitmapFileHeader = record
   bfType: Word;
   bfSize: Longint;
   bfReserved1: Word;
   bfReserved2: Word;
   bfOffBits: Longint;
end;
```

Hinweise

Der Record TBitmapFileHeader definiert den Kopf einer geräteunabhängigen Bitmapdatei, die Daten zur Definition der Typgröße und des Layouts der Bitmap-Datei enthält.

Eine geräteunabhängige Bitmapdatei besteht aus TBitmapFileHeader, gefolgt von **TBitmapInfo** Record oder **TBitmapCoreInfo** Record, darauf die wirklichen Bitmap-Daten.

Felder

bfType

bfType gibt den Dateityp an, dieser muß BM sein.

bfSize

bfSize gibit die Größe der Datei in 4 Byte großen Blöcken an.

bfReserved1 und bfReserved2

bfReserved1 und bfReserved2 sind von Windows reserviert.

bfOffBits

Das Feld bfOffBits gibt an, wieviele Bytes vom Dateianfang entfernt die Bitmap-Information beginnt.

TBitmapInfo (Record) (Unit WinTypes)

Deklaration

```
TBitmapInfo = record
   bmiHeader: TBitmapInfoHeader;
   bmiColors: array[0..0] of TRGBQuad;
end;
```

Hinweise

TBitmapInfo-Records enthalten Größe- und Farbinformationen für geräteunabhängige Bitmaps in Windows 3.0.

Das eigentliche Bitmap ist als Array von Bytes definiert, welches die Pixel des Bitmaps darstellen.

Felder

bmiHeader

Das Feld bmiHeader enthält einen **TBitmapInfoHeader**-Record, der Größe- und Farbformatierung des Bitmaps definiert.

bmiColors

Das Feld bmiColors ist ein Array von **TRGBQuad**-Records mit den Farben des Bitmaps.

Die Zahl der Einträge im Array wird durch das Feld biBitCount im Record bmiHeader bestimmt.

TBitmapInfoHeader (Record) (Unit WinTypes)

Deklaration

```
TBitmapInfoHeader = record
   biSize: Longint;
   biWidth: Longint;
   biHeight: Longint;
   biPlanes: Word;
   biBitCount: Word;
   biCompression: Longint;
   biSizeImage: Longint;
   biXPelsPerMeter: Longint;
   biYPelsPerMeter: Longint;
   biClrUsed: Longint;
   biClrImportant: Longint;
end;
```

Hinweise

TBitmapInforHeader-Records werden von **TBitmapInfo**-Records verwendet, um Größe- und Farbformatierung eines geräteunabhängigen Bitmaps für Windows 3.0 festzulegen.

Felder

biSize

Das Feld biSize enthält die Größe des Records in Bytes.

biWidth und biHeight

biWidth und biHeight geben Breite und Höhe des Bitmaps in Pixel.

biPlanes

biPlanes nennt die Anzahl von Ebenen für das Zielgerät und muß auf 1 gesetzt werden.

biBitCount

Das Feld biBitCount enthält die Anzahl von Bits, die zur Beschreibung jedes Pixels des Bitmaps benötigt werden.

Die Bedeutung der Werte von biBitCount entspricht genau dem Feld bcBitCount eines **TBitmapCoreHeader**-Records.

biCompression

biCompression enthält den Kompressionstyp des Bitmaps, jede der **bi_xxx Konstanten** ist als Wert zulässig.

biSizeImage

biSizeImage gibt die Größe des Bitmap-Image in Bytes an.

biXPelsPerMeter und biYPelsPerMeter

biXPelsPerMeter und biYPelsPerMeter enthalten horizontale und vertikale Auflösung des Zielgeräts für das Bitmap.

biClrUsed

Das Feld biClrUsed enthält die Zahl der Farbpaletteneinträge, welche das Bitmap tatsächlich verwendet.

Der Wert biBitCount legt die Maximalzahl an Einträgen fest. Der Wert 0 in biClrUsed

bedeutet, daß die Maximalzahl verwendet wird.

Hat biClrUsed einen Wert zwischen 1 und 23, gibt die Zahl die verwendeten Farben an.

Hat biBitCount den Wert 24, gibt biClrUsed die Größe der Referenzfarbtabelle an, die Windows verwendet, um den Palettenzugriff zu beschleunigen.

biClrImportant

Das Feld biClrImportant enthält die Anzahl der zur Darstellung des Bitmaps wichtigen Farben. Der Wert 0 gibt an, daß alle Farben wichtig sind.

TCatchBuf (Record) (Unit WinTypes)

Deklaration

```
TCatchBuf = array[0..8] of Integer;
```

Hinweise

TCatchBuf ist ein Record, der den Status aller Systemregister enthält. Er wird von den Funktionen **Catch** und **Throw** verwendet.

TClientCreateStruct (Record) (Unit WinTypes)

Deklaration

```
TClientCreateStruct = record
   hWindowMenu: THandle;
   idFirstChild: Word;
end;
```

Hinweise

TClientCreateStruct enthält beim Erstellen von MDI-Client-Fenstern die Fenster-ID und Menü-Information.

Felder

hWindowMenu

hWindowMenu ist ein Handle zum Programm-Menü.

idFirstChild

idFirstChild enthält die ID des ersten untergeordneten Fensters des MDI-Programms.

IDs von untergeordneten Fenstern werden von Windows zugewiesen und verwaltet.

TColorRef (Typ) (Unit WinTypes)WinTypesUnit

Deklaration

```
TColorRef = Longint;
```

Hinweise

TColorRef ist ein 32-Bit-Wert, der einer Farbe zugeordnet ist und von GDI-Funktionen verwendet wird. Er kann, abhängig vom höherwertigen Byte des höherwertigen Word im Long-Integerwert, auf drei Weisen interpretiert werden.

Ist das höchste Byte gleich 0, stellen die nächsten drei Bytes RGB-Farbintensitäten für Blau, Grün und Rot dar. Der Wert $00FF0000 steht für intensives reines Blau, $0000FF00 für reines Grün und $000000FF für reines Rot.

$00000000 repräsentiert schwarz und $00FFFFFF weiß.

RGB-Werte können mit der Funktion **RGB** in TColorRef-Werte verwandelt werden.

Wenn das höchste Byte gleich 1 ist, muß das nächste Byte 0 sein. Die nächsten beiden Bytes bilden einen Index in die logische Palette. $01000000 ist der Index Null (der erste Eintrag) der Palette. Integer-Indices der Palette können mit der Funktion **GetNearestPaletteIndex** in TColorRef-Werte verwandelt werden.

Wenn das höchste Byte gleich 2 ist, stellen die nächsten drei Bytes RGB-Farbintensitäten dar, deren Wert aber der ähnlichsten Farbe der logischen Palette des aktuellen Geräts angepaßt wird. Paletten-relative RGB-TColorRef-Werte können durch die Funktion **PaletteRGB** aus RGB-Werten erzeugt werten.

Damit ein Palettenindex oder paletten-relative RGB-TColorRef-Werte zu einem Gerätkontext passen, muß ein Programm mit eigener Palette (durch Verwendung von **SelectPalette** und **RealizePalette**) die Palette wählen und realisieren, damit mit korrekten Farben gearbeitet wird. Ebenso muß vor dem Erstellen eines logischen Zeichenwerkzeugs die Palette gewählt und realisiert werden.

TCompareItemStruct (Record) (Unit WinTypes)

Deklaration

```
TCompareItemStruct = record
   CtlType: Word;
   CtlID: Word;
   hwndItem: HWnd;
   itemID1: Word;
   itemData1: Longint;
   itemID2: Word;
   itemData2: Longint;
end;
```

Hinweise

Der TCompareItemStruct Record wird zum Vergleich von Elementen in selbstdefinierten Kombinationsfenstern oder Listen benützt.

Beim Zufügen von Elementen wird eine **wm_CompareItem**-Botschaft erzeugt, einer der Parameter davon ist ein Zeiger auf TCompareItemStruct. Nach Empfang der Meldung vergleicht der Eigner die Elemente im Record und gibt je nach Ergebnis einen Wert zurück.

Felder

CtlType

Das Feld CtlType enthält eine **odt_xxx Konstante**, die angibt, daß es sich um ein selbstdefiniertes Kombinationsfenster (odt_ComboBox) oder Listenfenster (odt_ListBox) handelt.

CtlID und hwndItem

CtlID und hwndItem geben die Element-ID und das Fenster-Handle des Elements an.

itemID1 und itemData1

itemID1 und itemData1 sind der Index auf das erste Element in Kombinationsfenster oder Liste und die Daten desselben.

itemID2 und itemData2

itemData1 sind Daten des Parameters lParam der Botschaft, die das Element in die Liste eingefügt hat.

itemID2 und itemData2 enthalten disselben Daten für des zweite Element.

TComStat (Record) (Unit WinTypes)

Deklaration

```
TComStat = record
   Flags: Byte;
   cbInQue: Word;
   cbOutQue: Word;
end;
```

Hinweise

TComStat Record enthält den Gerätestatus.

Dieser wird von der Funktion **GetCommError** verwendet.

Felder

Flags

Das Feld Flags ist ein Bitmap-Feld, dessen Bits durch **com_xxx Konstanten** definiert werden.

cbInQue und cbOutQue

Die Felder cbInQue und cbOutQue enthalten die Anzahl der Zeichen der Sende- und Empfangswarteschlangen.

TCreateStruct (Record) (Unit WinTypes)

Deklaration

```
TCreateStruct = record
   lpCreateParams: PChar;
   hInstance: THandle;
   hMenu: THandle;
   hwndParent: HWnd;
   cy: Integer;
   cx: Integer;
   y: Integer;
   x: Integer;
   style: LongInt;
   lpszName: PChar;
   lpszClass: PChar;
   dwExStyle: Longint;
end;
```

Hinweise

Im TCreateStruct-Record werden Initialisierungsparameter an die Fensterfunktion eines Programms weitergegeben.

Felder

lpCreateParams

Das Feld lpCreateParams zeigt auf Fenstererstellungsdaten.

hInstance, hMenu, und hwndParent

hInstance, hMenu und hwndParent sind Handles zur Modulinstanz des Moduls, das Besitzer des Fensters ist, zum Fenstermenü und zum übergeordneten Fenster des neuen Fensters.

hwndParent

Das Feld hwndParent ist 0, wenn das erstellte Fenster das Hauptfenster des Programms ist.

cy, cx, y, und x

cy und cx sind Höhe und Breite des Fensters. y und x sind vertikale und horizontale Koordinaten der oberen linken Fensterecke, relativ zum übergeordneten Fenster, falls vorhanden.

style

style enthält die Stil-flags des Fensters.

lpszName und lpszClass

lpszName und lpszClass zeigen auf den null-terminierten String mit Namen und Klassennamen des Fensters.

dwExStyle

dwExStyle enthält weitere Stilinformation für das Fenster.

TDCB (Record) (Unit WinTypes)

Deklaration

```
TDCB = record
   Id: Byte;
   BaudRate: Word;
   ByteSize: Byte;
   Parity: Byte;
   StopBits: Byte;
   RlsTimeout: Word;
   CtsTimeout: Word;
   DsrTimeout: Word;
   Flags: Word;
   XonChar: Char;
   XoffChar: Char;
   XonLim: Word;
   XoffLim: Word;
   PeChar: Char;
   EofChar: Char;
   EvtChar: Char;
   TxDelay: Word;
end;
```

Hinweise

TDCB Records enthalten Informationen für serielle Schnittstellen, die von den Funktionen **BuildCommDCB**, **GetCommState** und **SetCommState** verwendet werden.

Felder

Id

Das Feld Id ist die ID der Schnittstelle.

Ist das hochwertige Bit gesetzt (vgl. Maske **LPTx**), ist die Schnittstelle parallel, ansonsten seriell.

BaudRate, ByteSize, Parity, und StopBits

BaudRate, ByteSize, Parity und StopBits definieren Kommunikationsparameter der Schnittstelle.

Das Feld ByteSize gibt die Anzahl von Bits jedes Zeichens an und liegt im Bereich zwischen 4 und 8.

Parity ist eine der **Kommunikationskonstanten** EvenParity, MarkParity, NoParity, OddParity oder SpaceParity.

StopBits ist eine der Konstanten OneStopBit, One5StopBits oder TwoStopBits.

RlsTimeout, CtsTimeout, und DsrTimeout

RlsTimeout, CtsTimeout und DsrTimeout geben die Wartezeit in Millisekunden an, die ein Gerät auf die Signale RLSD, CTS und DSR warten sollte.

Flags

Im Feld Flags ist jedes Bit ein Schalter für eine Fehlerüberprüfung.

Die Bits werden in der folgenden Tabelle definiert, Zugriff auf einzelne Bits ist über die **dcb_xxx Konstanten** möglich.

Bit	Bedeutung, wenn gesetzt
fbinary	Binärmodus verwendet
fRtsDisable	RTS ausgeschaltet
fParity	Paritätsprüfung eingeschaltet
fOutxCtsFlow	CTS bei Übertragung überprüft
fOutxDsrFlow	DSR bei Übertragung überprüft
fDummy	Reserviert
fDtrDisable	DTR ausgeschaltet
fOutX	Xon/Xoff bei Übertragung
fInX	Xon/Xoff bei Empfang
fPeChar	Paritätsfehler ersetzt
fNull	Nullzeichen entfernt
fChEvt	EvtChar-Zeichen als Ereignis
fDtrFlow	DTR für Empfangskontrolle
fRtsFlow	RTS für Empfangskontrolle
fdummy2	Reserviert

XonChar und XoffChar

XonChar und XoffChar geben die Werte der Xon- und Xoff-Zeichen für Senden und Empfang an.

XoffLim

Das Feld XoffLim gibt die Anzahl von Zeichen in der Empfangswarteschlange an, bei der ein Xoff ausgelöst wird.

PeChar, EofChar, und EvtChar

PeChar, EofChar und EvtChar definieren Zeichen, die zum Ersetzen von Paritätsfehlern, zur Anzeige des Datenendes und zur Anzeige eines Ereignisses verwendet werden.

TxDelay

Das Feld TxDelay wird nicht verwendet.

TDDEAck (Record) (Unit WinTypes)

Deklaration

```
TDDEAck = record
   Flags: Word;
end;
```

Hinweise

Der TDDEAck-Record enthält Ack-Informationen (Empfangsbestätigung) in einem Parameter der Meldung **wm_dde_Ack** in Reaktion auf alle DDE-Meldungen außer **wm_dde_Initiate**.

Felder

Flags

Das Feld Flags ist ein Bitmap-Word, von dem zur Zeit nur zwei Bits für die Verwendung in einem Programm definiert sind.

Das Setzen des Bits fAck zeigt an, daß die Anfrage akzeptiert wurde.

Das Setzen des Bits fBusy zeigt an, daß das Programm auf die Anfrage nicht reagieren kann. fBusy ist nur sinnvoll, wenn fAck 0 ist.

Das niederwertige Byte von Flags ist das "Feld" bAppReturnCode, auf welches mit der Maske **dde_AppReturnCode** zugegriffen werden kann. Es enthält anwendungsspezifische Rückgabecodes.

Die übrigen Bits sind für Windows reserviert.

TDDEAdvise (Record) (Unit WinTypes)

Deklaration

```
TDDEAdvise = record
   Flags: Word;
   cfFormat: Integer;
end;
```

Hinweise

Der TDDEAdvise-Record enthält eine Anfrage an einen DDE-Server, die in einem Parameter der Botschaft **wm_dde_Advise** weitergegeben wird.

Felder

Flags

Das Feld Flags ist ein Bitmap-Feld, in dem nur zwei Bits definiert sind: fAckReq und fDeferUpd. Auf sie kann mit **dde_xxx Konstanten** zugegriffen werden.

Die 14 niedrigwertigen Bits von Flags sind nicht definiert, aber reserviert.

Das Setzen des Bits fAckReq zeigt an, daß der Server aufgefordert wird, seine wm_dde_Data-Botschaften mit gesetztem fAckReq bit zu senden, um auf diese Weise eine Kontrolle des Botschaftsflusses zu implementieren.

Das Setzen des Bits fDeferUpd zeigt an, daß der Server aufgefordert wird, seine wm_dde_Data-Botschaften mit hData-Handles auf 0 zu senden, um das Client-Programm auf geänderte Daten hinzuweisen.

Auf diese Weise alarmiert, kann das Client-Programm mit der Botschaft wm_dde_Request antworten und die geänderten Daten abfragen.

cfFormat

cfFormat gibt über eine der **cf_xxx Konstanten** das bevorzugte Datenformat des Client an.

TDDEData (Record) (Unit WinTypes)

Deklaration

```
TDDEData = record
   Flags: Word;
   cfFormat: Integer;
   Value: array[0..1] of Char;
end;
```

Hinweise

Der TDDEData Record enthält Daten, die von einem Programm zu einem anderen übertragen werden. Er wird als Parameter in **wm_dde_Data**-Botschaften übergeben.

Felder

Flags

Das Feld Flags ist ein Bitmap-Feld, in dem nur drei Bits definiert sind: fAckReq, fRelease, und fRequested, auf die mit **dde_xxx Konstanten** zugegriffen wird.

DasSetzen des Bits fAckReq zeigt an, daß das Client-Programm mit einer wm_dde_Ack-Botschaft den Empfang der Daten bestätigen soll.

Das Setzen des Bits fRelease zeigt an, daß das Client-Programm die Daten der Meldung wm_dde_Data nach der Bearbeitung freigeben soll.

Das Setzen des Bits fRequested zeigt an, daß die Daten als Antwort auf eine Anfrage kommen.

Alle anderen Bits in Flags sind reserviert.

cfFormat

cfFormat enthält eine **cf_xxx Konstante**, die das Format der zum Client-Programm zu sendenden Daten angibt.

Value

Das Feld Value enthält die übertragenen Daten im Format cfFormat.

TDDEPoke (Record) (Unit WinTypes)

Deklaration

```
TDDEPoke = record
   Flags: Word;
   cfFormat: Byte;
   Value: array[0..1] of Byte;
end;
```

Hinweise

Der TDDEPoke-Record enthält nicht abgefragte Daten zusammen mit einer **wm_dde_Poke** Botschaft.

Felder

Flags

Das Feld Flags ist ein Bitmap-Feld, in dem nur ein Bit definiert ist: fRelease. Das Setzen des Bits fRelease zeigt an, daß der Empfänger die Daten nach Verarbeitung freigeben sollte.

Alle anderen Bits von Flags sind reserviert.

Auf Release kann mit der **dde_Release Konstanten** zugegriffen werden.

cfFormat

Das Feld cfFormat gibt das vom Client-Programm bevorzugte Datenformat als **cf_xxx Konstanten** an.

Value

Das Feld Value enthält die übertragenen Daten im Format cfFormat.

TDeleteItemStruct (Record) (Unit WinTypes)

Deklaration

```
TDeleteItemStruct = record
   CtlType: Word;
   CtlID: Word;
   itemID: Word;
   hwndItem: HWnd;
   itemData: Longint;
end;
```

Hinweise

Der TDeleteItemStruct-Record beschreibt ein Element, das aus einem Kombinationsfenster oder einer Liste gelöscht wurde. Die Botschaft **wm_DeleteItem** wird an den Eigner des Elements gesendet, wobei der Parameter lParam auf den Record TDeleteItemStruct zeigt.

Felder

CtlType

Das Feld CtlType gibt den Elementtyp an, dieser ist entweder Listenfenster (**odt_ListBox**) oder Kombinationsfenster (odt_ComboBox).

CtlId

Das Feld CtlId ist die Element-ID des Fensters.

itemID

Das Feld itemID ist der Index des gelöschten Elements.

hwndItem

Das Feld hwndItem ist das Fenster-Handle des Elements.

itemData

Das Feld itemData ist der 32-bit-Wert des Indexelements.

TDevMode (Record) (Unit WinTypes)

Deklaration

```
TDevMode = record
   dmDeviceName: array[0..cchDeviceName-1] of Char;
   dmSpecVersion: Word;
   dmDriverVersion: Word;
   dmSize: Word;
   dmDriverExtra: Word;
   dmFields: LongInt;
   dmOrientation: Integer;
   dmPaperSize: Integer;
   dmPaperLength: Integer;
   dmPaperWidth: Integer;
   dmScale: Integer;
   dmCopies: Integer;
   dmDefaultSource: Integer;
   dmPrintQuality: Integer;
   dmColor: Integer;
   dmDuplex: Integer;
end;
```

Hinweise

TDevMode Records werden von **DeviceCapabilities** und ExtDeviceMode für Informationen über Druckertreiber verwendet.

Felder

dmDeviceName

Das Feld dmDeviceName enthält einen null-terminierten String mit dem Namen des Geräts.

dmSpecVersion

dmSpecVersion ist die Versionsnummer der Datenspezifikation, aktuell $0300.

dmDriverVersion

dmDriverVersion ist die Versionsnummer des Treibers vom Hersteller.

dmSize

dmSize gibt die Größe des Records an, wobei das Feld dmDriverData am Ende des REcords nicht berücksichtigt wird.

dmDriverExtra

dmDriverExtra gibt die Größe des Feldes dmDriverData an.

dmFields

dmFields ist ein 32 Bit großes Bitmap-Feld, das anzeigt, welche der übrigen Felder initialisiert wurden.

Jedes Bit ist mit den Konstanten der folgenden Tabelle einem Feld zugordnet:

Flag	Feld
dm_Color	dmColor
dm_Copies	dmCopies
dm_DefaultSource	dmDefaultSource

dm_Duplex	dmDuplex
dm_Orientation	dmOrientation
dm_PaperLength	dmPaperLength
dm_PaperSize	dmPaperSize
dm_PaperWidth	dmPaperWidth
dm_PrintQuality	dmPrintQuality
dm_Scale	dmScale
dm_SpecVersion	dmSpecVersion

dmOrientation
dmOrientation gibt die Papierausrichtung (Hoch- bzw. Querformat) über eine der Konstanten **dmorient_xxx Konstanten**an.

dmPaperSize
dmPaperSize selektiert die Papiergröße mit einer der Konstanten **dmpaper_xxx Konstanten**.

dmPaperLength und dmPaperWidth
dmPaperLength und dmPaperWidth überschreiben die Papierlänge und -breite im Feld dmPaper.

dmScale
dmScale skaliert die Druckausgabe mit dem Faktor dmScale/100.

Beispielsweise werden bei einem Wert von 75 beim Ausdruck Bilder auf 75% der Originalgröße verkleinert

dm_Copies
dm_Copies gibt die Zahl der auszudruckenden Kopien an.

dmDefaultSource
dmDefaultSource gibt den Vorgabe-Papiereinzug an, in **dmbin_xxx Konstanten**.

dmDefaultSource
dmDefaultSource spezifiziert die zu verwendende Druckauflösung mit geräteunabhängigen **dmres_xxx Konstanten** (negative Werte) oder mit einer positiven Zahl, die einen geräteabhängigen Wert für Punkte pro Zoll darstellt.

dmColor
Das Feld dmColor spezifiert Farb- oder Schwarzweiß-Druck über **dmcolor_xxx Konstanten**.

dmDuplex
dmDuplex spezifiziert einseitigen oder zweiseitigen Druck mit **dmdup_xxx Konstanten**.

dmDriverData
dmDriverData enthält Daten für und vom Treiber.

TDrawItemStruct (Record) (Unit WinTypes)

Delcaration

```
TDrawItemStruct = record
  CtlType: Word;
  CtlID: Word;
  itemID: Word;
  itemAction: Word;
  itemState: Word;
  hwndItem: HWnd;
  hDC: HDC;
  rcItem: TRect;
  itemData: Longint;
end;
```

Hinweise

Der TDrawItemStruct Record enthält Daten zum Zeichnen selbstdefinierter Elemente.

Der Eigner des Elements erhält einen Zeiger auf TDrawItemStruct im Parameter lParam der Botschaft **wm_DrawItem**.

Felder

CtlType

Das Feld CtlType ist der Elementtyp, der durch eine **odt_xxx Konstante** zugeordnet wurde.

CtlID

Element-ID Nummer (wird nicht für Menüs verwendet).

itemID

Menü-ID des Elementindex. Für leere Listen oder Kombinationsfenster st eht der Wert -1.

itemAction

Das Feld itemAction definiert mit Hilfe von **oda_xxx Konstanten**, wann und wie das Element gezeichnet wird.

itemState

Das Feld itemState beschreibt den Zustand des Elements nach dem Zeichnen und verwendet dazu oda_ constants.

hwndItem

Das Feld hwndItem ist das Fenster-Handle des Elements. Bei einem Menü ist dies das Handle des Menüs, welches das Element enthält.

hDC

hDC ist das Handle des Gerätekontextes, das beim Zeichnen dieses Elements verwendet werden muß.

rcItem

Das Feld rcItem spezifiziert das umgrenzende Rrechteck des Elements für das Gerät. Windows schneidet selbstdefinierte Elemente an dieser Grenzlinie ab.

itemData

Das Feld itemData enthält:

- einen Wert eines selbstdefinierten Kombinationsfensters oder einer Liste, die mit einer der Botschaften **cb_AddString**, **cb_InsertString**, **lb_AddString** oder **lb_InsertString** erzeugt wurden
- einen Long-Integerwert für das Menüelement im Parameter NewItem des Aufrufs von InsertMenu, mit es einfügt wurde.

itemData ist für selbstdefinierte Aktionsschalter undefiniert.

TFarProc (Typ) (Unit WinTypes)

Deklaration

```
TFarProc = Pointer;
```

Hinweise

TFarProc ist ein Zeiger, üblicherweise auf eine Prozedur.

TGlobalHandle (Typ) (Unit WinTypes)

Deklaration

```
TGlobalHandle = THandle;
```

Hinweise

TGlobalHandle entspricht exakt THandle, aber verdeutlicht einem Leser des Programms, daß es sich um ein Handle auf ein globales Element, beispielsweise einen globalen Speicherblock, handelt.

THandle (Typ) (Unit WinTypes)

Deklaration

```
THandle = Word;
```

Hinweise

THandle definiert einen generischen Handle-Typ.

THandleTable (Record) (Unit WinTypes)

Deklaration

```
THandleTable = record
   objectHandle: array[0..0] of THandle;
end;
```

Hinweise

THandleTable ist ein Array von Handles, üblicherweise zum Speichern von Zeichenwerkzeugen.

TLocalHandle (Typ) (Unit WinTypes)

Deklaration

```
TLocalHandle = THandle;
```

Hinweise

TLocalHandle entspricht THandle, aber macht für einen Leser des Programms deutlich, daß es sich um ein Handle auf ein lokales Element handelt.

TLogBrush (Record) (Unit WinTypes)

Deklaration

```
TLogBrush = record
   lbStyle: Word;
   lbColor: Longint;
   lbHatch: Integer;
end;
```

Hinweise

Der TLogBrush Record enthält Informationen zum Erzeugen eines logischen Pinsels mit der Funktion **CreateBrushIndirect**.

Felder

lbStyle

Das Feld lbStyle enthält eine der **bs_xxx Konstanten**, die den Stil des Pinsels festlegen, d.h. in welcher Form der Pinsel zeichnet (gefüllte Flächen, Umrisse, schraffierte oder gemusterte Flächen).

lbColor

lbColor ist ein **TColorRef**-Record. Die Farbe wird ignoriert, wenn der Stil Umrißzeichnung oder Muster spezifiziert.

Ist der Stil **bs_DIBPattern** muß das niederwertige Word eine der **DIB_xxx Konstanten** enthalten, die die Farben explizit oder als Index in eine Palette enthalten.

lbHatch

Das Feld lbHatch enthält den Schraffurstil. Je nach Stil des Zeicheninstruments kann lbHatch folgende Werte enthalten:

Stil	lbHatch enthält
bs_DIBPattern	Handle eines geräteunabhängigen Bitmaps
bs_Hatched	Eine der **hs_xxx Konstanten** mit Ausrichtung der Schraffur
bs_Hollow	Wird Ignoriert
bs_Pattern	Handle des Bitmaps mit dem Muster
bs_Solid	Wird ignoriert

TLogFont (Record) (Unit WinTypes)

Deklaration

```
TLogFont = record
   lfHeight: Integer;
   lfWidth: Integer;
   lfEscapement: Integer;
   lfOrientation: Integer;
   lfWeight: Integer;
   lfItalic: Byte;
   lfUnderline: Byte;
   lfStrikeOut: Byte;
   lfCharSet: Byte;
   lfOutPrecision: Byte;
   lfClipPrecision: Byte;
   lfQuality: Byte;
   lfPitchAndFamily: Byte;
   lfFaceName: array[0..lf_FaceSize - 1] of Byte;
end;
```

Hinweise

Der TLogFont Record enthält Attribute einer logischen Schrift für die Funktion **CreateFontIndirect**.

Felder

lfHeight und lfWidth

lfHeight und lfWidth enthalten Durchschnittshöhe und -breite der Schrift.

lfEscapement und lfOrientation

lfEscapement und lfOrientation geben Neigungswinkel und Ausrichtung des Textes in Zehntelgrad gegen den Uhrzeigersinn von der X-Achse aus an.

lfWeight

Das Feld lfWeight gibt die Dichte der Schrift an, in gefüllten Pixel pro 1000. Der Wert kann im Bereich 0 bis 1000 liegen.

400 ist normal, 700 fett. Die Werte variieren nach Schriftart.

Der Wert 0 bedeutet, daß eine Vorgabedichte verwendet wird.

lfItalic, lfUnderline und lfStrikeOut

Die Felder lfItalic, lfUnderline und lfStrikeOut sind üblicherweise auf 0 gesetzt. Sind sie ungleich 0, stehen sie für kursiv, unterstrichen oder durchstrichen.

lfCharSet

lfCharSet ist einer der drei vordefinierten **Zeichensätze**: ANSI_CharSet, OEM_CharSet oder Symbol_CharSet. Andere Zeichensätze sind definierbar.

lfOutPrecision

Das Feld lfOutPrecision enthält eine der **out_xxx Flags**; Vorgabe ist out_Default_Precis.

lfClipPrecision

lfClipPrecision enthält die Clipping precision der Schrift in **clip_xxx Flags**. Vorgabe ist clip_Default_Precis.

lfQuality

Das Feld lfQuality enthält **Schriftqualitäts-flags**: Default_Quality, Draft_Quality, oder Proof_Quality.

lfPitchAndFamily

lfPitchAndFamily ist eine Kombination von **font pitch flag** (Default_Pitch, Fixed_Pitch, oder Variable_Pitch) und **font family flag** (wie ff_Roman oder ff_Script).

lfFaceName

lfFaceName enthält den Namen der Schrift in einem null-terminierten String. **nil** in lfFaceName läßt GDI eine Vorgabeschrift verwenden.

TLogPalette (Record) (Unit WinTypes)

Deklaration

```
TLogPalette = record
   palVersion: Word;
   palNumEntries: Word;
   palPalEntry: array[0..0] of TPaletteEntry;
end;
```

Hinweise

Der TLogPalette Record enthält Daten für eine logische Palette, wie für die Funktion **CreatePalette**.

Felder

palVersion

Das Feld palVersion enthält die Windowsversion der Struktur, aktuell $0300.

palNumEntries

palNumEntries ist die Anzahl Einträge der Palette.

palPalEntry

palPalEntry ist ein Array von **TPaletteEntry**-Records, ein Element füür jeden der Einträge der Palette.

TLogPen (Record) (Unit WinTypes)

Deklaration

```
TLogPen = record
   lopnStyle: Word;
   lopnWidth: TPoint;
   lopnColor: Longint;
end;
```

Hinweise

Der TLogPen Record enthält Attribute eines logischen Stifts, wie für die Funktion **CreatePenIndirect**.

Felder

lopnStyle

lopnStyle ist der Stil des Stifts, eine der **ps_ constants**.

lopnWidth

lopnWidth ist die Breite des Stifts in logischen Einheiten.

lopnColor

lopnColor ist die Stiftfarbe.

TMDICreateStruct (Record) (Unit WinTypes)

Deklaration

```
TMDICreateStruct = record
   szClass: PChar;
   szTitle: PChar;
   hOwner: THandle;
   x, y: Integer;
   cx, cy: Integer;
   style: LongInt;
   lParam: LongInt;
end;
```

Hinweise

Der TMDICreateStruct Record enthält Daten zur Erstellung eines MDI des untergeordneten Fensters.

Der Parameter lParam einer Meldung **wm_Create** enthält einen **TCreateStruct**-Record, dessen lpCreateParams Felder auf einen TMDICreateStruct Record zeigen, welchen die Meldung **wm_MDICreate** liefert.

Felder

szClass

szClass zeigt auf die Klasse des untergeordneten Fensters.

szTitle

szTitle zeigt auf den Fenstertitel.

hOwner

hOwner ist das Instanz-Handle des Programms, welches das Fenster erzeugt.

cx, cy, x, und y

cx und cy sind ursprüngliche Höhe und Breite; x und y X- und Y-Koordinaten des untergeordneten Fensters.

Ein Wert von **cw_UseDefault** für einen dieser Koordinaten führt zu entsprechenden Vorgabewerten.

style

style enthält zusätzliche Stilelemente für das Fenster, wie **ws_ window styles**: ws_Minimize, ws_Maximize, ws_HScroll, oder ws_VScroll.

lParam

lParam wird vom Programm definiert.

TMeasureItemStruct (Record) (Unit WinTypes)

Deklaration

```
TMeasureItemStruct = record
   CtlType: Word;
   CtlID: Word;
   itemID: Word;
   itemWidth: Word;
   itemHeight: Word;
   itemData: Longint;
end;
```

Hinweise

Der TMeasureItemStruct Record enthält die Dimensionen eines selbst definierten Elements.

Eine **wm_MeasureItem** Meldung enthält einen Zeiger auf einen TMeasureItemStruct Record in seinem Parameter lParam.

Felder

CtlType

CtlType ist der Elementtyp, der von einer der **odt_constants** bezeichnet wird.

CtlID

CtlID ist die ID des Elements (nicht für Menüs).

itemID

itemID ist die Menü-ID des Elementindexes.

itemWidth and itemHeight

itemWidth und itemHeight speichern die Höhe und Breite des Elements.

itemData

Das Feld itemData enthält entweder einen selbstdefinierte Liste oder ein Kombinationsfenster, das von den Meldungen **cb_AddString**, **cb_InsertString**, **lb_AddString**, oder **lb_InsertString** zurückgegeben wird, die das Element erstellten, oder ein Long-Integer für das Menüelement aus dem NewItem Parameter des Aufrufes von InsertMenu, der es einfügte.

itemData

itemData ist für selbstdefinierte Aktionsschalter undefiniert.

TMenuItemTemplate (Record) (Unit WinTypes)

Deklaration

```
TMenuItemTemplate = record
  mtOption: Word;
  mtID: Word;
  mtString: PChar;
end;
```

Hinweise

Ein TMenuItemTemplate-Record speichert die Informationen über ein Menüelement. TMenuItemTemplate-Records können aneinandergereiht werden und eine Liste von Menüelementen bilden.

Zusammen mit einem Record vom Typ **TMenuItemTemplateHeader** bildet solch eine Liste eine vollständige Menüvorlage.

Felder

mtOption

mtOption enthält eine (oder eine Kombination von) **mf_xxx Konstanten**. These define the type and state of the menu item.

mtID

mtID speichert einen ID-Code für ein Menüelement. Pop-up-Menüelemente (diejenigen, die ein anderes Menü aufrufen) haben kein mtID-Feld.

mtString

mtString enthält einen null-terminierten String, der den Namen des Menüelements angibt.

TMenuItemTemplateHeader (Record) (Unit WinTypes)

Deklaration

```
TMenuItemTemplateHeader = record
   versionNumber: Word;
   offset: Word;
end;
```

Hinweise

Der Record TMenuItemTemplateHeader enthält den Kopf einer Menüvorlage.Eine vollständige Menüvorlage setzt sich aus dem Kopf und einer Elementliste zusammen.

Die Elementliste besteht normalerweise aus einer Liste von **TMenuItemTemplate**-Records.

Felder

versionNumber

versionNumber enthält die Versionsnummer der Menüvorlage, diese Nummer sollte 0 sein.

offset

offset gibt den Offset (in Bytes) des Anfangs der Elementeliste an (relativ zum Dateiheader).

TMetaFilePict (Record) (Unit WinTypes)

Deklaration

```
TMetaFilePict = record
   mm: Integer;
   xExt: Integer;
   yExt: Integer;
   hMF: THandle;
end;
```

Hinweise

Der TMetaFilePict Record definiert das Format des Metadatei-Bilds, das zum Austausch von Metadatei-Daten über die Zwischenablage verwendet wird.

Felder

mm

mm enthält den Abbildungsmodus des Bildes.

xExt und yExt

xExt und yExt sind Höhe und Breite des Rechtecks für das Bild, außer **mm_mapping mode** ist mm_Isotropic oder mm_Anisotropic, wobei die Felder empfohlene (mm_Anitostropic) oder relative Größen (mm_Isotropic) bekommen.

hMF

hMF ist ein Handle einer Speicher-Metadatei.

TMetaHeader (Record) (Unit WinTypes)

Deklaration

```
TMetaHeader = record
   mtType: Word;
   mtHeaderSize: Word;
   mtVersion: Word;
   mtSize: Longint;
   mtNoObjects: Word;
   mtMaxRecord: Longint;
   mtNoParameters: Word;
end;
```

Hinweise

Der TMetaHeader Record definiert das Headerformat für die Metadatei.

Eine Metadatei besteht aus einem TMetaHeader Record, gefolgt von einer Liste von Metadatei-Records, üblicherweise vom Typ **TMetaRecord**.

Felder

mtType

Das Feld mtType enthält einen von zwei Werten:
1 heißt, daß die Metadatei im Arbeitsspeicher ist, 2 auf der Platte.

mtHeaderSize

Das Feld mtHeaderSize ist die Größe des Headers in Bytes.

mtVersion

mtVersion ist die Windows-Versionsnummer, aktuell $0300 für version 3.0.

mtSize

mtSize ist die Größe der Datei in Words.

mtNoObjects und mtMaxRecord

mtNoObjects und mtMaxRecord Felder geben die Maximalzahl von Objekten an, die eine Metadatei aufnehmen kann, und die Gößе (in words) des größten Metadatei-Records.

mtNoParameters

mtNoParameters wird momentan nicht verwendet.

TMetaRecord (Record) (Unit WinTypes)

Deklaration

```
TMetaRecord = record
   rdSize: Longint;
   rdFunction: Word;
   rdParm: array[0..0] of Word;
end;
```

Hinweise

Der TMetaRecord Record definiert einen typischen Metadatei-Record. Eine Liste derartiger Records folgt dem Header der Metadatei.

Felder

rdSize

rdSize ist die Größe des Record in Words.

rdFunction

rdFunction ist die Funktionsnummer aus **meta_ Konstanten**.

rdParm

rdParm ist ein Array von Word-type Parametern der Funktion, in umgekehrter Ordnung der Übergabe an die Funktion gespeichert.

TMsg (Record) (Unit WinTypes)

Deklaration

```
TMsg = record
   hwnd: HWnd;
   message: Word;
   wParam: Word;
   lParam: LongInt;
   time: Longint;
   pt: TPoint;
end;
```

Hinweise

Der TMsg Record enthält Meldungsdaten für Programme von Windows. Die Information wird an passende Elemente (Fenster) zur Verarbeitung weitergegeben. Die Felder enthalten Informationen für das Programm.

Felder

hwnd

hwnd ist das Handle des Fensters, welches die Meldung erhält.

message

message ist die Anzahl an Meldungen.

wParam und lParam

wParam und lParam enthalten Informationen für das Fenster in der Länge Word und Long-Integer. Diese Information hängt von message ab.

time

time enthält den Zeitpunkt der Meldung.

pt

pt enthält die Cursorposition beim Senden der Meldung.

TMultiKeyHelp (Record) (Unit WinTypes)

Deklaration

```
TMultiKeyHelp = record
   mkSize: Word;
   mkKeyList: Byte;
   szKeyPhrase: array[0..0] of Byte;
end;
```

Hinweise

Der TMultiKeyHelp-Record enthält einen Index auf eine Tabelle von Schlüsselbegriffen und einen zu suchenden Texteintrag, diese werden vom Windows-Hilfesystem verwendet.

Felder

mkSize

mkSize enthält die Länge des Record.

mkKeyList

mkKeyList ist ein Zeichen, das bestimmt, welche Tabelle durchsucht wird.

szKeyPhrase

szKeyPhrase ist ein null-terminierter String, der den zu findenden Schlüsselbegriff enthält.

TOFStruct (Record) (Unit WinTypes)

Deklaration

```
TOFStruct = record
   cBytes: Byte;
   fFixedDisk: Byte;
   nErrCode: Word;
   reserved: array[0..3] of Byte;
   szPathName: array[0..127] of Char;
end;
```

Hinweise

Der TOFStruct Record enthält Informationen, die beim Öffnen einer Datei eingelesen werden.

Felder

cBytes

cBytes enthält die Länge des Records.

fFixedDisk

fFixedDisk ist ungleich 0, wenn die Datei sich auf einer Festplatte befindet, sonst 0.

nErrCode

nErrCode enthält den DOS Fehlercode, wenn die Funktion **OpenFile** scheiterte. (OpenFile gibt beim Scheitern -1 zurück.)

reserved

reserved enthält vier Bytes, die für zukünftigen Gebrauch von Windows reserviert sind.

szPathName

szPathName ist ein null-terminierter String mit dem vollständigen Pfadnamen der Datei.

TPaintStruct (Record) (Unit WinTypes)

Deklaration

```
TPaintStruct = record
   hdc: HDC;
   fErase: Bool;
   rcPaint: TRect;
   fRestore: Bool;
   fIncUpdate: Bool;
   rgbReserved: array[0..15] of Byte;
end;
```

Hinweise

Der TPaintStruct Record enthält Informationen, mit denen Programme den Client-Bereich ihrer Fenster malen.

Die meisten dieser Informationen dienen dem internen Gebrauch von Windows, aber einige Felder können vom Anwender benutzt werden.

Felder

hdc

Das Feld hdc ist das Handle des Anzeigekontextes, wo das Ausmalen passieren soll.

fErase

fErase gibt an, ob der Hintergrund neu gemalt wurde; 0 heißt, er wurde nicht neu gemalt.

rcPaint

rcPaint definiert das Rechteck, in welchem ausgemalt wird.

Alle anderen Felder sind für internen Gebrauch von Windows reserviert.

TPaletteEntry (Record) (Unit WinTypes)

Deklaration

```
TPaletteEntry = record
   peRed: Byte;
   peGreen: Byte;
   peBlue: Byte;
   peFlags: Byte;
end;
```

Hinweise

Der TPaletteEntry-Recordtyp definiert einen Eintrag in eine logische Palette, wie in **TLogPalette**.

Felder

peRed, peGreen, und peBlue

peRed, peGreen, und peBlue geben die Intensität von Rot, Grün und Blau im Paletteneintrag an.

peFlags

peFlags enthält Informationen über die Palettenverwendung.

Es kann 0 oder eines der **pc_Flags** sein: pc_Explicit, pc_NoCollapse, oder pc_Reserved.

TPattern (Typ) (Unit WinTypes)

Deklaration

```
TPattern = TLogBrush;
```

Hinweise

TPattern ist ein anderer Name für **TLogBrush**.

Wenn ein logischer Pinsel zum Ausfüllen von Mustern verwendet wird, ist der Name TPattern klarer.

TPoint (Record) (Unit WinTypes)

Deklaration

```
TPoint = record
   x: Integer;
   y: Integer;
end;
```

Hinweise

Der TPoint-Record gibt X- und Y-Koordinaten (der x und y Felder) eines Punkts auf dem Bildschirm oder in einem Fenster an.

TRect (Record) (Unit WinTypes)

Deklaration

```
TRect = record
   left: Integer;
   top: Integer;
   right: Integer;
   bottom: Integer;
end;
```

Hinweise

Der TRect Record definiert ein Rechteck durch Angabe der Koordinaten der oberen linken und unteren rechten Ecke.

Felder

left und top

left und top sind X- und Y-Koordinaten der oberen linken Ecke des Rechtecks.

right und bottom

right und bottom sind X- und Y-Koordinaten der unteren rechten Ecke des Rechtecks.

Anmerkung: Das Rechteck darf 32,768 Einheiten in keiner Richtung überschreiten.

TRGBQuad (Record) (Unit WinTypes)

Deklaration

```
TRGBQuad = record
   rgbBlue: Byte;
   rgbGreen: Byte;
   rgbRed: Byte;
   rgbReserved: Byte;
end;
```

Hinweise

Der TRGBQuad Record enthält RGB-Farbdaten für Bitmaps, wie im Feld bmiColors eines **TBitmapInfo**-Record.

Felder

rgbBlue, rgbGreen, und rgbRed

rgbBlue, rgbGreen, und rgbRed sind die Intensitäten von Blau, Grün und Rot im Bitmap-Pixeleintrag.

rgbReserved

rgbReserved wird nicht verwendet und muß 0 sein.

TRGBTriple (Record) (Unit WinTypes)

Deklaration

```
TRGBTriple = record
   rgbtBlue: Byte;
   rgbtGreen: Byte;
   rgbtRed: Byte;
end;
```

Hinweise

Der TRGBTriple Record enthält RGB-Farbintensitäten für Bitmaps, wie im Feld bmciColors eines **TBitmapCoreInfo** Record.

Felder

rgbtBlue, rgbtGreen, und rgbtRed

rgbtBlue, rgbtGreen, und rgbtRed sind die Intensitäten von Blau, Grün und Rot im Bitmap-Pixeleintrag.

TTextMetric (Records) (Unit WinTypes)

Deklaration

```
TTextMetric = record
   tmHeight: Integer;
   tmAscent: Integer;
   tmDescent: Integer;
   tmInternalLeading: Integer;
   tmExternalLeading: Integer;
   tmAveCharWidth: Integer;
   tmMaxCharWidth: Integer;
   tmWeight: Integer;
   tmItalic: Byte;
   tmUnderlined: Byte;
   tmStruckOut: Byte;
   tmFirstChar: Byte;
   tmLastChar: Byte;
   tmDefaultChar: Byte;
   tmBreakChar: Byte;
   tmPitchAndFamily: Byte;
   tmCharSet: Byte;
   tmOverhang: Integer;
   tmDigitizedAspectX: Integer;
   tmDigitizedAspectY: Integer;
end;
```

Hinweise

Der TTextMetric Record enthält einige Felder, die eine Schrift in Einheiten beschreiben, die vom Abbildungsmodus des Bildschirms abhängen.

TTextMetric Records werden von den Funktionen **GetDeviceCaps** und **GetTextMetrics** verwendet.

Felder

tmHeight, tmAscent, und tmDescent

Das Feld tmHeight ist die Schrifthöhe nach Überschreiten (tmAscent) und Unterschreiten (tmDescent) der Grundlinie.

tmInternalLeading und tmExternalLeading

tmInternalLeading und tmExternalLeading geben den zulässigen Platz über tmHeight hinaus an.

tmInternalLeading ist Platz innerhalb der Grenze; tmExternalLeading wird zwischen Textzeilen eingefügt. Beide Felder können auf Null gesetzt werden.

tmAveCharWidth und tmMaxCharWidth

tmAveCharWidth und tmMaxCharWidth geben Durchschnitts- und Maximalbreite der Schriftzeichen an.

tmWeight

tmWeight ist die Dichte der Schrift.

tmItalic, tmUnderlined, und tmStruckOut

tmItalic, tmUnderlined, und tmStruckOut bedeuten kursiv, unterstrichen und durchstrichen, wenn sie nicht 0 sind.

tmFirstChar, tmLastChar und tmDefaultChar

Der Bereich definierter Zeichen ist durch tmFirstChar und tmLastChar gegeben, Zeichen außerhalb des Bereichs bekommen tmDefaultChar.

tmBreakChar

tmBreakChar ist das Zeichen, das begrenzt und zur Ausrichtung umbricht.

tmPitchAndFamily und tmCharSet

Schriftbreite, Schrifttyp und Zeichensatz der Schrift werden durch die Parameter tmPitchAndFamily und tmCharSet definiert.
Das niederwertige Bit in tmPitchAndFamily bestimmt die Neigung Buchstabenbreite der Schrift: Variabel bei gesetztem Bit, fest bei nicht gesetztem. Die vier hochwertigen Bits bestimmen den Schrifttyp, die mit den **ff_xxx Flags** eingestellt wird.

Der Zeichensatz wird in tmCharSet mit den entsprechenden Flags eingestellt.

tmOverhang

tmOverhang enthält die Sonderbreite, wen aus einer normalen eine fette Schrift hergestellt werden muß.

tmDigitizedAspectX und tmDigitizedAspectY

Die horizontalen und vertikalen Aspekte des Geräts, für das die Schrift entworfen wurde, in den Feldern tmDigitizedAspectX und tmDigitizedAspectY.

TWndClass (Record) (Unit WinTypes)

Deklaration

```
TWndClass = record
   style: Word;
   lpfnWndProc: TFarProc;
   cbClsExtra: Integer;
   cbWndExtra: Integer;
   hInstance: THandle;
   hIcon: HIcon;
   hCursor: HCursor;
   hbrBackground: HBrush;
   lpszMenuName: PChar;
   lpszClassName: PChar;
end;
```

Hinweise

Der TWndClass Record enthält die Attribute einer Fensterklasse, wenn sie mit der Funktion **RegisterClass** registriert wird.

Felder

style

style enthält den Stil der Klasse. Einer oder eine Kombination von **cs_xxx Konstanten**.

lpfnWndProc

lpfnWndProc zeigt die Fensterfunktion des Fensters, die Meldungen erhält und verarbeitet.

cbClsExtra

cbClsExtra ist die Anzahl von Bytes, die am Ende des TWndClass Record zugewiesen wird.

Diese werden die extra Bytes der Klasse genannt, Zugriff auf sie ist mit der Funktion **GetWindowLong** oder **GetWindowWord**; Setzen mit **SetWindowLong** oder **SetWindowWord** möglich.

cbWndExtra

cbWndExtra enthält die Anzahl von Bytes, die am Ende der Fensterinstanz zugewiesen werden.

hInstance

hInstance ist ein Instanz-Handle, das das Klassenmodul angibt und nicht Null sein darf.

hIcon, hCursor, und hbrBackground

hIcon, hCursor, und hbrBackground sind die Handles des Icon und Cursor der Klasse und deren Hintergrundfarbe.

Diese sollte einen Farbwert von **color_ Konstanten** +1 haben, oder einen Handle eines Hintergrundpinsels.

Ist hbrBackground 0, muß der Hintergrund bei jedem Malen des Client-Bereichs neu gemalt werden. Ob das nötig ist, kann die Meldung **wm_EraseBkgnd** oder Prüfen des Feldes fErase des **TPaintStruct** Record von **BeginPaint** ergeben.

lpszMenuName und lpszClassName

lpszMenuName und lpszClassName zeigen auf null-terminierte Strings, den Ressourcennamen des Klassenmenüs und den Namen der Klasse.

Unit WinProcs

Die Unit WinProcs definiert Funktions- und Pozedur-Kopfzeilen für Windows API.

Jede Routine aus den Windows-Bibliotheken kann über WinProcs angesprochen werden.

Zusammen mit der Unit **WinTypes** definiert WinProcs die Turbo Pascal Implementation von Windows API.

Siehe auch

Alphabetischer Index der Windows API Prozeduren und Funktionen
Kategorischer Index der Windows API Prozeduren und Funktionen

Dynamische Linkbibliotheken (DLLs)

In der Windows-Umgebung ermöglichen Dynamische Linkbibliotheken (DLLs) die gemeinsame Verwendung von Programmteilen und Ressourcen.

Eine DLL ist ein ausführbares Programm-Modul mit der Extension .DLL, das Code oder Ressourcen enthält, die von anderen DLLs oder Programmen verwendet werden.

So können zwei Programme (DLLs) eine Kopie einer Routine gemeinsam verwenden. Die .DLL-Datei muß vorhanden sein, wenn das Client-Programm läuft.

Das einer DLL ähnlichste Konzept in Turbo Pascal ist die **Unit**. Routinen in Units werden beim Linken in das Programm eingefügt ("statisch gelinkt"), wogegen DLL-Routinen zur Laufzeit verfügber gemacht werden ("dynamisch gelinkt").

Windows linkt dynamisch **Prozedur**- and **Funktions**-Aufrufe im Programm mit ihren Eintrittspunkten in den DLLs, die das Program verwendet.

Ein Turbo Pascal-Programm kann DLLs verwenden, die nicht in Turbo Pascal geschrieben sind, ebenso können Programme anderer Sprachen DLLs in Turbo Pascal verwenden.

Siehe auch

DLLs verwenden
DLLs erstellen
Import-Units

DLLs verwenden

Wenn ein Modul eine Prozedur oder Funktion aus einer DLL verwendet, muß es die Prozedur oder Funktion mit einer **external**-Deklaration importieren.

In importierten Prozeduren und Funktionen nimmt der Befehl **external** den Platz der Deklaration und der Anweisungen ein.

Importierte Prozeduren und Funktionen verhalten sich wie normale Routinen, allerdings müssen sie das Aufrufmodell FAR verwenden (eine **far**-Prozedurdirektive oder **{$F+} Compilerdirektive**.

Turbo Pascal importiert Prozeduren und Funktionen auf drei Weisen:

- **über den Namen**
- **über einen neuen Namen**
- **über einen Ordinalwert**

Obwohl eine DLL Variablen enthalten kann, können diese nicht in andere Module importiert werden. Jeder Zugriff auf Variablen einer DLL muß durch ein prozedurale Schnittstelle stattfinden.

Beispiel

Diese external-Deklaration importiert die Funktion GlobalAlloc aus der DLL KERNEL (Windows-Kernel):

```
function GlobalAlloc(Flags: Word; Bytes: Longint): THandle; far; external
  'KERNEL' index 15;
```

Hinweis: Der DLL-Name nach dem reservierten Wort **external** und der neue Name in einer **name-Klausel** müssen keine String-Literale sein. Jede String-Konstante ist zulässig.

Genauso kann die Ordinalzahl in einer **index-Klausel** ein beliebiger konstanter Integerausdruck sein. Beispiel:

```
const
  TestLib = 'TESTLIB';
  Ordinal = 5;

procedure ImportByName;    external TestLib;
procedure ImportByNewName; external TestLib name 'REALNAME';
procedure ImportByOrdinal; external TestLib index Ordinal;
```

Siehe auch

DLLs schreiben
Import von Units

Import von Routinen aus einer Dll über den Namen

Wenn Sie eine Routine einer DLL ohne **index** oder **name**-Klausel importieren, wird die Prozedur oder Funktion explizit über den Namen importiert.

Der Name ist der **Bezeichner** der Prozedur oder Funktion.

Beispiel

Hier wird die Prozedur ImportByName aus der DLL TESTLIB über den Namen IMPORTBYNAME importiert:

```
procedure ImportByName;    external 'TESTLIB';
```

Import von Routinen aus einer Dll über einen neuen Namen

Wenn Sie eine Routine einer DLL mit einer **name**-Klausel importieren, wird die Prozedur oder Funktion mit einem anderen Namen als dem Bezeichner importiert.

Beispiel

Hier wird die Prozedur ImportByNewName aus der DLL TESTLIB mit dem Namen ANOTHERNAME importiert:

```
procedure ImportByNewName; external 'TESTLIB' name 'ANOTHERNAME';
```

Import von Routinen aus einer Dll über einen Ordinalwert

Wenn Sie eine Routine einer DLL mit einer **index-Klausel** importieren, geschieht dies mit Hilfe eines Ordinalwertes.

Der Import über Ordinalzahlen verkürzt die Ladezeit des Moduls, weil Windows den Namen nicht in der Namenstabelle der DLL suchen muß.

Beispiel

Hier wird die Prozedur ImportByOrdinal als fünfter Eintrag der DLL namens TESTLIB importiert:

```
procedure ImportByOrdinal; external 'TESTLIB' index 5;
```

Import-Units

Sie können Deklarationen importierter Prozeduren und Funktionen direkt in das Programm schreiben, das sie importiert.

Sie werden allerdings normalerweise in einer Import-Unit zusammengefaßt, die Deklarationen aller Prozeduren und Funktionen in einer DLL enthält, zusammen mit allen Konstanten und Typen, die für den Zugriff auf die DLL notwendig sind.

Die Units **WinTypes** und **WinProcs** sind solche Import-Units.

Import-Units sind für den Zugriff auf eine DLL nicht unbedingt erforderlich, vereinfachen aber den Umgang mit Projekten, die viele DLLs verwenden. Beim Ändern einer DLL muß dann nur die Import-Unit angepaßt werden.

DLLs erstellen

Die Struktur einer Turbo Pascal DLL entspricht der eines Programms, nur steht anstatt der **program**-Kopfzeile eine **library**-Kopfzeile.

Die **library**-Kopfzeile weist Turbo Pascal an, eine ausführbare Datei mit der Kennzeichnung .DLL anstatt .EXE zu erzeugen.

Wenn Prozeduren und Funktionen von einer DLL exportiert werden sollen, müssen sie mit dem Befehl **export** compiliert werden.

Beispiel

Eine DLL mit zwei exportierten Funktionen:

```
library MinMax;

{export-Prozedurdirektive, Min und Max werden exportiert}

function Min(X, Y: Integer): Integer; export;
begin
  if X < Y then Min := X else Min := Y;
end;

function Max(X, Y: Integer): Integer; export;
begin
  if X > Y then Max := X else Max := Y;
end;

{exports-Klausel exportiert die Routinen und weist ihnen eine optionale
 Ordinalzahl zu}

exports
  Min index 1,
  Max index 2;

begin
end.
```

Bibliotheken bestehen oft aus mehreren **Units**. In diesen Fällen enthält der Quelltext der Bibliothek oft nur eine **uses**-Klausel, eine **exports**-Klausel und den Initialisierungs-Code der DLL.

Beispiel:

```
library Editors;

uses EdInit, EdInOut, EdFormat, EdPrint;

exports
  InitEditors index 1,
  DoneEditors index 2,
  InsertText index 3,
  DeleteSelection index 4,
  FormatSelection index 5,
  PrintSelection index 6,
   .
```

```
    .
  SetErrorHandler index 53;

begin
  InitLibrary;
end.
```

Siehe auch

DLLs verwenden
Import-Units

Ressourcen

Ressourcen sind Daten, die in der ausführbaren Datei (.EXE) eines Programms gespeichert werden, allerdings getrennt vom normalen Datensegment des Programms.

Ressourcen werden außerhalb des Programmcodes zugewiesen und spezifiziert, dann zur Erzeugung der ausführbaren Datei eines Programms zum compilierten Code des Programms hinzugefügt.

Folgende Ressourcen werden Sie am häufigsten erzeugen und verwenden:

- Bitmaps
- Zeichen-Strings
- Dialogfenster
- Symbole und Cursor
- Tastenschlüssel
- Menüs

Siehe auch

Ressourcen einer EXE-Datei hinzufügen
Ressourcen erzeugen
Ressourcen in eine Anwendung laden
Ressourcen-Compiler Kommandozeilen-Optionen
Hinweise zum Ressourcen-Compiler
Den Ressourcen-Compiler verwenden

Ressourcen erzeugen

Mit Turbo Pascal können Sie **Ressourcen** mit einem Ressourcen-Editor oder einem **Ressourcen-Compiler.** erzeugen.

Normalerweise erzeugen Sie eine binäre Ressourcendatei (<Dateiname>.RES) für jede Anwendung. Diese Ressourcendatei enthält binäre Informationen über alle Menüs, Dialoge, Bitmaps und andere Ressourcen, die von der zugehörigen Anwendung verwendet werden.

Sie können die **Compiler-Direktive $R <Dateiname>** verwenden, um die binäre Ressourcendatei während der Übersetzung zur ausführbaren Datei (.EXE) hinzuzufügen. Es resultiert damit eine Datei, die den compilierten Quelltext der Anwendung und deren Ressourcen enthält.

Sie müssen zudem Code schreiben, der die Ressourcen in den Speicher lädt. Jede Ressource muß einzeln in den Speicher geladen werden.

Siehe auch

Ressourcen einer EXE-Datei hinzufügen
Ressourcen in eine Anwendung laden
Ressourcen-Compiler Kommandozeilen-Optionen
Hinweise zum Ressourcen-Compiler
Ressourcen (definiert)
Den Ressourcen-Compiler verwenden

Ressourcen einer EXE-Datei hinzufügen

Ressourcen können auf dreierlei Weise einer ausführbaren Datei hinzugefügt werden:

- Verwenden Sie den Ressourcen-Editor, um Ressourcen von einer .RES-Datei in die bereits compilierte .EXE-Datei des Programms zu kopieren.
- Geben Sie die **Direktive $R <Dateiname>** in die Quelltext-Datei ein.
  Jedes Pascal-Programm kann nur eine Ressourcen-Datei haben (diese Ressourcen-Datei kann jedoch andere Ressourcen-Dateien enthalten). Alle diese Dateien müssen .RES-Dateien sein, die die Ressourcen in binärem Format speichern.
- Verwenden Sie den Microsoft Ressourcen-Compiler.

Siehe auch

Ressourcen erzeugen
Ressourcen in eine Anwendung laden
Ressourcen-Compiler Kommandozeilen-Optionen
Hinweise zum Ressourcen-Compiler
Ressourcen (definiert)
Den Ressourcen-Compiler verwenden

Den Ressourcen-Compiler verwenden

Der Ressourcen-Compiler ist eine DOS-Anwendung, die

1) als Eingabe eine Ressourcen-Makrodatei (<Dateiname>.RC) erwartet und eine binäre Ressourcen-Datei (<Dateiname>.RES) erzeugt.
2) die Ressourcen direkt in die ausführbare Datei einfügt (daher ist die Verwendung der **Direktive $R <Dateiname>**. hier nicht nötig).

Sie können beide Schritte kombinieren und eine bestehende .EXE-Datei modifizieren, um Ressourcen zu aktualisieren. In beiden Fällen muß der Quelltext Ihres Turbo Pascal Programms (<Dateiname>.PAS) Befehle enthalten, mit denen die Ressourcen in den Speicher geladen werden.

Sie können den Ressourcen-Compiler von der DOS-Ebene aus aufrufen oder aus einem DOS-Fenster unter Windows.

Syntax

```
{To generate a binary .RES file:}
  RC [options]  <.RC input file>   [.EXE output file]

{To attach the .RES file to the executable file:}
  RC <.RES binary file>

{To directly update the resources in the .EXE file:}
  RC <.RC input file>
```

Hinweis: Die Angaben in eckigen Klammern sind optional.

Siehe auch

Ressourcen einer EXE-Datei hinzufügen
Ressourcen erzeugen
Ressourcen in eine Anwendung laden
Ressourcen-Compiler Kommandozeilen-Optionen
Hinweise zum Ressourcen-Compiler
Ressourcen (definiert)

Ressourcen-Compiler Kommandozeilen-Optionen

Option	Bedeutung
-r	Eine .RES-Datei erzeugen
-l	Eine Anwendung erzeugen, die LIM 3.2 EMS verwendet
-e	Einen Treiber erzeugen, der EMS-Speicher verwendet
-m	Mehrfachte Instanzen-Flags setzen
-p	Eine private Library erzeugen
-t	Eine Anwendung erzeugen, die nur im protected Mode arbeitet
-v	Verbal (Statusmeldungen ausgeben)
-d	Ein Symbol definieren
-fo	.RES -Datei umbenennen
-fe	.EXE-Datei umbenennen
-i	Pfad für Suche nach INCLUDE-Dateien einfügen
-x	INCLUDE Umgebungsvariable ignorieren
-k	.DEF Dateireihenfolge der Segmente beibehalten (Segments nicht zum schnellen Laden sortieren

Siehe auch

Ressourcen einer EXE-Datei hinzufügen
Ressourcen erzeugen
Ressourcen in eine Anwendung laden
Hinweise zum Ressourcen-Compiler
Ressourcen (definiert)
Den Ressourcen-Compiler verwenden

Hinweise zum Ressourcen-Compiler

Es gibt einige Dinge, die zur Makro-Programmiersprache des Ressourcen-Compiler anzumerken sind:

1) Kommentare werden durch ein Semikolon (**;**) eingeleitet und enden am Zeilenende.
Sie können Kommentare gemäß C-Konvention schreiben (Text zwischen **/*** und ***/**).

2) Der Ressourcen-Compiler unterscheidet bei reservierten Worten (wie **begin** und **end**) nicht zwischen Groß- und Kleinschreibung.

3) Der Ressourcen-Compiler unterscheidet bei symbolischen Namen (wie cmFileNew und AboutBox) zwischen Groß- und Kleinschreibung.

4) Der Ressourcen-Compiler ignoriert Leerzeichen (Whitespace).

5) Sie müssen die **#include**-Direktive verwenden, um Include-Dateien mit Konstanten einbinden zu können. Diese werden (ähnlich wie in C) über den Befehl **#define** definiert.

6) Sie können mit der Direktive **#rcinclude** weitere Ressource-Makrodateien (*.RC oder *.DLG) aufnehmen; beispielsweise um compilierte Dialogfenster einzufügen.

7) IAm besten verwenden anstelle von Zahlen symbolische Konstantennamen für Konstanten, wie z.B. String-IDs, Menübefehl-IDs, etc.
Diese werden normalerweise über C-ähnliche Konstanten-Definitionen in einer Include-Datei definiert. Sie müssen zudem Pascal-Konstanten in Ihrem Programm definieren, um auf diese Konstanten symbolisch zu referenzieren.

8) Jede Menüoptions-ID und String -D muß eindeutig sein.

9) Menüs können beliebig geschachtelt sein (erzeugt hierarchische Menüs).

10) In Menü- und Dialogkontrollelementen weist das Et-Zeichen (**&**) dem Program an, das nächste Zeichen zu unterstreichen.

11) In Menüeinträgen erzeugt die Zeichenfolge **\t** ein Tabulatorzeichen zur Ausrichtung von Text. Damit werden oft die Angaben von Tastenkürzeln bündig ausgerichtet.

12) Tastenkürzel müssen in einer Tastenbelegungs-Tabelle definiert werden. Der Eintrag im Menübegriff erzeugt nicht automatisch die Tastenbelegung.

13) Kontrollelemente, wie statischer Text, auf die nie aus dem Programm heraus zugegriffen wird, haben die ID -1.

14) Die Position von Dialog- und Kontrollelementen wird in der Reihenfolge Links, Oben, Breite, Höhe definiert.

15) Menübegriffe, Dialoge und Dialogkontrollelemente, wie Bildlaufleisten, Eingabefelder und Aktionsschalter, können durch die Verwendung von Konstanten, die der Windows-Konvention entsprechen, verändert werden. Verwenden Sie den bitweisen Operator **or** (vertikaler Strich (|)), um sie hinzuzufügen.

16) Ressourcen-Makrodateien müssen mit dem Ressourcen-Compiler compiliert werden.

17) In den meisten Fällen ist es einfacher, einen Ressourcen-Editor einzusetzen, als eine komplexe Ressourcen-Definition mit Hilfe von Makrodateien zu schreiben.

18) Sie müssen in Ihrem Turbo Pascal Programm Routinen zum Laden und Einsatz der Ressourcen definieren.

Siehe auch

Ressourcen einer EXE-Datei hinzufügen
Ressourcen erzeugen
Ressourcen in eine Anwendung laden
Ressourcen-Compiler Kommandozeilen-Optionen
Ressourcen (definiert)
Den Ressourcen-Compiler verwenden

Ressourcen in eine Anwendung laden

Sobald eine Ressource in eine ausführbare Datei eingefügt wurde, muß die Anwendung die Ressource explizit laden, bevor sie verwendet werden kann. Wie eine bestimmte Ressource geladen wird, hängt vom Ressourcentyp ab.

Menü-Ressourcen

Sie laden eine Menü-Ressource, indem Sie die Funktion **LoadMenu** mit dem Menü-ID-String aufrufen, wenn ein neues Fensterobjekt erzeugt wird.

Tastenbelegungs-Resourcen

Diese Ressourcen werden in einer Tastenbelegungs-Tabelle gespeichert.

Sie laden eine Tastenbelegungs-Tabelle, indem Sie die Funktion **LoadAccelerators** verwenden, die ein Handle auf die Tabelle liefert.
Für diese Tabelle gilt:

- Jede Anwendung kann nur über eine Tastenbelegungs-Tabelle verfügen.
- Die Objekte der Anwendung reservieren ein Objektfeld, **HAccTable**, zum Speichern eines Handle auf die Tastenbelegungs-Ressource.
- Normalerweise wird die Tastenbelegungs-Ressource in die Methode **InitInstance** des Anwendungsobjekts geladen.

Dialogfenster-Ressourcen

Dialogfenster bilden den einzigen Ressourcentyp, zu dem ObjectWindows Objekttypen direkt korrespondieren.

TDialog und seine abgeleiteten Typen, einschließlich **TDlgWindow**, definieren Schnittstellen-Objekte, die Dialogfenster-Ressourcen verwenden.

Jedes Dialogfenster-Objekt ist normalerweise einer Dialogfenster-Ressource zugeordnet, diese definiert seine Größe, Positon und die zugehörigen Kontrollelemente, wie Aktionsschalter und Eingabeaufzeichnungslisten.

Cursor- und Symbol-Ressourcen

Jeder Fenster-Objekttyp hat spezielle Atttribute, die sogenannten Registrierungs-Attribute, zu denen der Cursor und die Symbole des Fensters gehören.
Um diese Attribute für einen Fenstertyp zu setzen, müssen Sie eine Methode namens **GetWindowClass** und eine namanes **GetClassName.** definieren.

Der Cursor wird für jedes Fenster spezifiziert, während ein Symbol die gesamte Anwendung repräsentiert.

(Eine Ausnahme zu der Regel "Ein Symbol pro Anwendung" bilden Anwendungen, die dem MDI-Standard (Multiple Document Interface) entsprechen, dort hat jedes MDI-Child-Fenster sein eigenens Symbol.)

String-Ressourcen

Zwei Hauptgründe für die Definition der Strings einer Anwendung als Ressourcen sind,

1) die Anpassung der Anwendung auf bestimmte Verwendungszwecke zu erleichtern und
2) tdie Übersetzung der Anwendung in eine andere Sprache zu erleichtern.

Wenn die Strings als Ressourcen definiert werden, werden Sie in einer String-Tabelle in der ausführbaren Datei der Anwendung gespeichert.

In jeder ausführbaren Datei ist nur eine String-Tabelle zulässig.

Sie laden einen String aus einer String-Tabelle in den Puffer des Datensegments Ihrer Anwendung, imdem Sie die Funktion **LoadString** verwenden. Sie können eine String-Ressource einsetzen,

- um Text in einem Meldungsfenster anzuzeigen.
- um in Ihrem Quelltext Einträge an ein Menü anzuhängen oder in ein Menü einzufügen.

Bitmap-Ressourcen

Die Funktion **LoadBitmap** lädt Bitmap-Ressourcen, indem sie das Bitmap-Element in den Speicher lädt und das zugehörige Handle zurückgibt.

Sobald ein Bitmap-Element geladen wurde, bleibt es solange im Speicher, bis sie es explizit löschen (mit **DeleteObject**)..

Windows enthält vordefinierte Bitmaps als Teil der grafischen Schnittstelle von Windows. Ihre Anwendung kann diese Bitmaps laden, indem im Aufruf von LoadBitmap HInstance durch 0 ersetzt wird.

Sobald die Bitmap geladen wurden, kann die Anwendung sie in verschiedener Weise verwenden:

- um ein Bild in der Anzeige zu zeichnen.
- um einen Füllmuster zu erzeugen, mit dem Sie einen Bereich der Anzeige ausfüllen oder das Sie als Fensterhintergrund verwenden können.
- To display pictures rather than text for menu items or for items in a list box.

Siehe auch

Ressourcen einer EXE-Datei hinzufügen
Ressourcen erzeugen
Ressourcen-Compiler Kommandozeilen-Optionen
Hinweise zum Ressourcen-Compiler
Ressourcen (definiert)
Den Ressourcen-Compiler verwenden