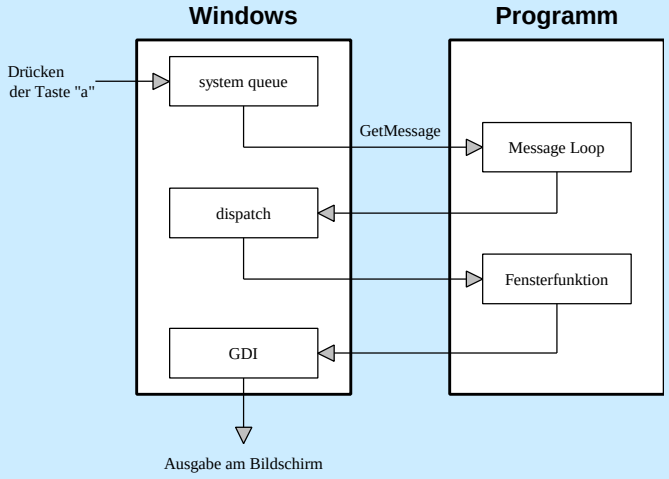
KV Betriebssysteme
Windowsprogrammierung

Peter R. Dietmüller
Inst. f. Informationsverarbeitung und Mikroprozessortechnik (FIM)
Johannes Kepler Univ. Linz, Altenbergerstr. 69, A-4040 Linz, Austria
E-Mail: dietmueller@fim.uni-linz.ac.at

Windows - Programmiermodell



Aufbau eines Windowsprogramm

Ein einfaches Windowsprogramm, das ein Fenster öffnet, besteht aus zwei Funktionen:

- **WinMain ("Hauptprogramm")**
  - Globale Initialisierung
  - Lokale Initialisierung
  - "message loop"
- **Fensterfunktion**
  - Darstellung des Fensters

WinMain

```
int WINAPI WinMain (  HINSTANCE hInstCurr,
                      HINSTANCE hInstPrev,
                      LPSTR     lpszCmd,
                      int       nCmdShow) {
  /* -- Globale Initialisierung -- */
  /* -- Lokale Initialisierung -- */
  /* -- „Message Loop“ -- */
  return 0;
}
```

```
typedef void *HINSTANCE;
typedef char *LPSTR;
```

Initialisierung

Instanzen

- **Ein Programm kann mehrmals aufgerufen werden.**
- **Dabei wird nur einmal der Code geladen.**
- **Die Daten werden für jede Instanz geladen.**

Globale Initialisierung

- **Initialisierung unabhängig von einer bestimmten Instanz**
- **Sie wird in der Regel nur einmal beim Aufruf der ersten Instanz durchgeführt.**

Lokale Initialisierung

- **Initialisierung einer bestimmten Instanz**

Initialisierung (2)

```
int WINAPI WinMain
  (hInstCurr, hInstPrev, lpszCmd, nCmdShow) {

  /* -- Globale Initialisierung -- */
  if (!hInstPrev)                          /* erste Instanz? */
    if (!InitApplication(hInstCurr)) return 0;

  /* -- Lokale Initialisierung -- */
  if (!InitInstance(hInstCurr, nCmdShow)) return 0;

  /* -- „Message Loop“ -- */
  return 0;
}
```

Initialisierung (3)

Erkennung der ersten Instanz

- **Die erste Instanz einer Applikation erkennt man am Parameter hInstPrev. Er ist NULL, wenn das Programm das erste Mal aufgerufen wird.**

Globale Initialisierung

- **Funktion: int InitApplication(HINSTANCE hInstCurr);**
- **Registrierung der benötigten Fensterklassen**

Lokale Initialisierung (Funktion InitInstance)

- **int InitInstance(HINSTANCE hInstCurr, int nCmdShow);**
- **Öffnen des Hauptfensters der Applikation**

Fensterklassen

Was ist eine Fensterklasse (Window Class)?

- **Menge von Eigenschaften**
- **Template für neue Fenster**
- **Fensterklassen sind Prozeß-spezifisch.**
- **Alle Fenster einer Klasse teilen sich eine Fensterfunktion.**

Was ist eine Fensterfunktion (Window Procedure)?

- **Verarbeitet Nachrichten für alle Fenster seiner Klasse.**
- **Kontrolliert das Verhalten und Erscheinungsbild aller Fenster seiner Klasse. Daher haben alle Fenster einer Klasse dasselbe Erscheinungsbild und Verhalten.**

Arbeiten mit Fensterklassen

1. Registrierung einer Fensterklasse

- **Funktion:**
  ATOM RegisterClass(CONST WNDCLASS *lpWndClass);
- **Durch die Registrierung bekommt eine Fensterklasse einen Namen, eine Fensterfunktion und Stile, wie z.B. Mauscursor, Hintergrundfarbe, ...**

2. Öffnen eines Fensters

- **Funktion:**
  HWND CreateWindow(LPCTSTR lpClassName, ...);
- **Mit der Funktion CreateWindow wird ein Fenster einer bestimmten Klasse angelegt.**

Struktur WNDCLASS

```
typedef struct _WNDCLASS {
   UINT        style;          /* Klassenstile          */
   WNDPROC     lpfnWndProc;    /* Fensterfunktion       */
   int         cbClsExtra;     /* Extra Klassenspeicher */
   int         cbWndExtra;     /* Extra Fensterspeicher */
   HINSTANCE   hInstance;      /* Instanz               */
   HICON       hIcon;          /* Icon, NULL -> kein I. */
   HCURSOR     hCursor;        /* Mauscursor, NULL      */
   HBRUSH      hbrBackground;  /* Hintergrundfarbe      */
   LPCTSTR     lpszMenuName;   /* Menüname, NULL        */
   LPCTSTR     lpszClassName;  /* Klassenname           */
} WNDCLASS, *PWNDCLASS;
```

Globale Initialisierung - Beispiel

```
static char szWndClassName[] = "CHelloWorld";

static int InitApplication(HINSTANCE hInstCurr) {
    WNDCLASS wc;   /* window class */
    wc.style                = CS_HREDRAW | CS_VREDRAW ;
    wc.lpfnWndProc          = (WNDPROC)WndProc ;
    wc.cbClsExtra           = 0 ;
    wc.cbWndExtra           = 0 ;
    wc.hInstance            = hInstCurr ;
    wc.hIcon                = LoadIcon(NULL, IDI_WINLOGO);
    wc.hCursor              = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground        = (HBRUSH)(COLOR_WINDOW + 1);
    wc.lpszMenuName         = NULL;
    wc.lpszClassName        = szWndClassName;
    return RegisterClass(&wc);
}
```

Fensterklassen

Zusammenfassung

- **Eine Fensterklasse faßt die gemeinsamen Eigenschaften aller zu dieser Klasse gehörenden Fenster zusammen. Dazu gehört zum Beispiel auch die Ereignisverarbeitungsprozedur.**
- **Das bedeutet, daß alle Fenster einer Klasse im wesentlichen dieselbe Funktionalität und dasselbe Aussehen haben.**
- **Windows stellt einige Standardklassen (Button, Listbox, Combobox, Eingabefeld, Radio Button, ...) zur Verfügung. Darüber hinaus kann jedes Programm eigene Fensterklassen anlegen.**

CreateWindow

```
HWND CreateWindow(
  LPCTSTR      lpClassName,     // registered class name
  LPCTSTR      lpWindowName, // window name
  DWORD        dwStyle,         // window style
  int          x,               // horizontal position of window
  int          y,               // vertical position of window
  int          nWidth,          // window width
  int          nHeight,         // window height
  HWND         hWndParent,// handle to parent or owner window
  HMENU        hMenu,           // menu handle or child identifier
  HINSTANCE    hInstance,       // handle to application instance
  LPVOID       lpParam          // window-creation data
);
```

Lokale Initialisierung

```
static int InitInstance(HINSTANCE hInstCurr, int nCmdShow) {

    HWND hWnd;                                  // Fenster-Handle, eindeutige ID
    hWnd = CreateWindow(szWndClassName,         // Klassenname
        "Hello World",                          // Fenstertitel
        WS_OVERLAPPEDWINDOW,                    // Fensterstil
        CW_USEDEFAULT, CW_USEDEFAULT,           // x,y-Koordinate
        CW_USEDEFAULT, CW_USEDEFAULT,           // Breite und Höhe
        NULL, NULL, hInstCurr, NULL);           // Parent, Menü, Instanz, Zusatz
    if (hWnd) {
        ShowWindow(hWnd, nCmdShow);             // Fenster anzeigen
        UpdateWindow(hWnd);                     // Fensterinhalt erneuern
        return TRUE;
    }
    return FALSE;
}
```

„Message Loop“

```
int WINAPI WinMain (hInstCurr, hInstPrev,
                    lpszCmd, nCmdShow) {
   MSG msg ;

   ... (Globale und lokale Initialisierung) ...

   while (GetMessage (&msg, NULL, 0, 0)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return msg.wParam;
}
```

„Message Loop“

Bestandteile

- **GetMessage liest die nächste Nachricht aus der Warteschlange.**
- **TranslateMessage übersetzt "virtual key messages" in "character messages" (z.B. Kombinationen von WM_KEYDOWN und WM_KEYUP).**
- **DispatchMessage sendet die Nachricht an das zuständige Fenster (=> Fensterfunktion).**
- **Die Schleife wird beendet, wenn GetMessage 0 zurückliefert. Das ist der Fall, wenn die Nachricht "WM_QUIT" in der Warteschlange steht.**

Fensterfunktion

Schnittstelle

- **Jedes Fenster (eig. Fensterklasse) hat eine Fensterfunktion.**
- **Der Funktionsname kann frei gewählt werden.**
- **LRESULT WINAPI WndProc(**

```
HWND hwnd,          /* handle of window          */
UINT uMsg,          /* message identifier        */
WPARAM wParam,      /* 1st message parameter     */
LPARAM lParam);     /* 2nd message parameter     */
```

(The WINAPI modifier is used because the window procedure must be declared with the standard call calling convention.)

Fensterfunktion (2)

Aufgabe

- **Empfang von Nachrichten**
- **„Geeignete“ Reaktion auf die Nachrichten**
  - Man kann auf die Nachricht in seiner Fensterfunktion reagieren oder
  - sie an die „**Default Window Procedure**“ (Standardfunktion) weiterleiten. In den Standardfunktionen ist zum Beispiel das Verschieben eines Fensters oder das Verkleinern und Vergrößern implementiert.

Fensterfunktion (3)

```
LRESULT WINAPI WndProc (HWND ...) {

    switch (message) {

        case WM_DESTROY: {
            PostQuitMessage (0) ; /* post WM_QUIT */
            return 0;
        }

        default:
            return DefWindowProc (hwnd, message, wParam, lParam) ;
    }
}
```

Standardfunktionen

Funktion	ist vorgesehen für
DefDlgProc	Dialoge
DefFrameProc	MDI Frame Window
DefMDIChildProc	MDI Client Window
DefScreenSaverProc	Screen Saver
DefWindowProc	"normales" Fenster

Nachrichten

WM_DESTROY:
- **Das Fenster wird "zerstört“. Es ist eine der letzten Nachrichten, bevor das Fenster aufhört zu existieren.**

WM_MOUSEMOVE:
- **Die Maus wurde über das Fenster bewegt.**

WM_MOVE:
- **Die Position des Fensters hat sich geändert.**

WM_PAINT:
- **Inhalt des Fensters soll neu gezeichnet werden.**

WM_QUIT:
- **Das Programm soll beendet werden.**

WM_SIZE:
- **Die Größe des Fensters hat sich geändert.**

Nachrichten (2)

WM_DESTROY
- **keine Parameter**

WM_MOUSEMOVE
- **wParam = key flags**
- **LOWORD(lParam) = hor. Pos.**
- **HIWORD(lParam) = ver. Pos.**

WM_MOVE
- **LOWORD(lParam) = hor. Pos.**
- **HIWORD(lParam) = vert. Pos.**

WM_PAINT
- **wParam = Device Context Handle**

WM_QUIT
- **wParam = exit code**

WM_SIZE
- **wParam = sizing-type flag**
- **LOWORD(lParam) = Breite**
- **HIWORD(lParam) = Höhe**

Beispiel

Das Beispiel öffnet ein Fenster, in dem der Text "Hello World" angezeigt wird.

Dazu wird im Programm eine neue Fensterklasse "CHelloWorld" angelegt, dessen Fensterfunktion

- **beim Schließen des Fensters das Programm beendet und**
- **bei WM_PAINT den Text "Hello World" ausgibt.**

WHello.c

Lebenszyklus eines Fenster

Funktion CreateWindow
- **Sendet die Nachricht WM_CREATE**
- **Gibt ein „Window-Handle“ (eindeutige Nummer) zurück**

Nachricht WM_CREATE
- **Ressourcen für das Fenster anlegen**

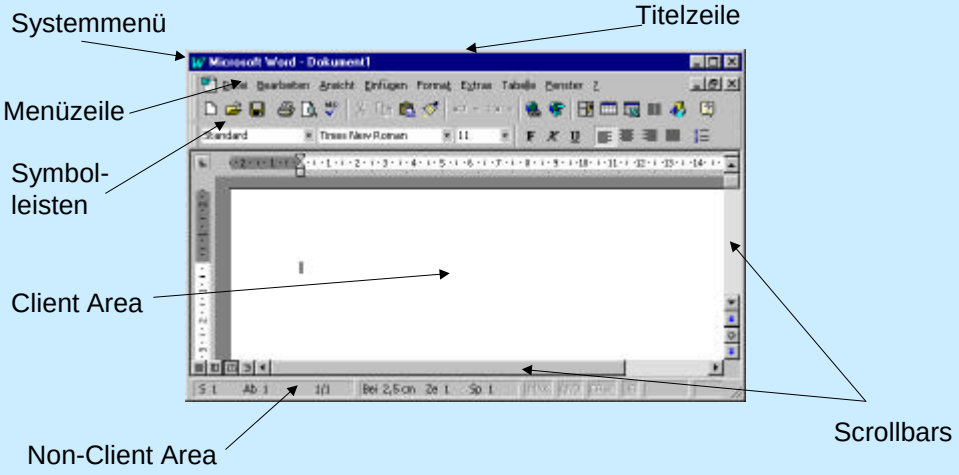
Fenster anzeigen
- **Fensterstil WS_VISIBLE (CreateWindow), Funktion ShowWindow**

Funktion DestroyWindow
- **Schließt das Fenster**
- **Sendet die Nachricht WM_DESTROY**

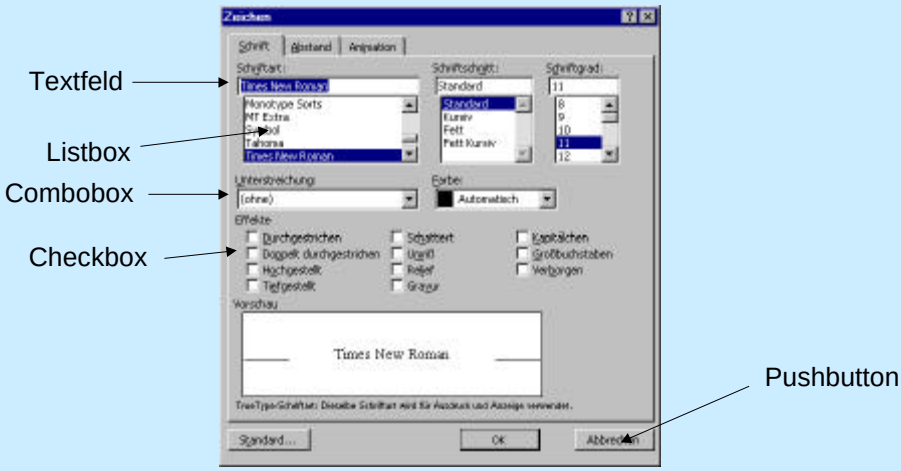
Nachricht WM_DESTROY
- **Angelegte Ressourcen freigeben**
- **Bei Bedarf Programm schließen durch Aufruf von PostQuitMessage (Dadurch wird WM_QUIT gesendet, was die message loop beendet).**

KV Betriebssysteme
Windowsprogrammierung

Fensterelemente



Fensterelemente



Fensterstatus

enabled/disabled

- **Ein Fenster, das enabled ("aktiv") ist, kann Eingabeereignisse empfangen und verarbeiten. Dieser Status wird mit der Funktion EnableWindow gesetzt.**

minimized/maximized/restored

- **Dabei wird die Größe des Fensters festgelegt. Minimized bedeutet Icon-Größe, Maximized Vollbild-Größe und restored bedeutet normale Größe.**

hidden/visible

- **Ein Fenster muß nicht unbedingt sichtbar sein. Ohne besondere Vorkehrungen wird ein neues Fenster erst durch ShowWindow sichtbar.**

Graphics Device Interface

- **Modul für Zeichenoperationen (z.B.: LineTo, Ellipse, …).**
- **Die Zeichenoperationen sind geräteunabhängig.**
- **Die Verbindung von Zeichenoperationen zum Ausgabegerät wird über den sog. "Gerätekontext" (device context) hergestellt.**
- **Es gibt Gerätekontexte für Bildschirm, Drucker und Bitmap.**
- **Ausgabeoperationen können damit für Bildschirm, Drucker oder Bitmaps gleich programmiert werden.**
- **Allerdings gibt es in manchen Bereichen (z.B. Auflösung) Unterschiede zwischen den Fähigkeiten der Ausgabemedien, wodurch nicht immer alle Zeichenoperationen verfügbar sind.**

Device Context

Ein Device Context definiert

- **das Ausgabemedium**
- **die Koordinateneinheiten (Pixel, Zoll,…)**
- **die gerade gewählten Zeichenobjekte**
- **die aktuelle Zeichenposition (ähnlich wie log. Cursor)**

Zeichenobjekte sind

- **Stifte (für Linien)**
- **Pinsel (für flächendeckende Zeichenoperationen)**
- **Schriftarten (für Text)**
- **logische Farben (für Hintergrundfarbe, Textfarbe,…)**

Anzeige des Fensterinhalt

Windows merkt sich den Inhalt eines Fensters nicht. Die Verantwortung über die korrekte Anzeige des Inhalt wird der Fensterfunktion übertragen.

Immer wenn ein Teil eines Fensters oder das ganze Fenster neu gezeichnet werden muß, sendet Windows die Nachricht WM_PAINT.

Windows merkt sich welche Teile eines Fensters neu zu zeichnen sind (Update Region) und kann dadurch das Neuzeichnen beschleunigen (Clipping).

Update Region

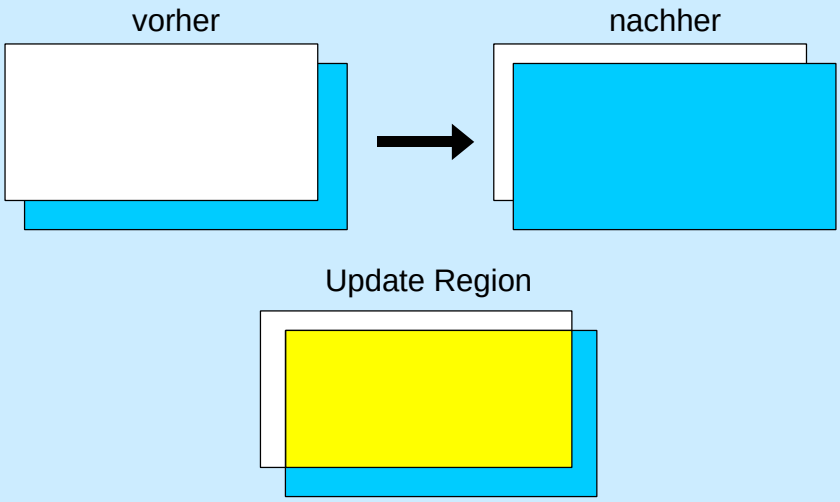
Die „Update Region“ eines Fensters ist jener Teil dieses Fensters, der ungültig („invalid“) ist und neu gezeichnet werden muß. Windows führt dazu eine Liste von ungültigen Bereichen (Rechtecke, Polygone, Kreise, ...).

Mit den Funktionen InvalidateRect und InvalidateRgn können Teile eines Fensters für „ungültig“ erklärt werden und werden somit in die „Update Region“ aufgenommen.

Mit ValidateRect und ValidateRgn passiert genau das Gegenteil. Die angegebenen Bereiche werden aus der Liste genommen.

Windows selbst fügt Bereiche in diese Liste hinzu, z.B. wenn ein Fenster verschoben wird oder in den Vordergrund kommt.

Update Region



Nachricht WM_PAINT

Reaktion:

- **Aufruf der Funktionen BeginPaint**
  - Liefert einen Device Context.
  - Die Clipping Region des Device Context wird auf die Update Region des Fenster gesetzt.
  - Sendet die Nachricht WM_NCPAINT und WM_ERASEBKGND.
  - Löscht die Update Region des Fensters.
- **Ausgabe / Zeichnen des Fensterinhaltes**
  - Zeichnen des (gesamten?) Fensterinhalt mit Hilfe der GDI-Funktionen (mit dem Device Context von BeginPaint)
  - Zeichnen über die Clipping Region hinaus wird ignoriert (Clipping).
- **Aufruf der Funktion EndPaint**
  - Gibt den Device Context wieder frei.
  - Zeigt das Caret wieder an.

Nachricht WM_ERASEBKGND

In der Fensterklasse kann man eine Hintergrundfarbe festlegen. Sie wird benutzt, um den Inhalt des Fensters zu löschen.

Dazu wird die Nachricht WM_ERASEBKGND von der Funktion BeginPaint (noch vor WM_PAINT) an das Fenster gesendet.

Überläßt man die Nachricht der Default Window Procedure, dann werden jene Teile des Fensters, die neu zu zeichnen sind, mit der Hintergrundfarbe ausgefüllt.

Bildschirmausgabe

Standardmodell

- **Der Fensterinhalt wird nur in WM_PAINT ausgegeben.**
- **Bildschirmausgabe:**
  - asynchron: InvalidateRect(), warten bis WM_PAINT gesendet wird.
  - synchron: InvalidateRect(), UpdateWindow() (sendet sofort die Nachricht WM_PAINT)
- **Vorteile:**
  - klares Modell
  - keine Codeverdopplung
  - wenig fehleranfällig
- **Nachteile:**
  - Langsam bei häufigen kleinen Änderungen

Bildschirmausgabe

UpdateWindow

- **BOOL UpdateWindow(HWND hWnd);**
- **Die Funktion veranlaßt ein Neuzeichnen der Client Area des angegebenen Fensters, indem eine Nachricht WM_PAINT an das Fenster gesendet wird, wenn die Update Region des Fensters nicht leer ist. Wenn sie leer ist, wird keine Nachricht gesendet.**
- **Die Nachricht WM_PAINT wird direkt an die Fenster-funktion gesandt. Die Queue wird dabei übergangen.**
- **Bei Erfolg gibt die Funktion einen Wert ungleich 0 zurück.**

Bildschirmausgabe

Variante #1

- **In WM_PAINT wird der aktuelle Fensterinhalt ausgegeben.**
- **Bildschirmausgabe:**
  - Aufruf von GetDC()
  - Aufruf einer / mehrerer Zeichenfunktionen
  - ReleaseDC()
- **Vorteile:**
  - Schnelle Ausgabe und Reaktion
- **Nachteile:**
  - Code für Bildschirmausgabe existiert mehrfach in ähnlicher Form, weil WM_PAINT auch notwendig ist.

Bildschirmausgabe

Variante #2

- **Der aktuelle Fensterinhalt wird in einer Bitmap gespeichert.**
- **In WM_PAINT wird die Bitmap ausgegeben.**
- **Bildschirmausgabe:**
  - Zeichnen in die Bitmap und Aufruf von InvalidateRect()
- **Vorteile:**
  - Verhindert das Flackern, wenn keine Hintergrundfarbe verwendet wird.
  - Keine Codeverdopplung
- **Nachteile:**
  - Erhöhter Speicher- und Ressourcenbedarf

Display Device Context

Typ:

- **common: Device Context für die Client Area**
- **window: Device Context für das gesamte Fenster**
- **class, parent, private: Spezielle Varianten**

Bei WM_PAINT:

- **Anfordern mit BeginPaint (meist Typ common).**
- **Freigabe mit EndPaint.**

Direkte Ausgabe im Programm

- **Anfordern mit GetDC (meist Typ common) oder GetWindowDC (Typ window).**
- **Freigabe mit ReleaseDC.**

Zeichenfunktionen

MoveToEx

- **BOOL MoveToEx(HDC hdc, int x, int y, LPPOINT lpPoint);**
- **Setzt die aktuelle Zeichenposition auf einen bestimmten Punkt. Gibt die alte Zeichenposition in lpPoint zurück. Bei Erfolg ist der Rückgabewert ungleich 0.**

LineTo

- **BOOL LineTo(HDC hdc, int x, int y);**
- **Zieht eine Linie von der aktuellen Zeichenposition zu einem bestimmten Punkt und versetzt die Zeichenposition dorthin. Zum Zeichnen wird der aktuelle Stift verwendet. Bei Erfolg ist der Rückgabewert ungleich 0.**

Zeichenfunktionen

Rectangle

- **BOOL Rectangle(HDC hdc,
  int left, int top, int right, int bottom);**
- **Ein Rechteck wird mit dem aktuellen Stift gezeichnet. Der innere Bereich wird mit dem aktuellen Pinsel gefüllt. Zu beachten ist, daß das Rechteck nur bis right-1 und bottom-1 gezeichnet wird. Bei Erfolg ist der Rückgabewert ungleich 0.**

RGB

- **COLORREF RGB(BYTE red, BYTE green, BYTE blue);**
- **Dieses Makro erzeugt aus einzelnen RGB-Werten einen 32 Bit Wert vom Typ COLORREF, der eine Farbe exakt definiert.**

Zeichenfunktionen

SetPixel

- **COLORREF SetPixel(HDC hdc, int x, int y,
  COLORREF color);**
- **Der Rückgabewert ist die Farbe des gezeichneten Punkts. Diese kann von color abweichen, wenn für die gewünschte Farbe eine Näherung verwendet werden mußte. Der Rückgabewert ist -1, wenn der Punkt außerhalb des Clippingbereichs liegt.**
- **Eine Näherung für die Wunschfarbe muß verwendet werden, wenn aufgrund einer geringen Farbtiefe der Anzeige nicht jede RGB - Kombination zur Verfügung steht.**
- **Nicht jedes Ausgabegerät muß SetPixel unterstützen.**

Zeichenfunktionen

TextOut

- **BOOL TextOut(HDC hdc, int x, int y,
  LPCSTR text, int textLen);**
- **Der String text wird an der angegebenen Position gezeichnet. Es wird der aktuelle Zeichensatz, die aktuelle Hintergrund- und Textfarbe verwendet. Der String muß nicht nullterminiert sein, die Stringlänge wird in textLen übergeben. Bei Erfolg ist der Rückgabewert ungleich 0.**

Zeichenattribute in einem DC

Bitmap

- **CreateBitmap, CreateBitmapIndirect, CreateCompatibleBitmap, ...**

Pinsel (Brush)

- **CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, ...**

Schriftart (Font)

- **CreateFont, CreateFontIndirect**

Stift (Pen)

- **CreatePen, CreatePenIndirect**

Region

- **CreateEllipticRgn,, CreatePolygonRgn, CreateRectRgn, ...**

Wahl von Zeichenattributen

SelectObject

- **HGDIOBJ SelectObject(HDC hdc, HGDIOBJ obj);**
- **Zunächst besitzt jeder Gerätekontext einen Standardwert für den aktuellen Stift, Pinsel und Zeichensatz. Diesen Standard kann man jederzeit durch andere Zeichenobjekte ersetzen.**
- **Das durch obj übergebene Zeichenobjekt wird in den Gerätekontext selektiert und ersetzt damit das vorher selektierte Objekt gleichen Typs. Der Rückgabewert der Funktion ist ein Handle auf das ersetzte Objekt oder NULL, wenn der Aufruf fehlgeschlagen ist.**

Zeichenobjekte

Erzeugen

- **Funktionen CreateXXX und CreateIndirectXXX**
- **Bei den CreateIndirectXXX Funktionen werden die Objekteigenschaften in einer eigenen Struktur definiert, sonst direkt als Parameter.**

Freigeben

- **Funktion DeleteObject**
- **Ein Objekt darf zu diesem Zeitpunkt nicht mehr in einem Gerätekontext selektiert sein. Dazu kann man den ursprünglichen Wert wieder selektieren.**

Zeichenobjekte

```
HPEN hPen, hPenOld;
HDC hdc;

hPen = CreatePen(PS_SOLID, 6, RGB(0, 0, 255));
hPenOld = SelectObject(hdc, hPen);

… (z.B. mit dem neuen Stift zeichnen)

SelectObject(hdc, hPenOld);
DeleteObject(hPen);
```

Freigeben eines Zeichenobjektes

DeleteObject

- **BOOL DeleteObject(HGDIOBJ obj);**
- **Das bezeichnete Objekt und die damit verbundenen Ressourcen werden freigegeben. Bei Erfolg ist der Rückgabewert ungleich 0.**
- **Ein Objekt darf nicht in einem Gerätekontext selektiert sein, wenn es gelöscht werden soll.**

Erzeugen eines Stift

CreatePen

- **HPEN CreatePen(int penStyle, int width, COLORREF color);**
- **Ein Stift mit der angegebenen Farbe und Dicke wird im angegebenen Stil generiert. Der Rückgabewert der Funktion ist ein Handle auf den Stift oder NULL, wenn der Aufruf fehlgeschlagen ist.**
- **Einige mögliche Werte für penStyle:**
  - PS_SOLID durchgehend gezeichnet
  - PS_DOT Stift zeichnet punktiert, nur mit Stiftbreite 1

Erzeugen eines Pinsel

CreateBrushIndirect

- **HBRUSH CreateBrushIndirect(CONST LOGBRUSH* lb);**
- **Ein Pinsel mit den in der Struktur lb definierten Eigenschaften wird generiert. Der Rückgabewert der Funktion ist ein Handle auf den Pinsel oder NULL, wenn der Aufruf fehlgeschlagen ist.**
- **typedef struct {**

```
  UINT        lbStyle;
  COLORREF    lbColor; /* Farbe des Pinsels*/
  int         lbHatch;
} LOGBRUSH;
```

Erzeugen eines Pinsel

Für lbStyle existieren u. a. die folgenden Konstanten:

- **BS_SOLID** **Pinsel für einfache Farbflächen**
- **BS_HATCHED** **Pinsel für Schraffuren**

Ist lbStyle auf BS_SOLID gesetzt, wird der Wert von lbHatch ignoriert. Für Pinsel mit dem BS_HATCHED Stil existieren für lbHatch u. a. folgende Konstanten:

- **HS_BDIAGONAL** **45 Grad steigend schraffiert**
- **HS_FDIAGONAL** **45 Grad fallend schraffiert**
- **HS_HORIZONTAL** **horizontale Schraffur**
- **HS_VERTICAL** **vertikale Schraffur**

Erzeugen einer Schriftart

CreateFontIndirect

- **HFONT CreateFontIndirect(LOGFONT* lf);**
- **Ein Zeichensatz mit den in der Struktur lf definierten Eigenschaften wird selektiert. Der Rückgabewert der Funktion ist ein Handle auf den Zeichensatz oder NULL, wenn der Aufruf fehlgeschlagen ist.**

Erzeugen einer Schriftart

```
typedef struct tagLOGFONT {
  LONG lfHeight;                  /* Zeichenhöhe, auch negative Werte mögl.  */
  LONG lfWidth;                   /* Zeichenbreite                           */
  LONG lfEscapement;              /* Drehung                                 */
  LONG lfOrientation;             /* Drehung                                 */
  LONG lfWeight;                  /* Strichstärke, 1 - 1000, 400 = normal    */
  BYTE lfItalic;                  /* Kursiv                                  */
  BYTE lfUnderline;               /* Unterstrichen                           */
  BYTE lfStrikeOut;               /* Druchgestrichen                         */
  BYTE lfCharSet;                 /* Zeichensatz                             */
  BYTE lfOutPrecision;            /* Genauigkeit am Ausgabegerät             */
  BYTE lfClipPrecision;           /* Clipping-Genauigkeit                    */
  BYTE lfQuality;                 /* Darstellungsqualität, z.B. Antialiasing */
  BYTE lfPitchAndFamily;          /* Familie der Schriftart, z.B. Roman      */
  TCHAR lfFaceName[LF_FACESIZE];  /* Name der Schriftart                     */
} LOGFONT, *PLOGFONT;
```

Erzeugen einer Schriftart

lfHeight

- **Siehe nächste Seite**

lfFaceName

- **Name der gewünschten Schriftart**

Alle anderen Felder

- **Alle anderen Felder der Struktur lf können einfach auf 0 gesetzt werden, wodurch das System eine „vernünftige“ Wahl trifft.**

Erzeugen einer Schriftart

lfHeight

- **Gibt die Höhe der Schriftart (in logical units) an:**
  - > 0: Height of the font's character cell
  - < 0: Character Height (The character height value (also known as the em height) is the character cell height value minus the internal-leading value.)
- **Die übliche Zeichensatzgröße in Punkten bezieht sich auf 72 dpi Auflösung und muß auf die tatsächliche Bildschirmauflösung umgerechnet werden.**
- **GetDeviceCaps(hdc, LOGPIXELSY) liefert die vertikale Auflösung des Bildschirms in dpi.**

Erzeugen einer Schriftart

```
HFONT hfont, hfontOld;
LOGFONT lf;

memset(&lf, 0, sizeof(LOGFONT));
lf.lfHeight = -MulDiv(pointSize, GetDeviceCaps(hdc, LOGPIXELSY), 72);
strcpy(lf.lfFaceName, "Times New Roman");

hfont = CreateFontIndirect(&lf);
hfontOld = SelectObject(hdc, hfont);
...
TextOut(hdc, 10, 50, "Test", 4);
...
SelectObject(hdc, hfontOld);
DeleteObject(hfont);
```

Erzeugen einer Schriftart

MulDiv

- **int MulDiv(int nNumber,**
  int nNumerator,
  int nDenominator);
- **Die Funktion multipliziert zwei 32-Bit Werte und dividiert das 64-Bit Ergebnis durch einen dritten 32-Bit Wert. Das Ergebnis wird auf den nächsten ganzzahligen Wert auf- bzw. abgerundet.**
- **Bei Erfolg wird das Ergebnis der Multiplikation und Division zurückgegeben, sonst (bei Überlauf oder Divisor = 0) wird -1 zurückgegeben.**

Stock Objects

Stock Objects

- **„lagernde“ Objekte**
- **stellt das System immer zur Verfügung**
- **müssen nicht mit CreateXXX angelegt werden**

GetStockObject

- **HGDIOBJ GetStockObject(int objConst);**
- **In objConst wird übergeben, welches Zeichenobjekt benötigt wird. Der Rückgabewert ist ein Handle auf das gewünschte Objekt oder NULL wenn der Aufruf fehlgeschlagen ist.**

Stock Objects

Einige der möglichen Werte für objConst sind:

BLACK_BRUSH	schwarzer Pinsel
GRAY_BRUSH	grauer Pinsel
WHITE_BRUSH	weißer Pinsel
HOLLOW_BRUSH	durchsichtiger Pinsel
BLACK_PEN	schwarzer Stift
WHITE_PEN	weißer Stift
ANSI_FIXED_FONT	nicht proportionaler Systemzeichensatz
ANSI_VAR_FONT	proportionaler Systemzeichensatz
SYSTEM_FONT	Systemzeichensatz, z.B. für Menüs verwendet

Fokus-Rechteck

Bei WM_LBUTTONDOWN

- Mausereignisse „einfangen“: SetCapture()
- Fokus-Rechteck zeichnen: DrawFocusRect()

Bei WM_MOUSEMOVE

- Altes Fokus-Rechteck löschen: DrawFocusRect()
- Neues Fokus-Rechteck zeichnen: DrawFocusRect()

Bei WM_LBUTTONUP

- Altes Fokus-Rechteck löschen: DrawFocusRect()
- Mausereignisse „freigeben“: ReleaseCaptue()

Fokus-Rechteck

SetCapture

- **HWND SetCapture(HWND hWnd);**
- **Die Funktion veranlaßt, daß alle Mausereignisse an das angegebene Fenster gesendet werden, auch wenn die Maus aus dem Fenster hinausbewegt wird. Es kann nur ein Fenster die Mausereignisse einfangen.**
- **Bei Erfolg gibt die Funktion das Fenster zurück, das vorher alle Mausereignisse einfing. Bei einem Fehler wird NULL zurückgegeben.**
- **Wenn das Fenster nicht mehr alle Mausereignisse benötigt, muß die Funktion ReleaseCapture aufgerufen werden.**

Fokus-Rechteck

ReleaseCapture

- **BOOL ReleaseCapture(VOID);**
- **Die Funktion gibt die Mausereignisse wieder frei, sodaß danach wieder alle Fenster Mausereignisse zugesandt bekommen.**
- **Bei Erfolg wird ein Wert ungleich Null zurückgegeben.**

Arbeiten mit Bitmaps

Zeichenoperationen auf einer Bitmap im Speicher:

- **passender Gerätekontext für die Bitmap, einen sog. „Speichergerätekontext“ (memory device context)**
- **Bitmap, die in diesen Speichergerätekontext selektiert wird**

Zwischen Bitmap und Bildschirm können Ausschnitte in jede Richtung kopiert werden. Dazu müssen aber die internen Darstellungsformen beider Medien verträglich sein. Dazu verwendet man die Funktionen

- **CreateCompatibleDC**
- **CreateCompatibleBitmap**

Arbeiten mit Bitmaps

CreateCompatibleDC

- **HDC CreateCompatibleDC(HDC hdc);**
- **Erzeugt einen Speichergerätekontext, der zum in hdc übergebenen Gerätekontext kompatibel ist. Wenn hdc auf NULL gesetzt ist, wird ein zum Systembildschirm kompatibler Gerätekontext erzeugt. Der Rückgabewert ist ein Handle auf den neuen Speichergerätekontext oder NULL wenn der Aufruf fehlgeschlagen ist.**
- **Um auf einem Speichergerätekontext Zeichenoperationen durchführen zu können, muß eine Bitmap in den Kontext selektiert werden.**
- **Ein mit dieser Funktion erzeugter Gerätekontext muß nach Verwendung mit DeleteDC freigegeben werden.**

Arbeiten mit Bitmaps

DeleteDC

- **BOOL DeleteDC(HDC hdc);**
- **Ein mit Create...DC erzeugter Gerätekontext wird freigegeben. Zu diesem Zeitpunkt dürfen nur mehr Standard-Zeichenobjekte in den Gerätekontext selektiert sein. Der Rückgabewert ist ungleich 0 wenn der Aufruf erfolgreich war.**

Arbeiten mit Bitmaps

CreateCompatibleBitmap

- **HBITMAP CreateCompatibleBitmap(HDC hdc,**
  int width,
  int height);
- **Es wird eine zum in hdc übergebenen Gerätekontext kompatible Bitmap erzeugt. Die Breite der Bitmap wird durch width und die Höhe durch height definiert.**
- **Wird durch hdc ein Speichergerätekontext bezeichnet, hat die neue Bitmap die gleichen Eigenschaften wie die Bitmap, die gerade in hdc selektiert ist. Ein neu erzeugter Speichergerätekontext hat automatisch eine monochrome Bitmap (ein stock object) selektiert.**
- **Der Rückgabewert ist ein Handle auf die neue Bitmap oder NULL wenn der Aufruf fehlgeschlagen ist.**

Arbeiten mit Bitmaps

BitBlt

- **BOOL BitBlt(HDC dest,**
  int xDest, int yDest, int width, int height,
  HDC source, int xSrc, int ySrc,
  DWORD opCode);
- **Eine Bitmap wird von einem auf einen anderen Geräte-kontext kopiert. Kopiert wird von source nach dest. Die Größe des kopierten Rechtecks wird durch width und height bestimmt. Innerhalb des Zielgerätekontexts wird die linke obere Ecke der Bitmap mit xDest und yDest positioniert. Die linke obere Ecke innerhalb der Quelle wird durch xSrc und ySrc bestimmt. Durch opCode wird die Art der Rasteroperation bestimmt.**
- **Einige vordefinierten Konstanten für opCode sind:**
  - SRCCOPY exakte Kopie
  - SRCINVERT kombiniere Ziel und Quelle mit XOR

Arbeiten mit Bitmaps

Bei komplexen graphischen Fensterinhalten ist es oft nicht möglich / zielführend, die Ausgabe des gesamten Fensterinhalt mit Zeichenprimitiva wie Punkt, Linie, Buchstabe, … zu realisieren.

Mögliche Alternative: alle während des Programmablaufs am Bildschirm durchgeführten Zeichenoperationen werden parallel im Hintergrund auf einer Bitmap im Speicher durchgeführt. Ist ein Update des gesamten Fensters notwendig, wird diese Bitmap auf den Bildschirm kopiert - schnell und speicheraufwendig!

Es gibt auch eine etwas einfachere Alternative: alle Zeichenoperationen werden auf der Bitmap im Speicher durchgeführt und der Update des Bildschirms erfolgt immer über die Bitmap. Dadurch ergibt sich eine klarere Programmstruktur ohne Codeverdopplung (siehe Variante #2 der Bildschirmausgabe).

Arbeiten mit Bitmaps

```
#define MAXWIDTH …
#define MAXHEIGHT …
HBITMAP hbmpOld, hbmp; HDC hdc, hdcmem; RECT r;
…
/* -- Initialisierung der Bitmap -- */
hdcmem    = CreateCompatibleDC(NULL);
hbmp      = CreateCompatibleBitmap(hdc, MAXWIDTH, MAXHEIGHT);
hbmpOld  = SelectObject(hdcmem, hbmp);
r.left = 0; r.top = 0; r.right = MAXWIDTH; r.bottom = MAXHEIGHT;
FillRect(hdcmem, &r, GetStockObject(WHITE_BRUSH));
…
/* -- Kopieren auf den Bildschirm -- */
BitBlt(hdc, 0, 0, MAXWIDTH, MAXHEIGHT, hdcmem, 0, 0, SRCCOPY);
…
/* -- Aufräumen -- */
SelectObject(hdcmem, hbmpOld); DeleteDC(hdcmem);
```