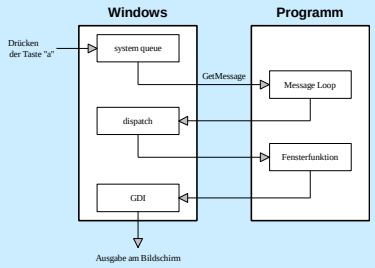
KV Betriebssysteme
Windowsprogrammierung

KV Betriebssysteme
Windowsprogrammierung
Hauptprogramm, Fenster und Nachrichten

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;
}
```

KV Betriebssysteme
Windowsprogrammierung

Initialisierung (3)

Erkennung der ersten Instanz
- Die erste Instanz einer Applikation erkennt man am Parameter hInstPrev. Er ist beim ersten Aufruf NULL.
- Das funktioniert NUR bei Win16-Programmen!
  - Bei Win32 Programmen definiert IMMER NULL

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 RegisterClassEx(CONST WNDCLASSEX *lpwcx);
- 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 WNDCLASSEX

```
typedef struct _WNDCLASSEX {
    UINT        cbSize;         /* Größe                    */
    UINT        style;          /* Klassenstile             */
    WNDPROC     lpfnWndProc;    /* Fensterfunktion          */
    int         cbClsExtra;     /* Extra Klassenspeicher    */
    int         cbWndExtra;     /* Extra Fensterspeicher    */
    HANDLE      hInstance;      /* Instanz                  */
    HICON       hIcon;          /* Icon, NULL -> kein I.    */
    HCURSOR     hCursor;        /* Mauscursor, NULL         */
    HBRUSH      hbrBackground;  /* Hintergrundfarbe         */
    LPCTSTR     lpszMenuName;   /* Menüname, NULL           */
    LPCTSTR     lpszClassName;  /* Klassenname              */
    HICON       hIconSm;        /* Kleines I., NULL->wie hIcon */
} WNDCLASSEX;
```

ACHTUNG: cbSize muß IMMER auf sizeof(WNDCLASSEX) gesetzt werden!

Globale Initialisierung - Beispiel

```
static char szWndClassName[] = "CHelloWorld";
static int InitApplication(HINSTANCE hInstCurr) {
    WNDCLASSEX wc;  /* window class */
    wc.cbSize           = sizeof(WNDCLASSEX);
    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;
    wc.hIconSm          = NULL;
    return RegisterClassEx(&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.

KV Betriebssysteme
Windowsprogrammierung

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/owner window
  HMENU       hMenu,           // menu handle or child identifier
  HANDLE      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)>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.**
- **-1 wird bei Fehlern zurückgeliefert (-> Programm abbrechen).**

Fensterfunktion

Schnittstelle

- **Jedes Fenster (eig. Fensterklasse) hat eine Fensterfunktion.**
- **Der Funktionsname kann frei gewählt werden.**
- **LRESULT CALLBACK WndProc(**
  HWND hwnd, /* handle of window */
  UINT uMsg, /* message identifier */
  WPARAM wParam, /* 1st message parameter */
  LPARAM lParam); /* 2nd message parameter */
- **Rückgabewert: Hängt von der Nachricht ab!**
  - (CALLBACK ist nötig, damit die Prozedur selbst als Parameter übergeben werden kann; „calling convention“ -> Parameter-Reihenfolge)

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.

KV Betriebssysteme
Windowsprogrammierung

Fensterfunktion (3)

```
LRESULT CALLBACK 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". Wird gesendet, nachdem das Fenster entfernt wurde (Childs existieren hier noch!).

WM_MOUSEMOVE:
- Die Maus wurde über das Fenster bewegt. Geht an das Fenster, das den Cursor besitzt (Außer: SetCapture(HWND)!).

WM_MOVE:
- Die Position des Fensters hat sich geändert.

WM_PAINT:
- Inhalt des Fensters soll neu gezeichnet werden.

WM_QUIT:
- Das Programm (=Applikation) soll beendet werden. Fenster müssen selbst zerstört worden sein!

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 (HDC)

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

KV Betriebssysteme
Windowsprogrammierung
Fenster-Lebenszyklus

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

KV Betriebssysteme
Windowsprogrammierung

Lebenszyklus eines Fenster

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

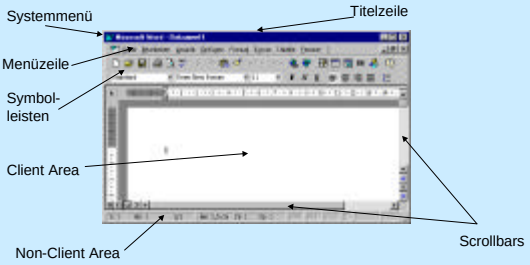
Nachricht WM_CREATE
- Ressourcen für das Fenster anlegen
- lParam: Zeiger auf CREATESTRUCT (enthält u. A. Start-Parameter)

Fenster anzeigen
- Stil WS_VISIBLE (CreateWindow), Funktion ShowWindow (Max., Rest., …)

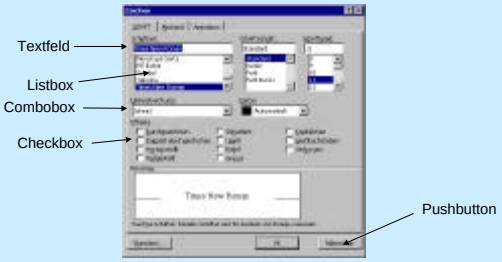
Funktion DestroyWindow
- Schließt das Fenster (inkl. Child-Windows)
- Sendet die Nachricht WM_DESTROY

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

Fensterelemente



Fensterelemente



Fensterstatus

enabled/disabled
- Ein Fenster, das enabled ("aktiv") ist, kann Eingabeereignisse (Maus + Tastatur) 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. Setzen mittels ShowWindow.

hidden/visible
- Ein Fenster muß nicht unbedingt sichtbar sein. Ohne besondere Vorkehrungen wird ein neues Fenster erst durch ShowWindow sichtbar!

Fensterelemente

Größe von Elementen
- Die Breite des Rahmens (verschiedene!), Höhe von Menüzeilen, etc. kann über GetSystemMetrics festgestellt werden.
- int GetSystemMetrics(int nIndex)
- nIndex bezeichnet des gewünschte Element:

SM_CMOUSEBUTTONS	Anzahl der Mausknöpfe
SM_CXFIXEDFRAME	Pixelbreite eines Fensterrahmens (Fenstergröße nicht veränderbar)
SM_CYFIXEDFRAME	Pixelhöhe Fensterrahmen
SM_CYCAPTION	Höhe der Fenster-Titelleiste
SM_CYMENU	Höhe einer einzeiligen Menüzeile

...

Fensterelemente

Fenstertitel
- Der Name eines Fensters (=Titelzeile) kann bei CreateWindow angegeben werden. Soll er nachträglich verändert werden, so steht hierfür die Funktion SetWindowText zur Verfügung.
- BOOL SetWindowText(HWND hWnd, LPCTSTR lpString) Das Fensterhandle und ein Null-terminierter String sind anzugeben. Bei Erfolg gibt die Funktion einen Wert ungleich 0 zurück.
- Zum Feststellen des Titels kann int GetWindowText(HWND hWnd,LPTSTR lpString,int nMaxCount) verwendet werden. Rückgabewert ist die Länge des gespeicherten Strings.

KV Betriebssysteme
Windowsprogrammierung

KV Betriebssysteme
Windowsprogrammierung
Zeichnen (GDI)

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

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 Metafile.**
- **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.**

Graphics Device Interface

Ein device context enthält:
•Pen (Stift für Linien)
•Brush (Füllmuster)
•Bitmap (Für kopieren und scrollen des Bildschirms)
•Palette (Verfügbare Farben)
•Font (Schriftart)
•Region (Für clipping und anderes)
•Path (Ein oder mehrere Zeichenobjekte)

Device Context

Ein Device Context definiert

- **das Ausgabemedium**
- **die Koordinateneinheiten (Pixel, Zoll,…)**
- **die gerade gewählten Zeichenobjekte**
- **die aktuelle Zeichenposition (ähnlich wie log. Cursor)**
- **die Graphik-Modi:**
  - Background: Zeichnen des Hintergrunds (Opaque oder Transparent)
  - Drawing: Mischung von Zeichenfarbe und Hintergrund
  - Mapping: Umsetzung von logischen auf physikalische Koordinaten
  - Polygon-fill: Füllen von überlappenden Polygonen
  - Stretching: Farbmischung bei Bitmap-Skalierung

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, ...).

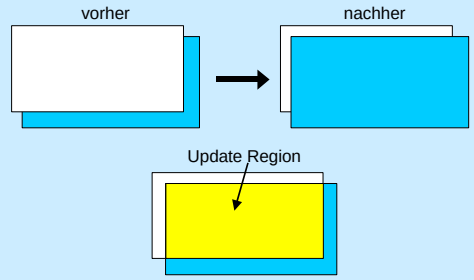
Mit den Funktionen InvalidateRect und InvalidateRgn können Teile eines Fensters für „ungültig“ erklärt werden und werden somit in die „Update Region“ aufgenommen.

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.

KV Betriebssysteme
Windowsprogrammierung

Update Region



Nachricht WM_PAINT

- **Aufruf der Funktionen BeginPaint**
  - Liefert einen Device Context.
  - Die Clipping Region des Device Context wird auf die Update Region des Fenster gesetzt.
  - Entfernt das Caret (=Cursor)
  - 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.
- **Jedes BeginPaint hat immer genau ein EndPaint!**

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 direkt im Code (Maus-Ereignisse, ...):**
  - Aufruf von GetDC()
  - Aufruf einer / mehrerer Zeichenfunktionen
  - ReleaseDC()
  - Speichern in einer Datenstruktur für WM_PAINT
- **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 direkt im Code (Maus-Ereignisse, ...):**
  - Zeichnen in die Bitmap und Aufruf von InvalidateRect()
- **Vorteile:**
  - Verhindert das Flackern (wenn keine Hintergrundfarbe verwendet wird).
  - Keine Codeverdopplung
- **Nachteile:**
  - Erhöhter Speicher- und Ressourcenbedarf (fast komplette Kopie des Bildschirms nötig bei maximierten Fenstern!)