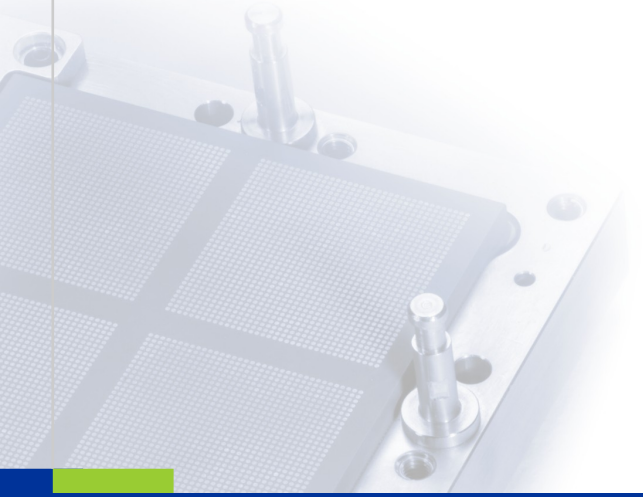
R|R|Z|N

Grafische Darstellung von Daten



Nutzung der Windows API

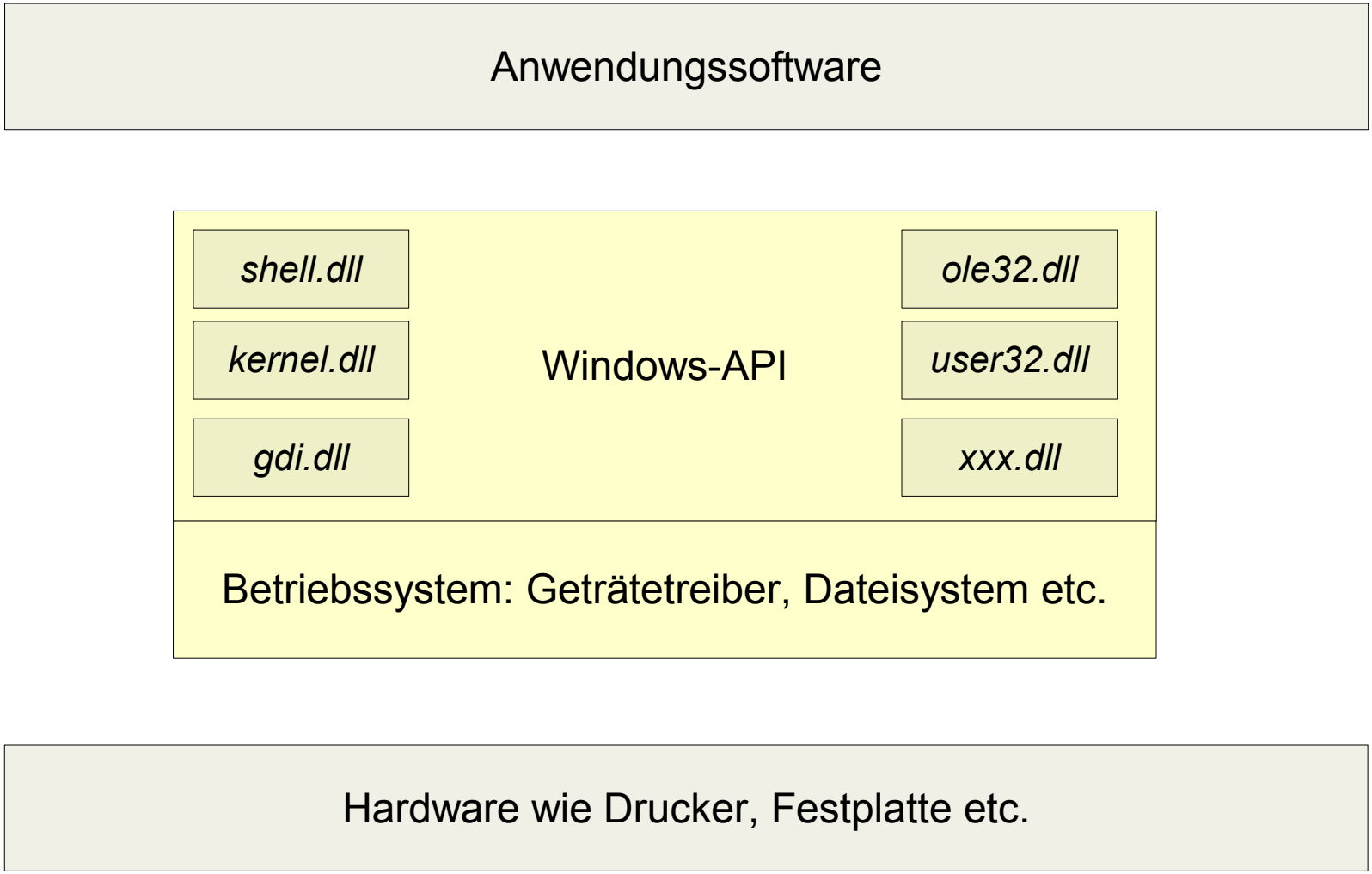
Informationen zur Windows API

R|R|Z|N

- /10/ Charles Petzold: Windows-Programmierung; Microsoft Press
- /11/ Peter Prinz u. Ulla Kirch-Prinz: C für PCs

- http://www.win-api.de/
- http://msdn2.microsoft.com/en-us/library/aa383750.aspx
- http://www.winprog.org/tutorial/
- http://runtime-basic.net/Windows-API:Windows-API

- Windows Application Programming Interface
- ... ist eine Schnittstelle zwischen einem Windows-Programm und dem Betriebssystem.
- ... stellt Funktionen zur Verfügung,
  - ... die die Erstellung von Dialogen, Menüs etc. vereinfachen.
  - ... Informationen zu dem Betriebssystem liefern.
- Die Funktionen werden zu verschiedenen Themen in Bibliotheken mit der Dateiendung ".dll" gesammelt.
  - Die Dateiendung ist eine Abkürzung für Dynamic Link Library.
  - Die Bibliotheken werden bei Bedarf in den Hauptspeicher geladen.
- Seit Windows 95 wird die Win32API genutzt.
- ... wird mit Hilfe von `#include <windows.h>` eingebunden.


Anwendungssoftware
shell.dll
kernel.dll
gdi.dll
Windows-API
ole32.dll
user32.dll
xxx.dll
Betriebssystem: Geträtetreiber, Dateisystem etc.
Hardware wie Drucker, Festplatte etc.

Beispiel

R|R|Z|N

```
#include <windows.h>
#include <stdio.h>
int main(){
          DWORD nLen;
          char computerName[255];
          nLen = sizeof(computerName);
          GetComputerName(computerName, &nLen);
          printf("Name des Computers: %s\n", computerName);

          char userName[255];
          nLen = sizeof(userName);
          GetUserName(computerName, &nLen);
          printf("Name des Users: %s\n", userName);

          char winPath[MAX_PATH];
          GetWindowsDirectory(winPath, sizeof(winPath));
          printf("Pfad zu Windows: %s\n", winPath);

          WinExec("notepad.exe", SW_NORMAL);
          return 0;
}
```

Beispiel

R|R|Z|N

```
#include <windows.h>
#include <stdio.h>
int main(){
        DWORD nLen;
       char computerName[255];
        nLen = sizeof(computerName);
        GetComputerName(computerName, &nLen);
        printf("Name des Computers: %s\n", computerName);

       char userName[255];
        nLen = sizeof(userName);
        GetUserName(computerName, &nLen);
        printf("Name des Users: %s\n", userNa
```

Um die Funktionen der Windows-API zu nutzen. muss `windows.h` eingebunden werden.

Es wird der Computername ausgegeben, auf dem das Programm läuft.

Es wird der Benutzername ausgegeben, der das Programm gestartet hat.

Beispiel

```
        ...
        char winPath[MAX_PATH];
        GetWindowsDirectory(winPath, sizeof(winPath));
        printf("Pfad zu Windows: %s\n", winPath);

        WinExec("notepad.exe", SW_NORMAL);
        return 0;
}
```

Der Pfad des Windows-Betriebssystems wird ausgegeben.

Hier wird ein weiteres Programm gestartet. Falls kein Systemprogramm gestartet wird, muss der Pfad angegeben werden.

Bestandteile eines Windows-Programms

- Die Header-Datei <*windows.h*> muss am Anfang des Programms eingebunden werden, um die Funktionen bekannt zumachen.
- Als Einstieg in das Programm wird die Hauptfunktion `WinMain()` benötigt. In dieser Funktionen
  - ... werden die Merkmale des Fensters festgelegt.
  - ... eine Struktur für ein Fenster registriert.
  - ... ein Fenster geöffnet.
- Jedes Fenster sendet Nachrichten und kommuniziert mit anderen Elementen. Die gesendeten Nachrichten werden in der Funktion `WndProc()` verarbeitet.

Datentypen

R | R | Z | N

Typ	Beschreibung	Präfix
BOOL	True (1, wahr); False (0, Falsch)	b; f
INT	int	i
UINT	unsigned int	
LONG; LRESULT	long	l
ULONG	unsigned long	
BYTE	unsigned char	by
WORD	unsigned short; Wort	w
DWORD	unsigned long; Doppelwort	dw
WPARAM	unsigned short (16 Bit)	
	unsigned int (32 Bit)	
LPARAM	long (16 und 32 Bit)	
LPSTR	char *	sz
LPCSTR	const char *	
HANDLE	void *	h
POINTER		p; lp

Beispiel

R|R|Z|N

```
LPCSTR lpszAppName = "myWindow";
LPCSTR lpszTitle = "Mein erstes Fenster";
```

Deklaration von Zeigern auf einen konstanten String in der WinAPI-Schreibweise.

```
const char *lpszAppName = "myWindow";
const char *lpszTitle = "Mein erstes Fenster";
```

Deklaration von Zeigern auf einen konstanten String in der C-Schreibweise.

Namenskonventionen

- Datentypen
  - ... werden immer groß geschrieben. Beispiel: WORD wAnzahl = 0;
- Symbolische Konstanten
  - ... werden groß geschrieben.
  - ... beginnen mit Buchstaben, die einen Hinweis auf die Art / Nutzung der Konstante gibt. Beispiel: WM_LBUTTONUP. WM verweist auf eine Windows Message.
- Variablennamen
  - ..., die aus mehreren Worten bestehen, beachten die ungarische Notation.
  - ... beginnen stets mit einem Präfix aus Kleinbuchstaben. Der Präfix liefert einen Hinweis auf den Datentyp der Variablen. Beispiel: char szTitle[100];

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine, int iCmdShow)
```

- ... entspricht der main-Funktion in der C-Programmierung.
- ... ist exakt einmal in einem Programm enthalten.
- ... erstellt ein Fenster.
- Parameter, die der Funktion übergeben werden:
  - Eine ID, die eine Programm eindeutig identifiziert. Wenn ein Anwendungsprogramm mehrmals gestartet wird, wird nicht der gesamte Programmcode nochmals geladen. Es wird nur eine Instanz mit einen eigenen Datenbereich und Nummer angelegt.
  - Der Parameter entspricht dem Parameter argc einer C-main-Funktion. Der Parameter ist ab Windows 95 immer 0.
  - Der Parameter entspricht dem Parameter argv einer C-main-Funktion. Es wird ein Zeiger auf ein String mit Argumenten von der Kommandozeile übergeben.
  - Darstellung des Fensters nach dem Öffnen.

... hat folgende Parameter

R|R|Z|N

- `HINSTANCE hInstance`
  - Eine Instanz ist ein Objekt zur Laufzeit. Bei einem mehrmaligen Start wird nicht der gesamte Code neu angelegt, sondern nur eine Instanz. Jede Instanz hat einen eigenen Datenbereich und eine eindeutige Nummer.
  - ... bezeichnet immer eine Instanz von Windows.
  - ... ist eine ID, die die Applikation eindeutig identifiziert.
  - ... wird von Windows zugewiesen.
  - ... wird bei der Registrierung der Struktur und der Erzeugung des Fensters genutzt.
- `HINSTANCE hPrevInstance`
  - ... bezeichnet die Eltern-Instanz.
  - ... ist seit Windows 95 null.
  - Der Parameter entspricht dem Parameter argc einer C-main-Funktion.

... hat folgende Parameter

- `LPSTR lpCmdLine`
  - Der Parameter entspricht dem Parameter argv einer C-main-Funktion.
  - Es wird ein Zeiger auf ein String mit Argumenten von der Kommandozeile übergeben.
- `iCmdShow`
  - Wie wird das Fenster dargestellt?
  - Welche Merkmale hat es standardmäßig?
  - ... wird beim Anzeigen eines Fensters genutzt.

Handle des Fensters

R|R|Z|N

```
HWND hWnd;
```

- ... ist ein Handle des Fensters.
- ... ist eine vorzeichenlose Integer-Zahl, die ein Fenster eindeutig identifiziert.

Beschreibung eines Windows-Fenster

- Position der linken oberen Ecke: x- und y-Position.
- Breite und Höhe des Fensters.
- Eine Titelleiste, die
  - ... ein Icon links oben enthält, welches das Programm symbolisiert. Mit einem Klick auf das Icon öffnet sich das Systemmenü.
  - ... eine Schaltfläche zum Minimieren und Maximieren enthält.
  - ... eine Beschreibung des Fensters enthält.
- Rahmen eines Fensters,
  - ... der nicht vorhanden oder fixiert ist, wenn das Fenster fest an einer Position steht und deren Größe nicht verändert werden kann.
  - ... der mit der Maus verändert werden kann.
- Hintergrundfarbe des Fensters.
- Mauszeiger.
- Eventuell Menüs.
- Reagiert auf Aktionen des Nutzers.

Generelle Darstellung eines Fensters

R|R|Z|N

```
WNDCLASSEX wc;
```

- Die Darstellung eines Fensters wird durch die Struktur WNDCLASSEX dargestellt.
- Bezeichner mit EX für "extended" bedeuten, dass hier eine Erweiterung einer früheren Definition vorgenommen wurde. Gegenüber der Struktur WNDCLASS kann
  - ... die Größe der Struktur vorgegeben werden und
  - ... ein kleines Icon eingebunden werden.
- ... ist seit dem Betriebssystem Windows 95 vorhanden.
- Die Größe der Struktur in Bytes wird mit Hilfe von `wc.cbSize = sizeof(WNDCLASSEX);` angegeben. Die Version der Struktur ist abhängig von der Größe der Struktur.

Beispiel

R|R|Z|N

```
wc.cbSize =  sizeof(WNDCLASSEX);
wc.style =  CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc =  WndProc;
wc.cbClsExtra =  0;
wc.cbWndExtra =  0;
wc.hInstance =  hInstance;
wc.hCursor =  LoadCursor(NULL,IDC_ARROW);
wc.hIcon =  LoadIcon(NULL, IDI_APPLICATION);
wc.hbrBackground =  (HBRUSH)GetStockObject(WHITE_BRUSH);
wc.lpszClassName =  lpszAppName;
wc.lpszMenuName  =  0;
wc.hIconSm =  LoadIcon(NULL, IDI_APPLICATION);
```

.cbSize	Größe der Struktur
.style	Verhaltensweise und Aussehen des Fensters
.lpfnWndProc	Nachrichtenschleife
.cbClsExtra	Zusätzlicher Speicher für die Struktur
.cbWndExtra	Zusätzlicher Speicher für das Fenster
.hInstance	Aktuelle Instanz der Applikation
.hCursor	Mauszeiger des Fensters
.hIcon	Icon in der Taskleiste zur Kennzeichnung
.hbrBackground	Hintergrundfarbe des Fensters
.lpszClassName	Ressource eines Fensters
.lpszMenuName	Ressource eines Menüs
.hIconSm	Icon links oben in der Titelleiste

Stil der Fensterklasse

R|R|Z|N

```
wc.style =  CS_HREDRAW | CS_VREDRAW;
```

- Mit Hilfe der Konstanten `CS_HREDRAW` und `CS_VREDRAW` wird das Fenster neu gezeichnet, sobald sich die Breite oder Höhe des Fensters verändert.
- ... legt das Verhalten des Fensters fest.
- Die Konstanten sind Symbole für Integer-Zahlen.
- Mehrere Konstanten werden mit Hilfe einer bitweisen ODER-Verknüpfung verbunden.
- Mit Hilfe der Konstanten `CS_NOCLOSE` wird die Schließen-Schaltfläche in der Titelleiste des Fensters ausgeblendet.

Handle übergeben

R|R|Z|N

```
wc.hInstance =  hInstance;
```

- Handle auf die aktuelle Instanz des Fensters.
- ... wird als Parameter an die WinMain()-Funktion übergeben.

Icon und Mauszeiger

```
wc.hCursor = LoadCursor(NULL,IDC_ARROW);
wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
wc.hIconSm =  LoadIcon(NULL, IDI_APPLICATION);
```

- `hIconSm` legt das Icon am linken Rand der Titelleiste für alle Fenster, die auf dieser Struktur beruhen, fest. `hIcon` ist ein größeres Icon, welches in dem Dialogfenster zum Wechsel der Task (<ALT>-<TAB>) angezeigt wird.
- `hCursor` legt das Aussehen des Mauszeigers fest. Hier wird der Mauszeiger innerhalb des Fenster als kleiner Pfeil dargestellt.

Load...

- Mit Hilfe der Funktion `Load...()` wird ein Bild für ein Mauszeiger oder Icon geladen.
- Folgende Parameter werden übergeben.
  - Der erste Parameter ist ein Instanzen-Handle auf eine exe- oder dll-Datei, in der das Bild enthalten ist. NULL wird für vom System vordefinierte Bilder genutzt.
  - Als zweiter Parameter wird der Name des Bildes übergeben. Vordefinierte Bilder werden durch symbolische Konstanten angesprochen.

Mögliche Icons

- IDI_APPLICATION
- IDI_ERROR
- IDI_INFORMATION
- IDI_QUESTION
- IDI_WARNING
- IDI_WINLOGO

Mögliche Mauszeiger

- `IDC_APPSTARTING` Kleiner Pfeil und Stundenglass.
- `IDC_ARROW`
- `IDC_CROSS`
- `IDC_HAND`
- `IDC_HELP`
- `IDC_WAIT`
- Weitere finden Sie in der Hilfe.

Hintergrundfarbe

```
wc.hbrBackground =  (HBRUSH)GetStockObject(WHITE_BRUSH);
```

- Welche Farbe wird für den Hintergrund genutzt?
- Mit einem Pinsel (`HBRUSH`) wird die Farbe aufgetragen (`GetStockObject()`).
- Hier wird die Farbe weiß (`WHITE_BRUSH`) als Hintergrundfarbe genutzt.

Hinweis auf die Fensterstruktur und das Menü


```
wc.lpszClassName =  lpszAppName;
```

- ... identifiziert die Fensterstruktur.
- Diese Variable muss gesetzt werden.
- Der Name kann als globale Variable definiert werden.
- In `CreateWindow()` wird dieser Name genutzt, um einem Fenster eine Struktur zuzuordnen.

```
wc.lpszMenuName  =  0;
```

- ... gibt einen Hinweis auf die Ressource eines Menüs.
- ... enthält in Regel ein Zeiger auf ein Standardmenü.
- Falls nicht das Standardmenü genutzt werden soll und kein benutzerdefiniertes Menü vorhanden ist, wird ein Wert null übergeben.

Fensterstruktur registrieren

R|R|Z|N

```
if(RegisterClassEx(&wc) == NULL){
                return 0;
}
```

- Das erstellte Fenster wird an dem Fenster-Manager von Windows angemeldet.
- Die Fensterstruktur wird registriert und damit deren Adresse spezifiziert.
- Falls ein Fehler auftritt, wird ein NULL-Zeiger zurückgeliefert.

Fenster erzeugen

R|R|Z|N

```
hWnd = CreateWindowEx(
       WS_EX_CLIENTEDGE,
       lpszAppName, lpszTitle,
       WS_OVERLAPPEDWINDOW | WS_VISIBLE,
       200, 300,
       CW_USEDEFAULT, CW_USEDEFAULT,
       0, 0,
       hInstance,
       0
)
```

- `CreateWindowEx()` legt die individuellen Eigenschaften eines Fensters fest.
- Die Funktion gibt ein Handle auf das erzeugte Fenster zurück. Falls ein Fehler erzeugt wurde, wird NULL zurückgeliefert.

Parameter von CreateWindowEx


```
hWnd = CreateWindowEx(
       DWORD dwExStyle,      /* Erweiterter Fensterstil */
       LPCSTR lpClassName,   /* = wc.lpszClassName */
       LPCSTR lpWindowName, /* Fenstertitel */
       DWORD dwStyle,        /* Fensterstil */
       int x,                /* x-Position  */
       int y,                /* y-Position */
       int nWidth,           /* Breite des Fensters */
       int nHeight,          /* Höhe des Fensters */
       HWND hWndParent,      /* Übergeordnetes Fenster */
       hMENU hMenu,          /* Menü */
       hINSTANCE hInstance, /* Programm - ID */
       LPVOID lpParam        /* zusätzlicher Parameter */
)
```

Fensterstile

- `WS_OVERLAPPEDWINDOW` besteht aus einer Kombination von
  - `WS_OVERLAPPED` überlappend
  - `WS_CAPTION` Titelleiste
  - `WS_SYSMENU` Systemmenü
  - `WS_THICKFRAME` Fenster kann in der Größe geändert werden
  - `WS_MINIMIZEBOX` Verkleinerungssymbol
  - `WS_MAXIMIZEBOX` Vergrößerungssymbol
- `WS_VISIBLE`
- `WS_DISABLED`
- Weitere Stile finden Sie in der Hilfe.

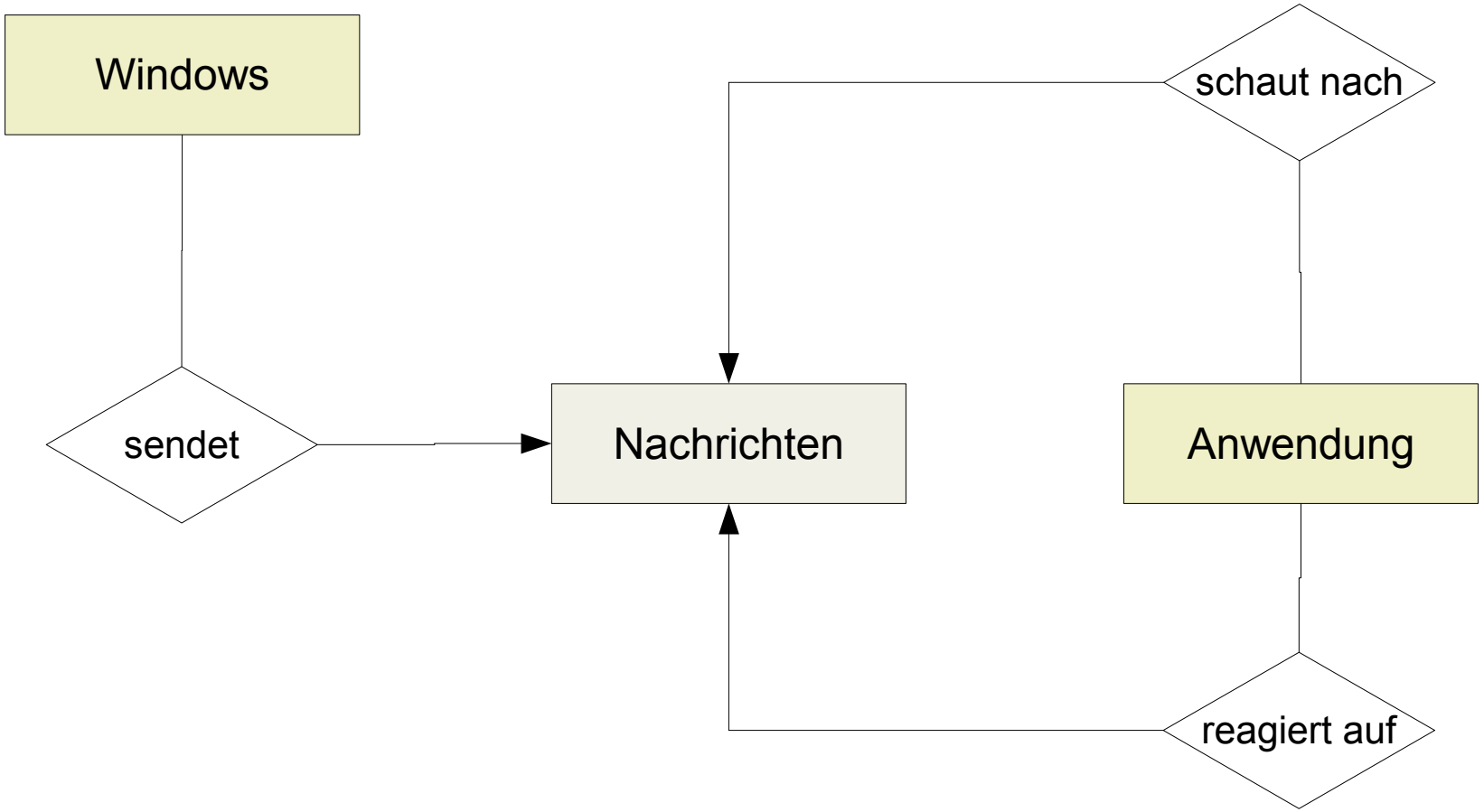
Fenster anzeigen

R|R|Z|N

```
ShowWindow(hWnd, iCmdShow);
UpdateWindow (hWnd);
```

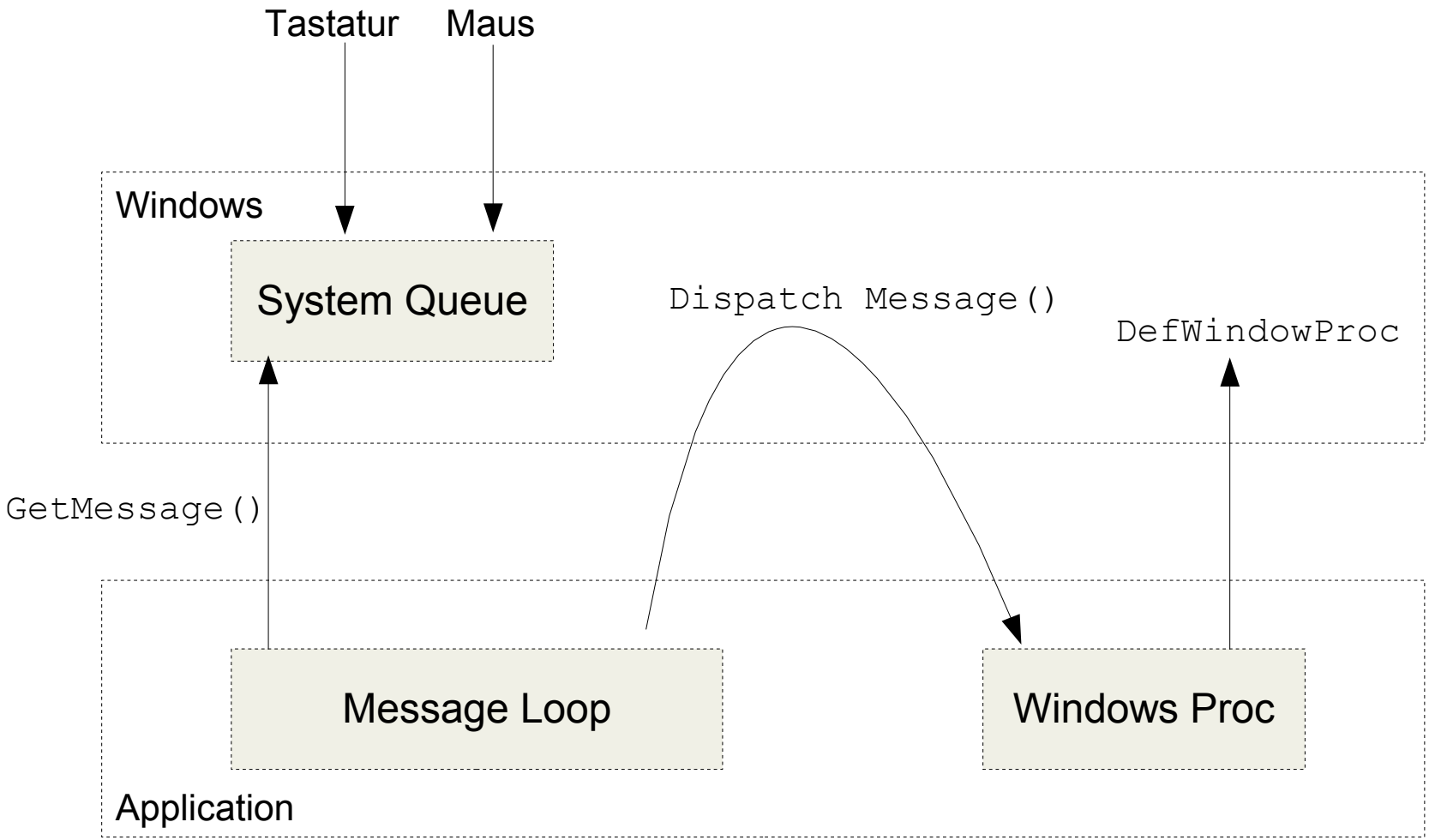
- Mit Hilfe von `ShowWindow()` wird der Anzeigestatus des Fensters gesetzt. Der Funktionen werden folgende Parameter übergeben:
  - Das Fenster-Handle wird übergeben.
  - Wie wird das Fenster angezeigt?
- `UpdateWindow()` schickt eine Nachricht an das Betriebssystem. Anschließend wird das Fenster auf dem Bildschirm gezeichnet.

Nachrichten verarbeiten



- An ein Fenster sind Funktionen gekoppelt, die ausgelöst werden, wenn der Anwender zum Beispiel in das Fenster klickt oder ein Fenster in der Größe ändert. Ein Mausklick etc. wird als Ereignis bezeichnet.
- Jedes dieser Ereignisse wird vom Betriebssystem als Meldung in den internen Puffer, die System-Queue abgelegt.
  - Eine Queue oder Warteschlange ist ein Speicherbereich, in dem Daten nach dem FIFO-Prinzip verwaltet werden. FIFO steht für "First in, First out". Das zuerst eingefügte Datenelement wird als erstes wieder entfernt.
- Windows wertet die Ereignisse aus und sendet die Meldung an die zuständige Anwendung.
- Jede Anwendung besitzt einen eigenen Puffer (Message Queue), in der Nachrichten abgelegt werden.
- In einer Meldungsschleife überprüft die Anwendung, ob Meldungen vorhanden sind. Wenn Meldungen vorhanden sind, wird die entsprechende Funktion ausgelöst. Die Anwendung reagiert auf die Meldung.

Realisierung in einem Windows-Programm



Struktur der Nachrichten

- Die Struktur besteht aus folgenden Strukturelementen:
  - Das Fenster-Handle zum Identifizieren des Fensters.
  - Die ID der Nachricht.
  - Zwei Parameter für besondere Angaben wie zum Beispiel Maus-Koordinaten, Tastaturstatus etc.
  - Zeitspanne zwischen dem Absenden von Windows und Ankunft der Nachricht beim Empfänger in Millisekunden.
  - x- und y-Koordinaten des Mauszeigers zum Zeitpunkt des Absenden der Nachricht.

```
typedef struct MSG {
        HWND hwnd;
        UINT message;
        WPARAM wParam;
        LPARAM lParam;
        DWORD time;
        POINT pt;
}
```

Message Loop

```
MSG msg;

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

- `GetMessage()` holt Nachrichten aus der Queue ab. Die Schleife wird durch das Senden von `WM_QUIT` abgebrochen.
- Wenn die Tastatur gedrückt wurde, wird mit Hilfe von `TranslateMessage()` ein virtueller Tastencode an die Message Loop weitergegeben. Dieser Tastencode wird in ASCII-Zeichen umgesetzt.
- `DispatchMessage()` verteilt die Nachrichten an die entsprechenden Windows-Prozeduren.

Nachrichten empfangen

R|R|Z|N

```
MSG msg;

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

- `GetMessage()` holt Nachrichten aus der Queue ab. Die Schleife wird durch das Senden von `WM_QUIT` abgebrochen.
  - Der erste Parameter ist ein Zeiger auf eine Nachricht.
  - Der zweiter Parameter bezieht sich auf das Fenster-Handle. Durch die Angabe von NULL werden alle Nachrichten vom Programm empfangen.
  - Die letzten zwei Parameter geben die minimale und maximale Anzahl von Nachrichten an. Durch die Angabe von 0 werden alle Nachrichten empfangen.

Nachrichten verarbeiten

R|R|Z|N

```
wc.lpfnWndProc =  WndProc;

LRESULT CALLBACK WndProc(HWND hWnd, UINT umsg,
                         WPARAM wParam, LPARAM lParam)
```

- Das Strukturelement `lpfnWndProc` des Fensters bekommt den Namen einer Funktion übergeben. Diese Funktion verarbeitet die, für die Anwendung, relevanten Nachrichten.
- Die Funktion `WndProc()` ist an das Fenster gebunden und nicht an das Programm.
  - Sie reagiert auf die gemeldeten Ereignisse.
  - Diese Funktion arbeitet alle Ereignisse ab, die in dem dazugehörigen Fenster passieren.

Parameter der Funktion

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT umsg,
                         WPARAM wParam, LPARAM lParam)
```

- Der Rückgabewert ist LRESULT.
- CALLBACK sagt aus, dass die Funktion von Windows aufgerufen wird.
- Folgende Parameter werden der Funktion übergeben.
  - Das Fenster-Handle zur Identifzierung des Fensters.
  - Die Nachricht, die verarbeitet werden soll.
  - Die letzten beiden Parameter sind von der Art der Nachricht abhängig.

Unbehandelte Nachrichten

R|R|Z|N

```
return DefWindowProc(hWnd, umsg, wParam, lParam);
```

- Es müssen nicht alle empfangenen Nachrichten behandelt werden.
- Die unbehandelten Nachrichten werden an Windows zurückgesandt.

Fenster schließen

R|R|Z|N

```
case WM_DESTROY:
{
    PostQuitMessage(0);
    return 0;
}
```

- Jede Nachricht beginnt mit WM (windows message).
- Mit Hilfe der Funktion `PostQuitMessage()` wird `WM_QUIT` an die Nachrichten-Schleife gesendet.
  - Die Nachrichtenschleife wird beendet.
  - Das Fenster sowohl als auch das Programm wird beendet.
  - Öffnen Sie den Task-Manager. Kommentieren diese Zeile aus und schließen das Fenster. Wird das Programm automatisch beendet?

Weitere Ereignisse

R|R|Z|N

Nachricht	Bedeutung	WPARAM	LPARAM
WM_DESTROY	Fenster schließen		
WM_CREATE	Fenster erstellen		Fenster
WM_SIZE	Fenster vergrößern	Art	Größe
WM_MOVE	Fenster verschieben		Position
WM_PAINT	Fenster neu zeichnen		
WM_MOUSEMOVE	Maus bewegen	Taste	Position
WM_LBUTTONDOWN	Maus gedrückt	Taste	Position
WM_RBUTTONDOWN			
WM_LBUTTONUP	Maus losgelassen	Taste	Position
WM_RBUTTONUP			
WM_KEYDOWN	Taste gedrückt	Taste	Dauer
WM_KEYUP	Taste losgelassen	Taste	Dauer

Beispiel: Fenstergröße verändern

```
static int cxClient;
static int cyClient;

case WM_SIZE:
        cxClient = LOWORD (lParam);
        cyClient = HIWORD (lParam);
        return 0;
```

- Die Nachricht `WM_SIZE` nutzt
  - ... `lParam`, um die Breite und Höhe des Fensters in Pixel an das Fenster zu übergeben.
  - ... `wParam`, um die Art der Veränderung mitzuteilen.
- `LOWORD` und `HIWORD` zerlegen ein 32-Bit-Wert in zwei 16-Bit-Werte.

Beispiel: Fenster neu zeichnen


```
PAINTSTRUCT ps;
HDC hDC;

case WM_PAINT:
        hDC = BeginPaint(hWnd, &ps);

        EndPaint(hWnd, &ps);
        return 0;
```

- `BeginPaint()` löscht die Client-Area des Fensters. Der Hintergrund des Fensters wird neu gezeichnet.
  - Der Funktion wird ein Fenster-Handle und einen Zeiger auf eine Struktur `PAINTSTRUCT`.
  - Es wird ein Gerätekontext auf ein Ausgabegerät zurückgeliefert.
- `EndPaint()` gibt den Gerätekontext frei.

Gerätekontext

- ... beschreibt immer eine Verbindung zu einem Ausgabegerät. Ein Ausgabegerät kann zum Beispiel der Bildschirm sein.
- ... wird benötigt, wenn mal irgendetwas gezeichnet werden soll.
- Der Gerätekontext enthält zum Beispiel Informationen zur Schriftart, dem Koordinatensystem, der Größe der Malfläche etc.

```
typedef struct PAINTSTRUCT {
        HDC hdc;
        BOOL fErase;
        RECT rcPaint;
        BOOL fRestore;
        BOOL fIncUpdate;
        BYTE rgbReserved[32];
}
```

- `recPaint` beschreibt ein Fenster mit Hilfe der Elemente `left`, `top`, `right` und `bottom`. Die Angaben erfolgen in Pixel und relativ zur linken, oberen Ecke des Anwendungsbereichs.

Beispiel: Linie zeichnen

```
case WM_PAINT:
        hDC = BeginPaint(hWnd, &ps);
        MoveToEx(hDC, 0, cyClient / 2, NULL);
        LineTo(hDC,  cxClient, cyClient / 2);
        EndPaint(hWnd, &ps);
        return 0;
```

- `MoveToEx()` setzt ein Startpunkt für das Zeichnen einer Linie.
  - Als erster Parameter wird der Gerätekontext übergeben. Wo soll gemalt werden?
  - Dann werden die x- und y-Koordinaten für den Startpunkt übergeben.
  - Der letzte Parameter kann einen Zeiger auf die aktuelle Position enthalten. Mit Hilfe der Funktion `GetCurrentPositionEx(hdc, &pt)` kann diese ermittelt werden.
- `LineTo()` zeichnet eine Linie von der aktuellen Position zum Endpunkt. Die momentane Position wird auf den Endpunkt umgesetzt.

Beispiel: Polygon zeichnen


```
POINT punkt[MAXNUM];

case WM_PAINT:
  hDC = BeginPaint(hWnd, &ps);
  for (count = 0; count < MAXNUM; count++){
       punkt[count].x = count * cxClient / MAXNUM;
       punkt[count].y = (int) (cyClient / 2 *
                          (1 - sin((2 * PI) * count  / MAXNUM)));
  }

  Polyline(hDC, punkt, MAXNUM);

  EndPaint(hWnd, &ps);
  return 0;
```

Die Variable von der Struktur POINT wird mit errechneten x- und y-Koordinaten gefüllt.

Die, in dem Array abgelegten Koordinaten werden mit Hilfe von Linien verbunden. Der letzte Parameter legt die Anzahl der zu verbindenden Koordinaten fest.

Beispiel: Zeichenstift initialisieren

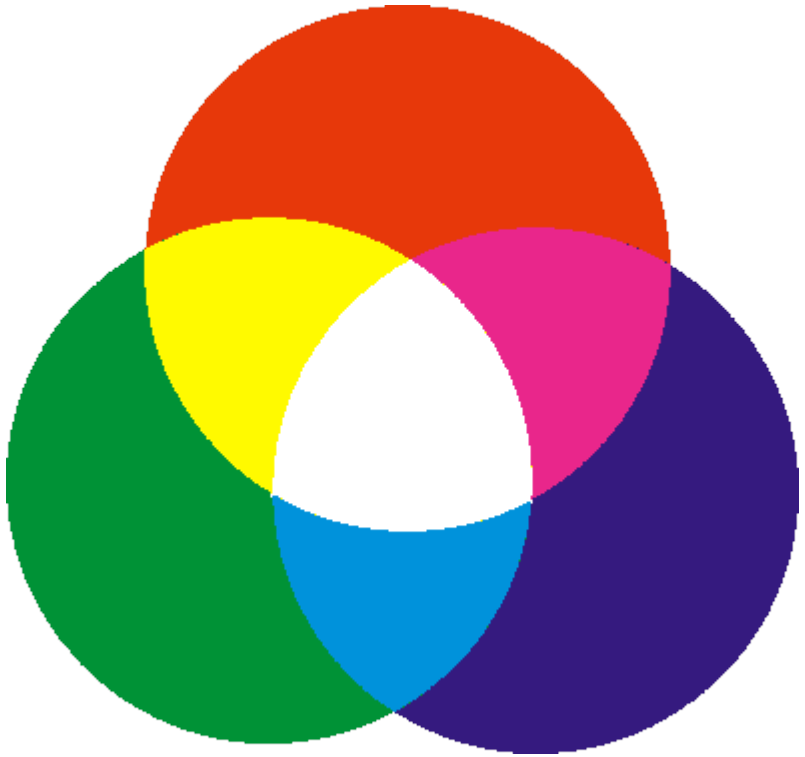

```
static HPEN myRot;
static HPEN mySchwarz;

case WM_CREATE:
      myRot = CreatePen(PS_SOLID, 2, RGB(255, 0, 0 ));
      mySchwarz = CreatePen(PS_SOLID, 1, RGB(0, 0, 0 ));
      return 0;
```

- Mit Hilfe von `HPEN` wird eine Variable für einen Zeichenstift erzeugt.
- `CreatePen()` erzeugt einen Zeichenstift.
  - Zuerst wird in die Art der Zeichenlinie übergeben. Möglichkeiten: `PS_SOLID`, `PS_DASH`, `PS_DOT` etc.
  - Dann wird die Strichbreite festgelegt.
  - Als letzter Parameter wird die Farbe übergeben. Mit Hilfe der Funktion `R(ed)G(reen)B(lue)()` wird eine Farbe gemischt.

Farbgestaltung

- Zur Darstellung von Farben auf dem Bildschirm wird das RGB-Farbsystem genutzt.
- Das RGB-Farbsystem addiert (mischt) Licht in verschiedenen Farben.
- Jede Farbe (Rot, Grün, Blau) wird in 256 Helligkeitsstufen unterteilt. Zum Beispiel:
  - `RGB(255, 255, 255)` stellt die Farbe weiß dar.
  - `RGB(0, 0, 0)` stellt die Farbe schwarz dar.
  - `RGB(255, 255, 0)` stellt die Farbe gelb dar.
- Um so mehr sich eine Farbe Weiß annährt, um so heller wird sie.



Beispiel: Zeichenstift nutzen


```
case WM_PAINT:
     hDC = BeginPaint(hWnd, &ps);
     SelectObject(hDC, mySchwarz);

case WM_DESTROY:
      DeleteObject(mySchwarz);
```

- Mit Hilfe von `SelectObject()` wird ein selbstdefiniertes Objekt ausgewählt. Der Funktion wird der Gerätekontext übergeben sowie das gewünschte Objekt. Anschließend wird mit Hilfe dieses Zeichenstifts gezeichnet.
- Mit Hilfe von `DeleteObject()` wird das nicht mehr benötigte Objekt aus dem Speicher entfernt.