AUF DER HEFT-CD

Demo-Programmierung unter Windows 95/98/NT

Jenseits aller Standards

Daß **Fenster nicht immer eckig** sein müssen, zeigte bereits der Künstler Hundertwasser an seinen Gebäuden in Wien. Machen Sie es ihm nach.

CARSTEN DACHSBACHER/
OLIVER KÄFERSTEIN

Fenster gehören zu den wichtigsten Elementen von Windows – schließlich ist das Betriebssystem nach ihnen benannt. Die Fenster sind nach bestimmten Regeln aufgebaut. Doch Standards sind auf die Dauer langweilig, erschaffen Sie daher Fenster in neuen Formen und Farben.

Zunächst lernen Sie beliebig geformte Fenster kennen, deren Form Sie durch eine Bitmap bestimmen. Außerdem zeigen wir Ihnen, wie Sie auf dem Desktop – ohne ein Fenster zu sehen – eine Grafik einblenden. Diese sogenannten Splash-Screens sind sehr beliebt, um beim Start eines Programms die Copyright-Meldungen anzuzeigen.

Bevor Sie eigene Fenster erschaffen, werfen Sie einen Blick in die entsprechenden Beispielprogramme Ihres Compilers. Experimentieren Sie an diesen Programmen herum.

Um ein Standardfenster zu erzeugen, registrieren Sie zunächst eine eigene Fensterklasse. Dazu legen Sie eine *WNDCLASS*-Struktur an. Diese enthält die Parameter für die Hintergrundfarbe, das zugehörige Icon und andere Eigenschaften des Fensters:

```
WNDCLASS wc;

wc.style = CS_BYTEALIGNCLIENT;
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = instance;
wc.hIcon = 0;
wc.hCursor = 0;

//Hintergrundfarbe
wc.hbrBackground = (HBRUSH)
  GetStockObject(BLACK_BRUSH);
wc.lpszMenuName = 0;
// Fensterklassenname
wc.lpszClassName =
  „Testfenster“;

//Messagehandler
wc.lpfnWndProc = WindowProc;

//Fensterklasse registrieren
RegisterClass(&wc);
```

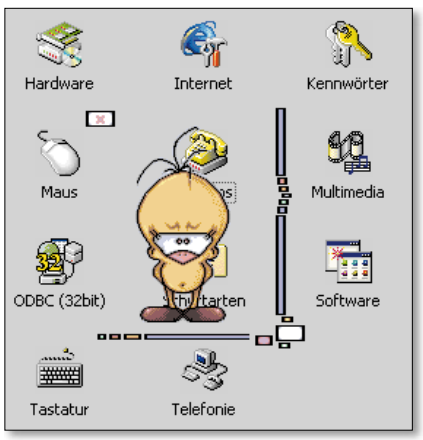
Bevor Sie in der letzten Listingzeile ein Fenster dieser Klasse erzeugen, weisen Sie ihm einen Messagehandler zu. Dieser Messagehandler arbeitet Nachrichten ab, die Windows an ihn schickt. Nachrichten sind Mausklicks auf das Fenster oder ein Tastendruck, sofern das Fenster aktiv ist. Alle Nachrichten, die Sie interessieren, geben Sie an den Standard-Messagehandler von Windows weiter.

Ein selbstdefinierter Messagehandler könnte dabei wie folgt aussehen:

```
long CALLBACK WindowProc(
  HWND hWnd, UINT message,
  WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
    case WM_CREATE:
      break;
    case WM_KEYDOWN:
      keycode = wParam & 255;
      return 0;
    case WM_DESTROY:
      break;
  }

  //alle uninteressanten Nach-
  //richten an Default-
  //Messagehandler weiterleiten
  return DefWindowProc(hWnd,
    message, wParam, lParam);
}
```

An dieser Stelle haben Sie alle Definitionen abgeschlossen, die das Aussehen und Verhalten Ihres Fensters betreffen.



UNTER WINDOWS DEFINIEREN Sie eigene Fenster aus mehreren Regionen.

Sie können nun ein Fenster dieser Klasse anlegen. Dazu benutzen Sie die Funktion *CreateWindowEx(...)*. In den übergebenen Parametern legen Sie unter anderem fest, ob Ihr Fenster *Minimize*- und *Maximize*-Knöpfe besitzt und wie groß es sein soll. Als Rückgabewert liefert *CreateWindowEx(...)* ein Handle vom Typ *HWND*, das das Fenster eindeutig identifiziert. Wichtig ist dabei, daß Sie das Handle der Application Instance – also das Handle, welches Ihr Programm identifiziert – mit übergeben:

```
HWND Window;
//Fenster an Position 100/120
//mit der Größe 320/240
Window = CreateWindowEx(
  WS_EX_TOPMOST, „Testfenster“,
  „Fenstertitel“, WS_POPUP,
  100, 120, 320, 240, 0, 0,
  instance, 0);
//Fehler:
if (!Window) return 0;
```

Jetzt brauchen Sie Windows nur noch mitzuteilen, daß Sie das Fenster sehen wollen. Dazu steht Ihnen der Befehl *ShowWindow(Window, ...)* zur Verfügung.

Um das Fenster mit einem Inhalt zu füllen, verwenden Sie den sogenannten *Device Context*. Damit können Sie zum Beispiel in das Fenster zeichnen oder Bitmaps darin anzeigen. Der *Device Context* ist eine Datenstruktur mit allen Informationen, die Windows braucht, um auf ein *Device* zu schreiben. Ein *Device* muß nicht unbedingt ein Fenster oder ein Bildschirm sein, es kann sich dabei auch um einen Drucker handeln. Den *Device Context* erhalten Sie über

```
HDC DeviceContext;
DeviceContext = GetDC(Window);
```

Möchten Sie beispielsweise eine Bitmap aus dem Hauptspeicher in das Fenster zeichnen, legen Sie eine *BITMAPINFO*-Struktur an. In dieser beschreiben Sie den Aufbau der Bitmap im Speicher:

```
int bisize = sizeof(
  BITMAPINFOHEADER);
bitmapinfo = (BITMAPINFO *)
  malloc(bisize+12);
ZeroMemory(&bitmapinfo->
```

```
  bmiHeader, bisize);

//BitmapInfoHeader erzeugen
bitmapinfo->bmiHeader.biSize
  = bisize;
bitmapinfo->bmiHeader.biWidth
  = SCREEN_X;
bitmapinfo->bmiHeader.biHeight
  = -SCREEN_Y;
bitmapinfo->bmiHeader.biPlanes
  = 1;
bitmapinfo->
  bmiHeader.biBitCount = 16;
bitmapinfo->
  bmiHeader.biCompression =
  BI_BITFIELDS;
//Farbfelder des Bitmaps setzen
*((long*) bitmapinfo->
  bmiColors +0) = 0xFF0000;
*((long*) bitmapinfo->
  bmiColors +1) = 0x00FF00;
*((long*) bitmapinfo->
  bmiColors +2) = 0x0000FF;
```

Dank dieser Struktur weiß Windows, wie die Grafikdaten im Speicher vorliegen. Sie können Windows veranlassen, die Bitmap in das Fenster zu zeichnen:

```
SetDIBitsToDevice(
  DeviceContext, 0, 0, Breite,
  Höhe, 0, 0, 0, Höhe,
  BitmapPtr, bitmapinfo,
  DIB_RGB_COLORS);
```

Sind Fenster und Bitmap unterschiedlich groß, übernimmt Windows auf Wunsch auch die Skalierung des Bildes. In diesem Fall verwenden Sie folgende Syntax:

```
//Größe des Fensters holen
GetClientRect(Window, &r);
StretchDIBits(DeviceContext,
  0, 0, r.right, r.bottom,
  0, 0, Breite, Höhe,
  BitmapPtr, bitmapinfo,
  DIB_RGB_COLORS, SRCCOPY);
```

Windows stellt Ihnen für die Behandlung von Bitmaps und *Device Contexts* noch weitere mächtige Befehle zur Verfügung. Diese sind in der Online-Hilfe Ihres Compilers erklärt.

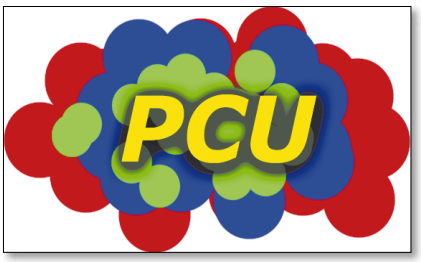
Fenster in allen Formen

Die Standardfenster in Windows sind rechteckig. Doch gerade Linien wirken steif und fördern nicht gerade die Kreativität. Diese Weisheit verdanken wir nicht zuletzt Stardesigner Colani und seinem legendären Biodesign. Nutzen Sie die Option von Windows, Fenster in einer beliebigen Form – sogar in nicht zusammenhängenden Teilen – anzuzeigen.

Definieren Sie sogenannte Regionen, mit denen Sie die sichtbaren Bereiche eines Fensters festlegen. Diese Regionen setzen Sie aus beliebigen Rechtecken zusammen. Da ein solches Rechteck auch nur ein Pixel groß sein kann, können Sie so jede beliebige Form erzeugen.

Die gewünschte Form legen Sie am besten durch eine Bitmap fest, die Sie in einem Zeichenprogramm Ihrer Wahl anfertigen. Für die spätere Anzeige des Fensters zerlegen Sie die undurchsichtigen Stellen der Bitmap in einige wenige Rechtecke – wenige deshalb, weil bei zu vielen Rechtecken die Performance leiden kann.

Die Funktionen zum Laden und Behandeln der Bitmaps finden Sie in der



DIESE BITMAP DIENT als Vorlage für den Splash-Screen.

pcPicture-Klasse im Quelltext zu dieser Ausgabe. Sie können damit ein Bild laden und mit 32 Bit Farbtiefe im Speicher ablegen:

```
pcWndRegion _pic;

//BMP laden
_pic.load(szPathName);

//32-Bit-Kopie anlegen
pcPicture bmReg(_pic.width(),
  _pic.height(), 32, TRUE );

//und BMP dorthin kopieren
_pic.blitTO(bmReg.hdc());

// Transparenzfarbe
COLORREF cTransCol =
  GetPixel(bmReg.hdc(), 0, 0);
```

Um die Bitmap in eine Region zu konvertieren, durchsuchen Sie es Zeile für Zeile. Jeweils zusammenhängende, nicht-transparente Pixel in einer Bitmap-Zeile ergeben ein Rechteck. Das ist kein optimaler, aber ein ausreichender Algorithmus, um die Bitmap in eine Region zu konvertieren. Als transparent gelten Pixel in einer vorgegebenen Farbe. Das Programm verwendet als Transparenzfarbe jeweils die Farbe des linken oberen Pixels der Bitmap:

```
for (y = 0; y < height; y ++)
{
  for (x = 0; x < width; x ++)
  {
    //suche zusammenhängendes
    //Gebiet von nicht trans-
    //parenten Pixeln
    dword x0 = x;
    dword *p = BitmapPtr +
       x + y * width;
    while (x < width)
    {
      // transparentes Pixel?
      if (*p == trancol) break;
      p ++;
      x ++;
    }

    if (x > x0)
    {
      //Rechteck gefunden
      ....
    }
  }
}
```

Nachdem Sie die Rechtecke gefunden haben, speichern Sie diese in der Datenstruktur *RGNDATA*, die Sie dann Ihrem Fenster zuweisen. Diese Struktur ist von Windows wie folgt definiert:

```
typedef struct
{
  RGNDATAHEADER rdh;
  char Buffer[1];
} RGNDATA;

typedef struct _RGNDATAHEADER
{
  DWORD dwSize;
  DWORD iType;
  DWORD nCount;
  DWORD nRgnSize;
  RECT  rcBound;
} RGNDATAHEADER;

RGNDATA *pData = new RGNData;
```

Zunächst füllen Sie den *RGNDATAHEADER* aus:

```
pData->rdh.dwSize =
  sizeof(RGNDATAHEADER);
pData->rdh.iType =
  RDH_RECTANGLES;
//Bisherige Anzahl Rechtecke
pData->rdh.nCount =
  pData->rdh.nRgnSize = 0;
//Größe der Regions auf
//Extremwerte setzen
SetRect(&pData->rdh.rcBound,
  MAXLONG, MAXLONG, 0, 0);
```

Wenn Sie nun ein Rechteck in Ihrer Bitmap ausgemacht haben, fügen Sie dieses der Region mit folgenden Codezeilen hinzu:

```
RECT *pr =
  (RECT *)&pData->Buffer;
SetRect(&pr[pData->rdh.nCount],
  x0, y, x, y+1);
//Größe der gesamten Region
//anpassen:
if (x0 < (dword)pData->
```

URLS FÜR DIE C/C++- UND WINDOWS-PROGRAMMIERUNG

Thema	Adresse
C++ Report	www.creport.com
MS VC++ Tutorials	http://msdn.microsoft.com/visualc/technical/training.asp
MS System Journal	www.microsoft.com/msj
MFC Professional Links	www.visionx.com/mfcpro/links.htm

```
    rdh.rcBound.left)
    pData->rdh.rcBound.left = x0;
  if (y < (dword)pData->
    rdh.rcBound.top)
    pData->rdh.rcBound.top = y;
  if (x > (dword)pData->
    rdh.rcBound.right)
    pData->rdh.rcBound.right = x;
  if (y+1 > (dword)pData->
    rdh.rcBound.bottom)
    pData->rdh.rcBound.bottom =
      y + 1;
  pData->rdh.nCount ++;
```

Nun können Sie aus den Regionendaten eine Gesamtregion erzeugen:

```
HRGN hrgn = ExtCreateRegion(
  NULL, sizeof(RGNDATAHEADER)
  + (sizeof(RECT) *
  AnzRechtecke), pData);
```

An dieser Stelle haben Sie prinzipiell schon alles, was Sie benötigen. Sie können nun einfach ein „normales“ Windows-Fenster erzeugen und diesem dann die berechnete Region zuweisen. Dazu verwenden Sie einfach:

```
SetWindowRgn( Window, hrgn,
  FALSE );
ShowWindow(Window,
  SW_SHOWNORMAL);
```

Damit besitzen Sie das Handwerkszeug, Fenster in jeder erdenklichen Form zu gestalten. Sie können auch die Form eines Fensters während der Laufzeit ändern. Dadurch können Sie zum Beispiel kleine Figuren programmieren, die auf dem Desktop herumlaufen. Oder Sie schreiben Fenster, die sich automatisch ihrem Inhalt anpassen. Die Anwendungsbereiche für selbstgeformte Fenster sind sehr vielseitig.

Splash-Screens

Kennen Sie Splash-Screens? Das sind kleine Bilder mit Programminformationen oder Copyright-Meldungen, die ein Programm beim Start einblendet. Würden Sie dazu ein Fenster ohne Titelleiste und Rahmen erzeugen, könnten Sie den Splashscreen – in unserem Fall eine einfache Bitmap – nicht mit optisch interessanten Effekten ein- oder ausblenden.

Wir zeigen Ihnen, wie Sie eine Bitmap auf das aktuell auf Ihrem Monitor sichtbare Bild einblenden. Leider stellt Windows keine Funktionen zur Verfügung, um (halb)transparente Fenster darzustellen. Also würde der weiter oben beschriebene Ansatz mit beliebig geformten Fenstern hier nicht funktionieren.

Verwenden Sie einen anderen Trick. Da Sie mit *Device Contexts* sowohl in Fenstern zeichnen als auch aus ihnen lesen können, legen Sie eine Kopie des Desktops an und nutzen diese als Hintergrundbild für das eigene Splash-Screen-Fenster. Sie brauchen nur den Teil des Desktops zu kopieren, den Ihr Fenster verdeckt. Dazu brauchen Sie den Handle des Desktop-Fensters:

```
HWND DesktopWindow =
  GetDesktopWindow();
HDC DesktopHDC =
  GetDC(DesktopWindow);
```

Nach diesen Zeilen greifen Sie auf den Desktop zu wie auf jedes andere Fenster.

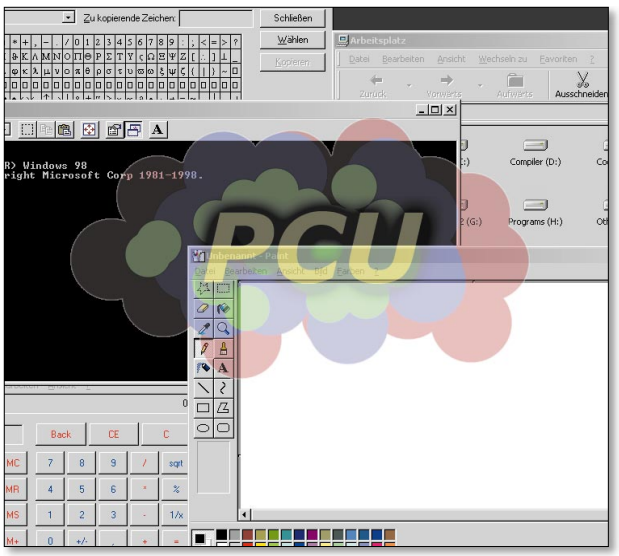
Verwenden Sie die Bitmap-Klasse, die Sie bei den Quelltexten finden. Laden Sie die Splashscreen-Bitmap, und erzeugen Sie eine neue Bitmap der gleichen Größe mit 32 Bit Farbtiefe. In die neue Bitmap kopieren Sie den später vom Splash-Screen verdeckten Desktop-Bereich:

```
picORG = new pcPicture;
picORG->load(„screen.bmp“);

picBACKGR = new pcPicture;
if (!picBACKGR) return FALSE;

//Screenshot-Buffer anlegen
picBACKGR->create( picORG->
  _width, picORG->_height, 32,
  TRUE );

//Screenshot des überdeckten
//Bereichs
RECT src = {pt.x, pt.y, pt.x +
  picORG->_width, pt.y +
  picORG->_height};
HDC hdc = GetDC(
  GetDesktopWindow());
picBACKGR->blitFROM(hdc, &src);
ReleaseDC(GetDesktopWindow(),
  hdc);
```



DER SPLASH-SCREEN wird in den Desktop eingeblendet.

Nun können Sie einfach die Splash-Screen-Bitmap mit dem Screenshot des Desktops kombinieren. Unser Beispielprogramm benutzt dabei wieder die Farbe des linken oberen Pixels als Transparenzfarbe:

```
picCOMBINE = new pcPicture(
  picORG->width(),
  picORG->height(), 32, TRUE);
if (!picCOMBINE) return FALSE;

picTRG = new pcPicture(
  picORG->width(),
  picORG->height(), 32, TRUE);
if (!picTRG) return FALSE;

bInitialized = picBACKGR->
  blitTO(picCOMBINE);

//Originalbild transparent
//über combine zeichnen
if (bInitialized)
  bInitialized = picORG->
    blitTRANSTO(picCOMBINE);

//noch ein Kopie für später
picBACKGR->blitTO(picTRG);
```

Wenn Sie auf weitere Effekte verzichten möchten, können Sie das kombinierte Bild jetzt schon in Ihr Fenster zeichnen:

```
picTRG->blitTO(hdc);
```

Blenden Sie das Bild schrittweise ein. Dazu lassen Sie das Hintergrundbild – also den Screenshot – und das kombinierte Bild ineinander übergehen. Interpolieren Sie zu diesem Zweck jeweils die RGB-Werte jedes einzelnen Pixels.

Spätestens an dieser Stelle zahlt es sich aus, daß Sie die Bitmaps in 32 Bit Farbtiefe im Speicher haben. Dadurch können Sie die Pixel leicht adressieren und ebenso einfach auf die RGB-Werte zugreifen. Eine mögliche Implementation des Überblendens könnte so aussehen:

```
//Gewichtung der Bilder
float percentage = 0.3;

//Zeiger auf die Bitmapdaten
pRGBS = (RGBQUAD*)
  (picBACKGR->_bytes);
pRGBT = (RGBQUAD*)
  (picCOMBINE->_bytes);
pRGBD = (RGBQUAD*)
  (picTRG->_bytes);

pDWS  = (dword*)
  (picBACKGR->_bytes);
pDWT  = (dword*)
  (picCOMBINE->_bytes);

//Anzahl der Pixel
dword iPixels = picORG->
  _width * picORG->_height;

//alle Pixel blenden
while(-iPixels)
{
  //nicht transparent?
  if (*pDWS++ != *pDWT++)
  {
    pRGBD->rgbRed = pRGBS->
```

```
        rgbRed - ((pRGBS->
        rgbRed - pRGBT->
        rgbRed) * percentage);
      pRGBD->rgbGreen = pRGBS->
        rgbGreen - ((pRGBS->
        rgbGreen - pRGBT->
        rgbGreen) * percentage);
      pRGBD->rgbBlue = pRGBS->
        rgbBlue - ((pRGBS->
        rgbBlue - pRGBT->
        rgbBlue ) * percentage);
    }
    ++ pRGBS;
    ++ pRGBT;
    ++ pRGBD;
  }
```

Sie brauchen nur die Pixel zu überblenden, die im Ausgangs- und Zielbild unterschiedliche Farben aufweisen. Deshalb befindet sich in der Schleife eine entsprechende *if*-Abfrage. Nun zeichnen Sie das berechnete Bild:

```
picTRG->blitTO(hdc);
```

Durch den einfachen Zugriff auf die Grafikdaten können Sie jeden beliebigen Effekt darstellen. Wenn Sie die Effekte wie in unserem Beispielprogramm animieren und zeitabhängig gestalten wollen, können Sie die Steuerung über die *WM_TIMER*-Nachricht Ihres Messagehandlers ablaufen lassen. Die Timer-Aufrufe gewährleisten, daß Animationen auf jedem Rechner unabhängig von dessen Geschwindigkeit ablaufen.

Zunächst starten Sie einen Timer:

```
SetTimer(WindowHWND, ID,
  TimeOut, NULL);
```

Dabei müssen Sie den *HWND* Ihres Fensters übergeben. Sonst weiß Windows nicht, an welches Fenster die Nachrichten gehen sollen. Die übergebene ID vom Typ *UINT* dient zur Identifikation des Timers, da Sie mehrere Timer starten können.

Der *TimeOut* gibt die Dauer der Zeitintervalle in Millisekunden an, nach denen die Nachricht an das Fenster geschickt wird.

Die Timer-Nachrichten empfangen Sie wie folgt:

```
long CALLBACK WindowProc(
  HWND hWnd, UINT message,
  WPARAM wParam, LPARAM lParam)
{
  switch (message)
  {
    ...
    case WM_TIMER:
      if (wParam == ID)
      {
          //irgendwas tun
          ...
      }
      break;
    ...
  }
  ...
}
```

Unsere Beispielprogramme, mit denen Sie frei herumexperimentieren können, finden Sie auf der Heft-CD. Tips und Tricks zu C, C++ und zur Windows-Programmierung finden Sie im Internet unter den in der Textbox unten angegebenen Adresse.

In der nächsten Ausgabe stellen wir Ihnen weitere Effekte vor, mit denen Sie Ihrer Oberfläche eine persönliche Note verpassen. PEI

Beispielprogramme sowie die Quelltexte zu den Fensterroutinen finden Sie zusammen mit der zugrundeliegenden Grafikbibliothek auf unserer Heft-CD unter der Rubrik *Praxis/PC Underground* und auf unserer Website

```
www.pc-magazin.de/magazin/
➥extras.htm
```

Klicken Sie unter *Online Extras* im Menü *Praxis* auf das entsprechende *Download*-Feld.