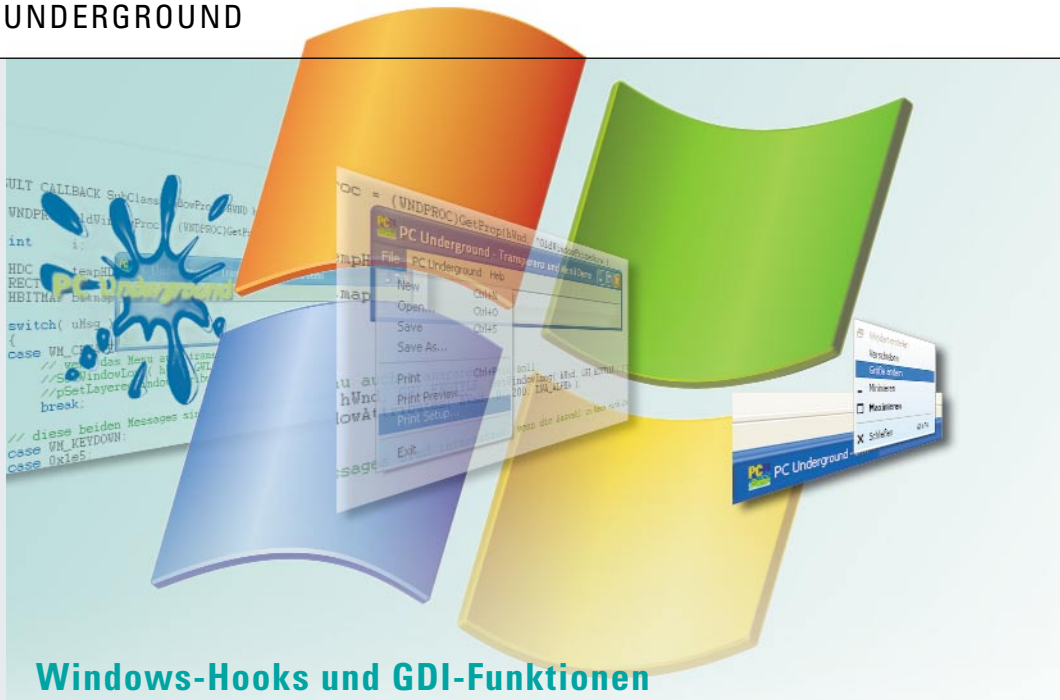


Windows-Hooks und GDI-Funktionen

Windows für Individualisten

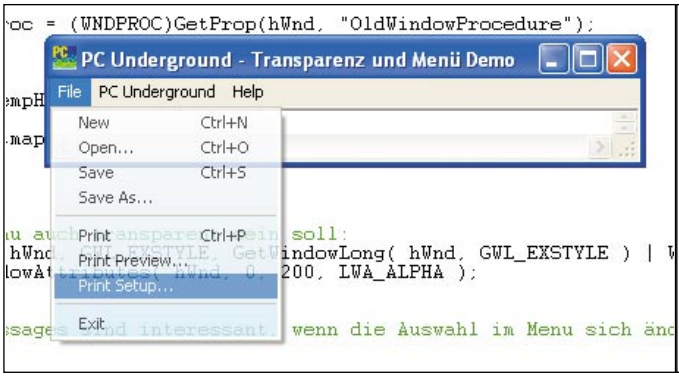
Mit einfach zu erlernenden Techniken können Sie Ihren Windows Programmen eine ganz persönliche Note geben. Wir zeigen Ihnen, was Sie mit Windows-Hooks und GDI-Funktionen erreichen können.

Carsten Dachsbacher

Gelangweilt klickt der Anwender alltägliche Windows-Dialoge ungelesen weg. Doch Sie gestalten in Ihrem Programm Fenster und Dialoge so, dass Ihnen die Aufmerksamkeit sicher ist. Dazu verwenden Sie GDI Funktionen (Graphics Device Interface), um beliebig transparente Fenster zu erzeugen und Windows-Hooks, um Window-Messages abzufangen, zu modifizieren und so ein verändertes Verhalten oder Aussehen zu erreichen.

Windows-Hooks sind ein Mechanismus, mit dem eine Funktion beliebige Events wie Nachrichten, Mausbewegungen oder Tastendrücke abfangen kann, bevor sie an eine Applikation gesendet werden. Die Funktion kann auf diese Events reagieren und in manchen Fällen sie modifizieren oder verwerfen. Solche so genannten Filterfunktionen werden unterschieden nach dem Typ von Events, die Sie beeinflussen wollen. Solche Funktionen müssen Sie durch ein Programm *installieren* – im Englischen wird von *attach* geredet. Für ein und denselben Typ wie einen Maus-Hook können Sie mehrere Filterfunktionen installieren. Es ergibt sich eine Kette von Funktionen: am Anfang der Kette steht die neueste, am Ende die älteste Funktion. Tritt das Event ein, ruft Windows die erste Funktion auf. Die Filterfunktionen müssen das jeweils nächste Element der Kette, bzw. die letzte Filterfunktion die erste, also Original-Routine aufrufen.

Hooks setzen Sie mit den Funktionen *SetWindowHookEx* und *UnhookWindowsHookEx*. Die Tabelle zeigt die verschiedenen Typen von Hooks und die jeweiligen Konstanten für die obigen Methoden.



Bunt: Unser Beispielprogramm zeigt transparente Pull-down-Menüs mit Windows Hooks.

Einhaken und Window Messages

Um Ihnen die Anwendung des Hook-Mechanismus zu demonstrieren, führen wir Sie hier durch die wichtigen Schritte unseres Beispielprogramms. Dieses installiert eine Filterfunktion, die die Window-Messages abfangen soll – nicht für alle Fenster, sondern für die Pull-down-Menüs der eigenen Applikation.
Zu Beginn des Programms, in der *WinMain*-Funktion, wird eine Fensterklasse registriert und ein Fenster mit einem Menü generiert. Dieses bekommt – wie jedes Fenster – eine *Message-Handler*-Funktion, um auf Aktionen des Benutzers und Windows-Nachrichten zu reagieren. Wenn eine Anwendung ein Fenster öffnet, wird diese Funktion mit der *WM_CREATE*-Message aufgerufen. Darin wird ein Client-Fenster angelegt und ein Hook installiert:

```
static HHOOK hHookID = 0;
hHookID = SetWindowsHookEx(
 WH_CALLWNDPROC, HookCallWindowProc,
 0,GetWindowThreadProcessId(hWnd,0));
```

Der erste Parameter ist die Konstante, die bestimmt, dass der Hook vom Typ *Window Messages* ist. Der zweite gibt die Adresse der Hook-Funktion an, zu der wir gleich kommen, und der dritte Parameter ist *0*, wenn sich diese Funktion im Code Segment des aktuellen Threads (Ihres Programms) befindet. Der letzte Parameter gibt die Prozess ID an, für die der Hook wirksam sein soll.
Die zweite hier wichtige Message des Message-Handlers ist *WM_DESTROY*. In diesem Fall wird der Hook wieder aus der Filterfunktion-Kette entfernt:

```
if ( hHookID )
  UnhookWindowsHookEx( hHookID );
```

Anschließend werfen Sie einen genaueren Blick auf die Hook-Filterfunktion, deren Kopf wie folgt definiert ist:

```
LRESULT CALLBACK HookCallWindowProc
 (int nCode, WPARAM wP, LPARAM lP )
{ CWPSTRUCT cwps; LONG handle;
  CHAR szClass[ 128 ];
```

Wenn der *nCode*- Parameter den Wert *HC_ACTION* enthält, interpretieren Sie den Wert des *lP*-Parameters als Zeiger auf eine *CPWSTRUCT*. Diese Struktur enthält alle Parameter einer Window-Message:

```
typedef struct tagCWPSTRUCT {
  LPARAM  lParam; WPARAM  wParam;
    UINT message; HWND    hwnd;
} CWPSTRUCT;
```

Also kopieren Sie die Nachricht:

```
if( nCode == HC_ACTION )
 {CopyMemory( &cwps, (void*)lP,
      sizeof( CWPSTRUCT ) );
```

So analysieren und reagieren Sie anschließend auf die Nachricht, wenn sie ein Fenster anlegen soll:

```
  switch( cwps.message )
   {case WM_CREATE:
GetClassName( cwps.hwnd, szClass,127);
if (lstrcmpi( szClass, „#32768“ )==0)
     {handle = SetWindowLong(
cwps.hwnd, GWL_WNDPROC,
      (LONG)SubClassWindowProc );
         SetProp( cwps.hwnd,
          „OldWindowProcedure“,
         (HANDLE)handle );
        }  break;  }
```

Wenn ein Fenster erzeugt wird, enthält die Nachrichten-Struktur die *WM_CREATE*-Message. Pull-down-Menüs erkennen Sie daran, dass der Klassenname *#32768* lautet. In diesem Fall verbiegen Sie den Zeiger des Window-Message-Handlers für dieses neue Pull-down-Menü auf die Funktion *SubClassWindowProc(...)*, die Sie selbst definieren. Sie speichern die Adresse des normalen Message-Handler mit der *SetProp(...)*-Methode. Die Adresse können Sie zu jedem Zeitpunkt wieder abfragen, da sie mit dem *HWND* des Pull-down-Menüs assoziiert ist. Zuletzt rufen Sie noch die nächste Funktion der Filterkette auf:

```
return CallNextHookEx(
   (HHOOK)WH_CALLWNDPROC,
 nCode, wParam, lParam );   }
```

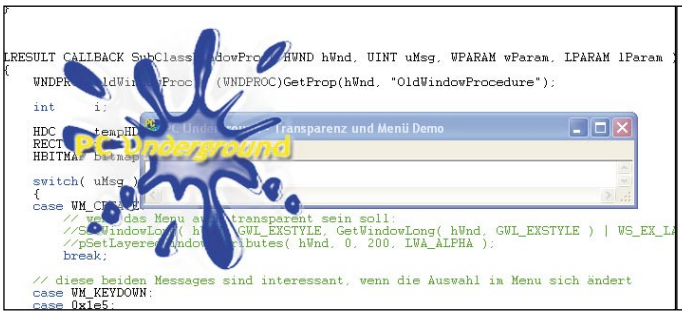
Als nächstes beschäftigen Sie sich mit der neu installierten Fensterfunktion für die Pull-down-Menüs Ihrer Applikation. Fenster-Attribute wie Transparenz können Sie hinzufügen, wenn Sie die SubClassWindowProc(...) Funktion mit der *WM_CREATE*-Message aufrufen. Unser Fall soll das Aussehen anderweitig modifizieren und den Hintergrund der Menüs mit einem Farbverlauf versehen.
Dazu bearbeiten Sie die *WM_PAINT*-Nachricht. Das Prinzip ist folgendermaßen: Wenn eine *WM_PAINT*-Nachricht eintrifft, fangen Sie diese ab und erzeugen sich zuerst zwei Device Kontexts und Bitmaps, die die Größe des Menü-Fensters besitzen:

```
HDC     tempHDC, tempHDC2;
RECT    rect;
HBITMAP bitmap, bitmap2;
GetClientRect( hWnd, &rect );
tempHDC  = CreateCompatibleDC(NULL);
bitmap   = CreateCompatibleBitmap(
  GetDC( GetDesktopWindow() ),
  rect.right, rect.bottom );
tempHDC2 = ...;bitmap2  = ...;
SelectObject( tempHDC,  bitmap  );
SelectObject( tempHDC2, bitmap2 );
```

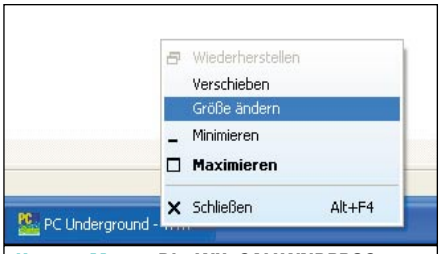
In eine Bitmap (*bitmap2*) zeichnen Sie einen Farbverlauf, wobei Ihnen z.B. die *CreateSolidBrush(..)- und FillRect(..)*-GDI-Funktionen helfen. Im anderen Bitmap (*bitmap*) möchten Sie ein Bild des Originialmenüs darstellen. Dazu bedienen Sie sich eines Tricks! Es gibt eine Window-Message (*WM_PRINTCLIENT*), die für das Drucken von Fensterinhalten gedacht ist. Diese zeichnet den Inhalt des Fensters (oder Pull-down-Menüs) in einen beliebigen Device-Kontext. Also verwenden Sie diese Methode, um sich vom Original-Message-Handler das Menü in das Bitmap bzw. den assoziierten Device-Kontext zeichnen zu lassen:

Filterfunktionstypen

Funktion	Bedeutung
WH_CALLWNDPROC	Window Messages von SendMessage(...) verschickt
WH_CBT	System Aktionen für Computer Based Training
WH_DEBUG	Verhindert den Aufruf anderer Filter
WH_FOREGROUNDIDLE	Wird bei Idle der Vordergrund-Applikation aufgerufen
WH_GETMESSAGE	Für Nachrichten von oder für *GetMessage(...)* und *PeekMessage*
WH_JOURNALPLAYBACK	Wiedergabe von Tastatur- oder Maus-Events
WH_JOURNALRECORD	Aufzeichnen von Tastatur- oder Maus-Events
WH_KEYBOARD	Bearbeiten, Modifizieren oder Verwerfen von Tastatur-Events
WH_MOUSE	Bearbeiten, Modifizieren oder Verwerfen von Maus-Events
WH_MSGFILTER	Bearbeiten oder Modifizieren von Nachrichten für Dialog/Message Boxes, Scroll Bars oder Menüs von Applikationen
WH_SYSMSGFILTER	wie oben nur für das System, nicht für Applikationen
WH_SHELL	Aktionen von Top-Level-Fenstern (Erzeugen, Zerstören)



Der etwas andere About Dialog: Per Pixel-Transparenz erstaunen den Anwender plakative Effekte mit Layered Windows.



Kontext-Menüs: Die *WH_CALLWNDPROC*-Hooks greifen auch hier ein.

```
// alter Message Handler
WNDPROC oldWindowProc = (WNDPROC)
  GetProp(hWnd,“OldWindowProcedure“);
// Bitmap löschen mit Menuhintergrund
FillRect( tempHDC, &rect,
  (HBRUSH)GetSysColor(COLOR_MENU) );
CallWindowProc( oldWindowProc, hWnd,
  WM_PRINTCLIENT, (LPARAM)tempHDC,
  PRF_CLIENT | PRF_CHECKVISIBLE );
```

Der letzte Parameter enthält Flags, die angeben, dass Sie den Fensterinhalt darstellen und auf Sichtbarkeit prüfen. Jetzt kopieren Sie die Bitmap mit dem Menü über die Bitmap mit dem Farbverlauf. Dabei stanzen Sie die Menü-Hintergrundfarbe aus, damit Sie nur die Pixel, die Teile des Menüs enthalten, überschreiben:

```
TransparentBlt( tempHDC2, rect.left,
rect.top, rect.right - rect.left,
rect.bottom - rect.top, tempHDC,
rect.left, rect.top, rect.right  -
rect.left, rect.bottom - rect.top,
  GetSysColor( COLOR_MENU ) );
```

Jetzt ist das Menü wie geplant gezeichnet und Sie können es auf den Bildschirm kopieren:

```
BitBlt( GetDC( hWnd ),
rect.left, rect.top, rect.right -
rect.left, rect.bottom - rect.top,
  tempHDC2, 0, 0, SRCCOPY );
```

Nachdem Sie die Device-Kontexts und Bitmaps wieder freigegeben haben, müssen Sie Windows noch mitteilen, dass der Fensterbereich fertig gezeichnet wurde:

```
ValidateRect( hWnd, &rect );return 0;
```

Wenn Sie die Funktion mit Rückgabewert *0* verlassen, haben Sie die *WM_PAINT*-Nachricht somit abgefangen, d.h. es werden keine weiteren Hook-Filterfunktionen oder Message-Handler aufgerufen.

Genauso, wie Sie mit *ValidateRect(..)* angeben, dass ein Bereich gezeichnet wurde, müssen Sie dafür sorgen, dass Teile des Menüs, die neu gezeichnet werden müssen, entsprechend bearbeitet werden. Dies ist der Fall, wenn entweder ein neues Menüelement mit der Maus oder Tastatur ausgewählt wird. Die beiden entsprechenden Nachrichten fangen Sie ab und sorgen dafür, dass das Menü vollständig neu gezeichnet wird. Allerdings geben Sie die Nachricht weiter, da die Internas, z.B. welches Element selektiert wurde, sonst nicht aktualisiert werden:

```
case WM_KEYDOWN:case 0x1e5:
  GetClientRect (hWnd,&rect);
  InvalidateRect(hWnd,&rect,false);
  return CallWindowProc( ...);
```

Transparente Fenster

Zuvor haben wir bereits erwähnt, dass Sie Ihre Pull-down-Menüs auch transparent gestalten können. Dazu bietet Windows 2000 und XP entsprechende Funktionalität. Die entsprechenden Methoden, um diese Funktionen zu nutzen, sind in der Bibliothek *User32.DLL* enthalten. Diese müssen Sie, unter Umständen je nach SDK-Version, selbst definieren und aus der DLL mit *GetProcAdress(..)* laden (siehe Beispielprogramm). Die beiden Funktionen sind *SetLayeredWindowAttributes* und *UpdateLayeredWindow*. Mit ersterer können Sie ein Fenster ganz einfach transparent machen wie z.B. Pull-down-Menüs:

```
case WM_CREATE:
 SetWindowLong(hWnd,GWL_EXSTYLE,
   GetWindowLong(hWnd,GWL_EXSTYLE)|
   WS_EX_LAYERED );
 SetLayeredWindowAttributes( hWnd,
   0, 200, LWA_ALPHA ); break;
```

Mit *SetWindowLong(..)* fügen Sie dem Fenster das *WS_EX_LAYERED*-Attribut hinzu. Dieses ist Vorraussetzung für transparente Fenster. Mit dem zweiten Aufruf ist das Fenster schon transparent! Aufwändiger, aber flexibler, gestaltet sich die Verwendung von *UpdateLayeredWindow*. Hiermit können Sie ein Fenster erzeugen, wobei Sie für jeden Pixel einen anderen Transparenzwert angeben können!

Analog zur *WM_CREATE*-Message werden an den Dialog Message Handler *WM_INITDIALOG*-Nachrichten versendet. Auf diese reagieren Sie, indem Sie das *_EX_LAYERED*-Attribut setzen. Jetzt können Sie ein Bitmap erstellen, das die Farben und die Transparenz enthält. Im Beispiel erzeugen Sie zuerst Device-Kontexts und eine Bitmap mit dem Pixelformat des Desktops:

```
GetClientRect( hDlg, &rect );
HDC dcScreen = GetDC( NULL );
HDC dcMemory =
  CreateCompatibleDC( dcScreen );
bmp = CreateCompatibleBitmap(
  GetDC( GetDesktopWindow() ),
  rect.right, rect.bottom );
```

Anschließend erzeugen oder laden Sie ein 32-Bit-Image (mit Alpha Kanal) in den Speicher. Dazu verwenden Sie einen Speicherbereich und einen *BITMAPINFOHEADER*, der das interne Bild- und Pixelformat beschreibt. Dieses Bild kopieren Sie dann in den Device-Context *dcMemory*, wobei der Wert *256* exemplarisch als Größe des Bitmaps dient:

```
DWORD image[ 256 * 256 ];
BITMAPINFO   *bitmapinfo = ?;
SetDIBitsToDevice( ...);
```

Jetzt können Sie die Attribute des Fensters mit *UpdateLayeredWindow setzen*, wobei Sie bitte weitere Ausführungen im Web beachten: **http://msdn.microsoft.com/library/en-us/gdi/bitmaps_3b3m.asp**.

Zu den Attributen zählt auch die Position und Größe des Fensters, der Ursprung der Bitmap (*srcPointer*) und natürlich die Beschreibung, wie die Farb- und Transparenzwerte zu verarbeiten sind:

```
BLENDFUNCTION blendPixelFunction =
  {AC_SRC_OVER,0,255,AC_SRC_ALPHA};
```

Dabei bedeutet *AC_SRC_ALPHA*, dass das Bild Alpha-Werte besitzt. Beachten Sie dabei, dass die API mit *premultiplied alpha* arbeitet, d.h. die Farbwerte multiplizieren Sie vorab mit dem Alpha-Wert. Dabei interpretieren Sie Alpha-Werte zwischen *0* und *255* als Zahlen zwischen *0* und *1*. *AC_SRC_OVER* ist die momentan einzig unterstützte Blending Operation, die das Quellbild (das Fenster) über das Zielbild (Hintergrund) legt. Legen Sie also die restlichen Attribute fest und rufen Sie die Funktion *UpdateLayeredWindow* auf. : et

Info:
www.dachsbacher.de/pcu
http://msdn.microsoft.com