API in Access

Vortrag auf der 10. AEK,
Oktober 2007, Nürnberg

Jörg Ostendorp
ostendorp@ecodatadesign.de

Inhalt

Einige Grundlagen und Begriffe ... 1
"WinApi" ... 1
„Apis“ verwenden ... 2
Handles ... 2
Gerätekontexte ... 2
Subclassing ... 4
Nahezu crashfreies Subclassing ... 6
Fehlerbehandlung ... 8

Ressourcen und Tools ... 8
MSDN ... 8
SDK ... 8
WinApi-Viewer ... 9
ApiViewer 2004 ... 9
API-Guide ... 9
Spy++ ... 10
UISpy ... 10
Winspector ... 10
ControlSpy ... 11
Accessibility-Tools ... 11

Fensterarchitektur von Access ... 12
MDI ... 12
Access 2000-2003 ... 13
Ein Blick auf Access 2007 ... 17

Praxisbeispiele ... 19
1. Handles ermitteln ... 20
2. Fensterpositionen ... 21
3. Fensterklassen manipulieren ... 22
4. Mausradbewegung per Peekmessage ... 22
5. Platz sparen mit Kontextmenüs ... 22
6. Splitter ... 23
7. De/Activeate-Event auch für Popup-Formulare ... 24
8. Größenänderungen von Formularen abfangen ... 24
9. Drag and Drop ... 25
10. Accessibility ... 25
11. SHFileOperation ... 26

Links ... 27

Einige Grundlagen und Begriffe

"WinApi"

WinApi ist die Abkürzung für das „Windows Application Programming Interface“. Dahinter verbergen sich eine Reihe von C-Dlls, die, wie der Name schon sagt, als Schnittstelle zwischen Windows auf der einen und Applikationen auf der anderen Seite fungieren. Das WinApi ist als relativ systemnahe Zugriffsschicht einerseits eine Grundvoraussetzung, damit eine Anwendung überhaupt unter Windows läuft - selbst moderne Layer wie das .Net-Framework bauen zumindest weitgehend auf ihm auf - aus praktischer Sicht stellt es mit seinen weit über 6.000 Prozeduren zugleich aber auch einfach eine nahezu unerschöpfliche und kostenlose Ressource für jeden Windows-Programmierer dar, unabhängig von der eingesetzten Entwicklungsumgebung. Dies umfasst z.B. die Verwendung der Windows-Dialoge, Programmierung der Windows-Controls, Auslesen von Systeminformationen, Durchführen von Dateioperationen, Fensterinteraktion und -manipulation, Graphikausgaben, Steuerung von Ein- und Ausgabegeräte, Registry, Zwischenablage und Speicher, Nutzerverwaltung, Ressourcen, Netzwerke, Drucker, Internet etc. etc.

Der Mehrwert, den das WinApi auch für Access-Programmierer bietet, dürfte somit auf der Hand liegen: Die Erweiterung von Access um eine Fülle zusätzlicher Möglichkeiten. Leider ist die API-Programmierung nicht immer ganz trivial und i.d.R. mit einem größeren Aufwand verbunden, als bei der Verwendung "höherer Objektmodelle". Auch die Tatsache, daß die gesamte Dokumentation von Microsoft primär für C und C++-Entwickler geschrieben wurde, erweist sich nicht unbedingt als sonderlich hilfreich. Ob man für den Mehrwert den höheren Aufwand in Kauf nehmen sollte, muß natürlich jeder selbst entscheiden. Kriterien sind in meinen Augen die Notwendigkeit von zusätzlichen Funktionalitäten, verfügbare Alternativen (z.B. OCX), der Kostenfaktor für die Entwicklung und nicht zuletzt der Spaßfaktor!

Angesichts des ähnlich leistungsstarken aber bequemer zu programmierenden .Net-Frameworks stellt sich auch die Frage, ob das WinApi als direkte Schnittstelle überhaupt noch zeitgemäß ist. Auf längere Sicht gesehen ist (war) es für die meisten Programmierer sicherlich ein Auslaufmodell. Da Access derzeit aber kein VBA.Net kann, ist das WinApi zumindest für Access-Entwickler genauso zeitgemäß wie ihre Programmiersprache VBA.

„Apis“ verwenden

Die Verwendung einer API-Funktion ist denkbar einfach: Man sucht sich die richtige Funktion aus und bindet sie über die entsprechende Declare-Anweisung ein*. Syntax:

```
[Public | Private] Declare Function name Lib "libname" [Alias "aliasname"] [([arglist])] [As type]
```

Die Gemeinheit liegt natürlich darin, ersteinmal die richtige Funktion zu finden und diese dann auch noch korrekt anzuwenden. Dafür wird man auf Dauer leider kaum umhin können, auch mal die ein oder andere Dokumentation zum Thema zu wälzen, allen voran das MSDN.

(*Wer es etwas bequemer mag, kann anstelle der manuellen Deklaration auch auf eine der TLBs mit den API-Deklarationen zurückgreifen, die im Internet kursieren. Allerdings ist hierfür ein zusätzlicher Verweis erforderlich).

Handles

"Handle" (zu deutsch "Griff" oder "Henkel") ist ein Begriff der bei der API-Programmierung an allen Ecken und Enden verwendet wird. Wie die deutsche Übersetzung schon andeutet, ermöglichen Handles in Form von Zugriffsnummern, die indirekte Ansprache von dahinterliegenden Objekte, wie z.B. einem Fenster oder einem Bitmap.

Die Ermittlung von Handles erfolgt zumeist durch den Rückgabewert eines Funktionsaufrufs, z.B.

```
hWnd = FindWindow (...)

hBitmap = LoadImage (...)
```

Gerätekontexte

Gerätekontexte sind windowsinterne Datenstrukturen zur Verwaltung von Ausgabegeräten wie Druckern, Bildschirmen aber auch Fenstern. Sie kommen also überall dort ins Spiel wo irgendetwas mit den Ausgabegeräten angestellt werden soll, z.B. beim Erzeugen eines Druckauftrags oder der graphischen Darstellung eines Textfeldes auf einem Formular. Ihr unschätzbarer Vorteil besteht darin, daß sie eine geräteunabhängige Arbeitsweise ermöglichen, d.h. der Entwickler arbeitet mit dem Gerätekontext (bzw. dessen Handle), Windows kümmert sich um den ganzen Rest.

Wichtige Gerätekontext-Funktionen sind:

CreateDC	Erzeugt einen Gerätekontext für physikalisches Gerät (Monitor, Drucker)
CreateCompatibleDC	Erzeugt einen Speichergerätekontext, der mit einem bestimmten Gerät kompatibel ist
GetDC	Ermittelt das Handle für den Gerätekontext des Clientbereichs
GetWindowDC	Ermittelt das Handle für den Gerätekontext des Fensters
GetDCEx	Ermittelt ein Handle für den Gerätekontext eines Fensters (Ex..)
ReleaseDC	Gibt den Gerätekontext wieder frei
DeleteDC	Löscht einen Gerätekontext (vgl. CreateDC)

Damit es nicht zu Speicherlecks kommt, sollte man die Dokumentation ernst nehmen und Objekte bzw. Gerätekontexte nach Verwendung stets entsorgen bzw. freigeben. Schema:

```
'Gerätekontext ermitteln/erstellen:
DC = GetWindowDC(hwnd) 'CreateCompatibleDC(GetDC(0))
'GDI-Object erstellen:
NewBrush = CreateSolidBrush(...)
'GDI Object zuweisen:
OldBrush = SelectObject(DC, NewBrush)
'.... GDI-Objekt verwenden......
'Ursprüngliches GDI-Objekt wieder zuweisen:
SelectObject DC, OldBrush
'Neues GDI-Objekt löschen:
DeleteObject NewBrush
'Gerätekontext freigeben:
ReleaseDC hwnd, DC
```

Um ganz sicher zu gehen, kann man seine Anwendung im Taskmanager auf nicht zerstörte GDI-Objekte überprüfen, indem man sich unter ->Taskmanager -> Prozesse ->Ansicht -> Spalte auswählen -> GDI-Objekte, die Anzahl der geöffneten GDI-Objekte anzeigen läßt.

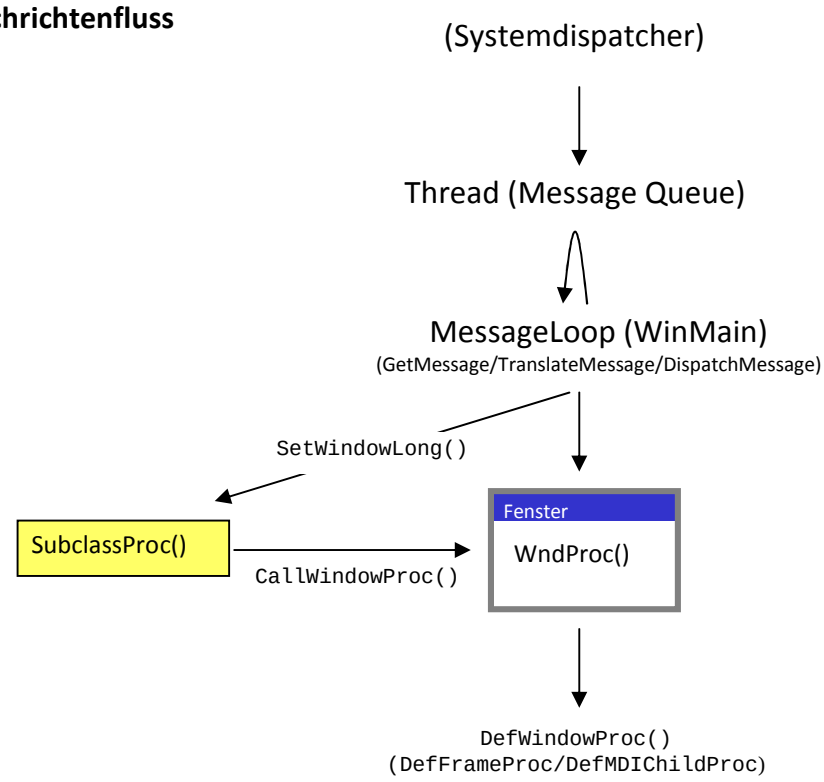
Ein Beispiel mit Gerätekontexten und GDI-Objekten findet sich in der Demo-DB unter "6. Splitter".

Subclassing

Windows ist ein System, das in hohem Maße auf dem Austausch von Nachrichten basiert. Bewegt der Nutzer z.B. die Maus, wird dies, etwas verkürzt gesagt, vom Systemdispatcher registriert und sofort eine entsprechende Nachricht (WM_MOUSEMOVE) an die passenden Threads bzw. deren MessageQueues verschickt. Über einen MessageLoop (GetMessage, TranslateMessage, DispatchMessage) fragt eine Anwendung das Queue permanent ab und verteilt die Nachricht an die Nachrichtenprozedur (WndProc) des passenden Fensters, von wo aus sie in aller Regel wiederum der Standardnachrichtenverarbeitung von Windows zugeführt wird (je nach Anwendungs- und Fenstertyp per DefWindowProc(), DefFrameProc() oderDefMDIChildProc())

Beim Subclassing wird nun mithilfe der Funktion `SetWindowLong`, die Adresse der Funktion für die Nachrichtenverarbeitung eines Fensters (`WndProc`) geändert, so daß sämtliche Nachrichten nicht mehr an die WndProc sondern an eine neue, vom Entwickler kontrollierbare Funktion umgeleitet werden. Innerhalb dieser neuen Prozedur kann ein Entwickler dann direkt auf die Nachricht reagieren und z.B. ein passendes Event bereitstellen. Insofern die Nachricht nicht unterdrückt werden soll, wird sie abschließend mit der Funktion CallWindowProc wieder an die WndProc-Adresse übergeben.

Nachrichtenfluss



Der entsprechende Code sieht im einfachsten Fall z.B. so aus:

```
Option Compare Database
Option Explicit

Private Declare Function CallWindowProc _
        Lib "user32" _
        Alias "CallWindowProcA" ( _
            ByVal lpPrevWndFunc As Long, _
            ByVal hWnd As Long, _
            ByVal Msg As Long, _
            ByVal wParam As Long, _
            ByVal lParam As Long _
        ) As Long

Private Declare Function SetWindowLong _
        Lib "user32" _
        Alias "SetWindowLongA" ( _
            ByVal hWnd As Long, _
            ByVal nIndex As Long, _
            ByVal wNewWord As Long _
        ) As Long

Private Const GWL_WNDPROC  As Long = (-4)

Private AlteProc As Long
Private m_Hwnd As Long

Public Function Subclass(ByVal hWnd As Long) As Boolean
    'Rückgabewert
    Subclass = False

    If AlteProc = 0 Then
        'Übergebenen Handle merken
        m_Hwnd = hWnd
        'Alte Proc merken und Neue Proc zuweisen
        AlteProc = SetWindowLong(m_Hwnd, GWL_WNDPROC, AddressOf NeueProc)
        'Ruckgabewert
        Subclass = CBool(AlteProc)
    End If
End Function

Public Function UnSubclass()
    'Alte Proc wieder zuweisen
    If AlteProc <> 0 Then
        SetWindowLong m_Hwnd, GWL_WNDPROC, AlteProc
        m_Hwnd = 0
        AlteProc = 0
    End If
End Function

Private Function NeueProc(ByVal hWnd As Long, ByVal Msg As Long, _
                      ByVal wParam As Long, ByVal lParam As Long) As Long

    On Error Resume Next
    Select Case Msg
        Case WM_X
            'Auf Nachricht reagieren
        Case WM_Y
            'Auf andere Nachricht reagieren
    End Select

    Nachricht an alte Proc weiterleiten
    NeueProc = CallWindowProc(AlteProc, hWnd, Msg, wParam, lParam)

    oder je nach Nachricht Alte Proc ggf. nicht aufrufen
    'NeueProc = 0

End Function
```

Subclassing ist innerhalb von VB(A)-Anwendungen ist nicht völlig problemlos:

Beim Codieren schlägt der etwas ominöse "IDE-Bug" zu, der es fast unmöglich macht, zwischen der Code- und der normalen Ansicht hin und her zu wechseln: Sobald das Subclassing nur einmal aktiv war führt ein entsprechender Versuch zu einem Absturz oder Hängenbleiben von Access.

Bei der Ausführung kann es Komplikationen mit Antivirenprogrammen geben, insbesondere mit Kaspersky, wodurch in meinen Augen ein praktikables Subclassing auf herkömmlichem Weg mit VBA nicht mehr möglich ist.

Unangenehme Nebeneffekte ergeben sich außerdem dadurch, daß eine SubclassProc sich wegen des AddressOf-Operators immer in einem Standardmodul befinden muß, wodurch das Handling mit mehreren Subclassing-Instanzen etwas umständlicher wird.

Nahezu crashfreies Subclassing

Eine Alternative zum herkömmlichen Vorgehen ist der Einsatz von ActiveX-Komponenten. Während es für VB mehrere Tools gibt (unter anderem von Microsoft), ist das Open-Source-ActiveX **SSubtmr6.dll** von Steve McMahon, das einzige mir bekannte Tool, das seinen Dienst auch klaglos unter VBA verrichtet.

Die Vorteile des Einsatzes dieser Komponente sind

- Problemlose Mehrfachinstanzierungen (bei Klassen ist kein Standardmodul für AddressOf notwendig)
- Wechsel Design-/Formularansicht möglich, insofern Subclassing z.B. im Unload rechtzeitig aufgehoben wird
- Öffnen der IDE ebenso wie das Schreiben von Code/CBF während aktivem Subclassing möglich.

Schönheitsfehler:

- Debugger darf durch Laufzeitfehler nicht angesprungen werden. Fehlerbehandlung am besten immer sofort einbauen!
- Haltepunkte funktionieren nur, wenn das Debuggen ohne Fehler bis zum Ende der Prozedur fortgeführt werden kann. Also am besten darauf verzichten oder Subclassing aufheben.
- Nicht kompilieren wenn Subclassing aktiv ist!
- Es können nicht immer alle Ereignisse wie beim normalen Subclassing verarbeitet werden. Problematisch ist z.B. die Unterdrückung einer WM_CLOSE-Nachricht an das Applikationsfensters.
- Es ist ein ActiveX....

Verwendung:

Die Verwendung der SSubtmr6.dll ist nur innerhalb von Klassen (Klassenmodule, Formulare, Berichte) möglich. Dazu wird das Interface ISubclass über ein entsprechendes Implements-Statement eingebunden und anschließend die Property ISubclass_MsgResponse und die Funktion ISubclass_WindowProc angelegt.

Im Gegensatz zur herkömmlichen Methode werden innerhalb von Access immer nur die angegebenen, nicht alle Nachrichten überwacht. Um eine Nachricht zur Überwachung hinzuzufügen, stellt die SSubTmr6.dll die Methode AttachMessage bereit, zum Entfernen DetachMessage. Die Gegenwärtig bearbeitete Nachricht läßt sich per CurrentMessage abfragen, so daß man innerhalb der ISubclass_MsgResponse z.B. bestimmen kann, in welcher Form bzw. wann auf die Nachricht reagiert werden soll (PostProcess, PreProcess, Consume; nicht zu verwechseln mit der eigentlichen Reaktion auf die Nachricht, die in der ISubclass_WindowProc erfolgt).

```
Option Compare Database
Option Explicit


Implements SSubTimer6.ISubclass
Public mhwnd as long

...
    'Message zur Subclassing-Proc hinzufügen
    AttachMessage Me, mhwnd, WM_XY
...
    'Message aus Subclassing-Proc entfernen
    DetachMessage Me, mhwnd, WM_XY
...


Private Property Let ISubclass_MsgResponse( _
                     ByVal RHS As SSubTimer6.EMsgResponse)
    'dummy
End Property

Private Property Get ISubclass_MsgResponse() As SSubTimer6.EMsgResponse
    Select Case CurrentMessage
        Case WM_XY
            ISubclass_MsgResponse = emrPostProcess
    End Select
End Property


'Eigentliche Fenster-Prozedur:
Private Function ISubclass_WindowProc( _
                 ByVal hwnd As Long, ByVal iMsg As Long, _
                 ByVal wParam As Long, ByVal lParam As Long _
                 ) As Long

    On Error Resume Next
    Select Case iMsg
        Case WM_XY
            'reagieren, ggf. vorher noch Handle abgleichen
    End Select

End Function
```

Konkrete Beispiele für die Verwendung sind in der Demo-DB enthalten (Beispiele 7.-9.).

Fehlerbehandlung

Verursachen Api-Funktionen einen Fehler, wird dieser automatisch err.LastDllError gespeichert. Diese Fehlernummer lässt leicht sich mit der Funktion `FormatMessage()` im Klartext auslesen.

Ein Beispiel hierzu findet sich in der Demo-DB im Modul modError.

Ressourcen und Tools

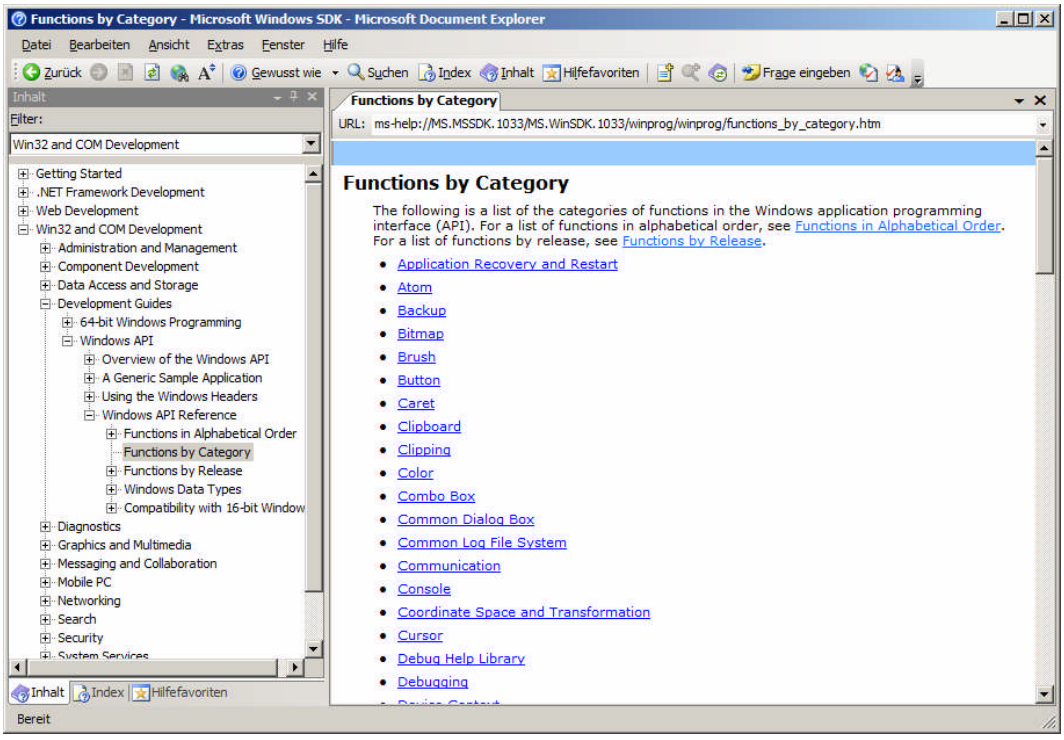
(Links siehe Anhang)

MSDN

Microsoft Developer Network. **DAS** Nachschlagewerk und erste Anlaufstelle zu allen Microsoft-Produkten.

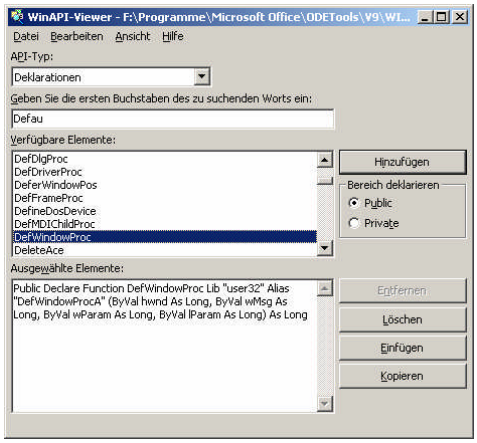
SDK

Software Development Kit für Windows Vista (lauffähig auch unter XP SP2). Enthält neben zahlreichen Tools u.a. auch die Dokumentation zum "WIN32 and COM-Development" aus dem MSDN für den Offline-Betrieb. Der Download fällt mit über einem GB ziemlich üppig aus, ist in meinen Augen aber ein Muß!



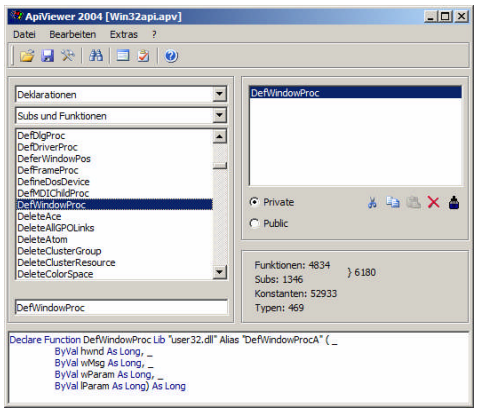
WinApi-Viewer

Offizielles (und nur deshalb erwähntes) Addin von Microsoft für die Übernahme von VB(A)-API-Deklarationen. In den ODE-Tools enthalten. Hoffnungslos veraltet (Stand 1998).



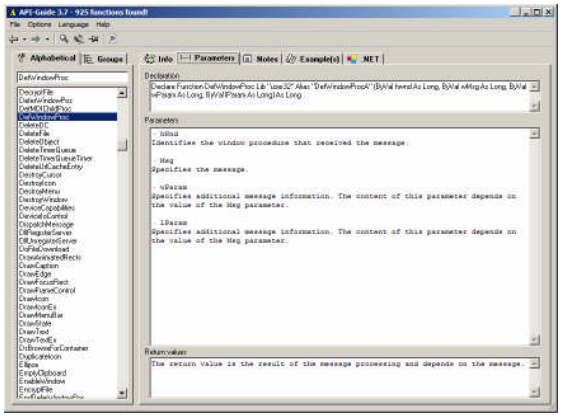
ApiViewer 2004

Komfortabler Ersatz für den WinApi-Viewer, von Christoph von Wittich. Das Programm ist separat oder als Addin aus dem VBA-Editor aufrufbar. Es enthält neben den Deklarationen für/bis Windows XP auch die Deklarationen für die GDI+.



API-Guide

Leider nicht mehr fortgeführtes Tool vom KPD-Team. Enthält Dokumentation und Beispiele zu 925 API-Funktionen für VB. Die Beispiele lassen sich meist nicht direkt 1:1 nach Access übernehmen, sind aber trotzdem sehr hilfreich. Das Programm benötigt IIRC die VB5-Runtime, die man ggf. separat downloaden und installieren muß.

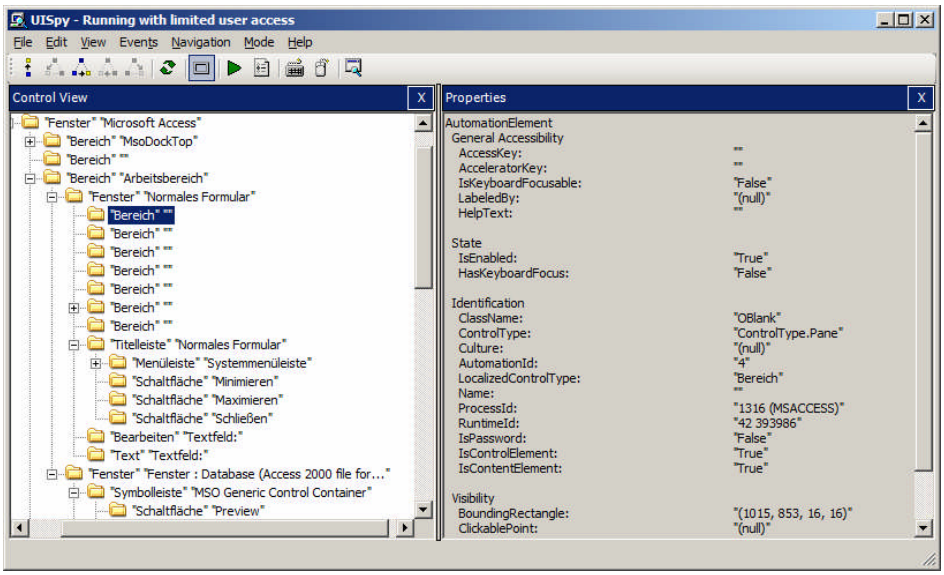


Spy++

Der Standard-Spyer von Microsoft aus dem Visual Studio, kein separater Download (und keine Erfahrungen meinerseits).

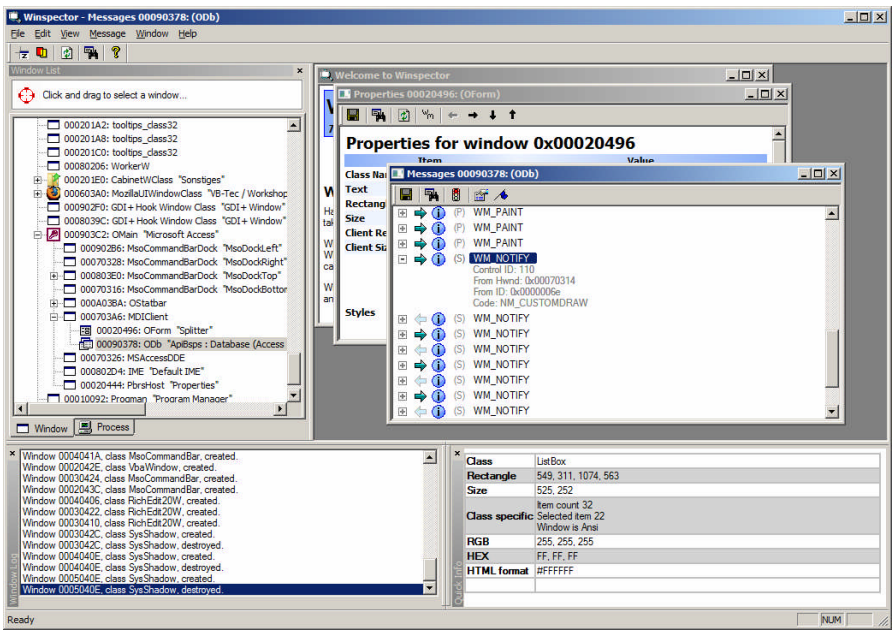
UISpy

Relativ neuer Spyer von Microsoft aus dem .Net-Framework 3.0. Auch im Vista-SDK enthalten.



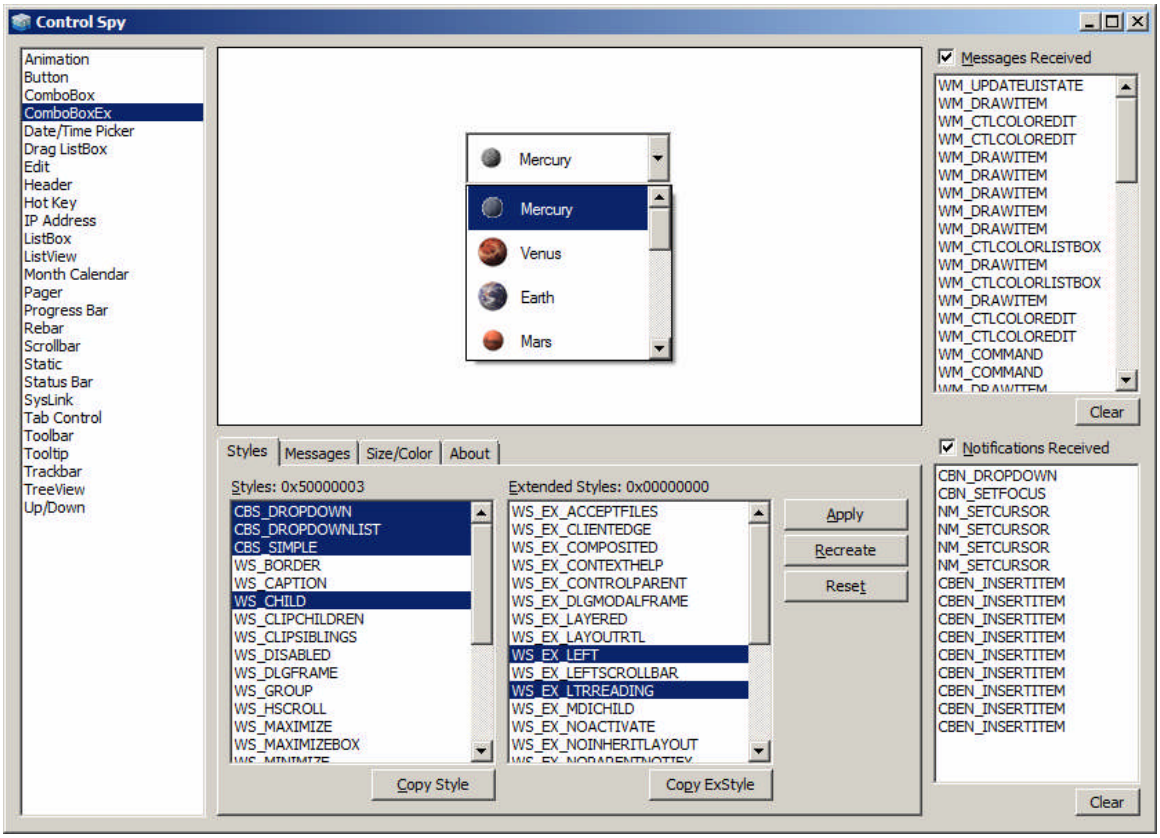
Winspector

Freeware von Russ Freeman. Vollwertiger Spyer, ausgesprochen nützlich, sollte man haben!



ControlSpy

MS-Spyer speziell für die Windows Controls. Ermöglicht das Bearbeiten der Styles und die Analyse der Messages und Notifictions.



Accessibility-Tools

Vier verschiedenen Tools von MS. Insbesondere ganz interessant: der **AccExplorer** zur Analyse von Accessible-Objects. Recht ähnlich aufgebaut wie ein Spyer.

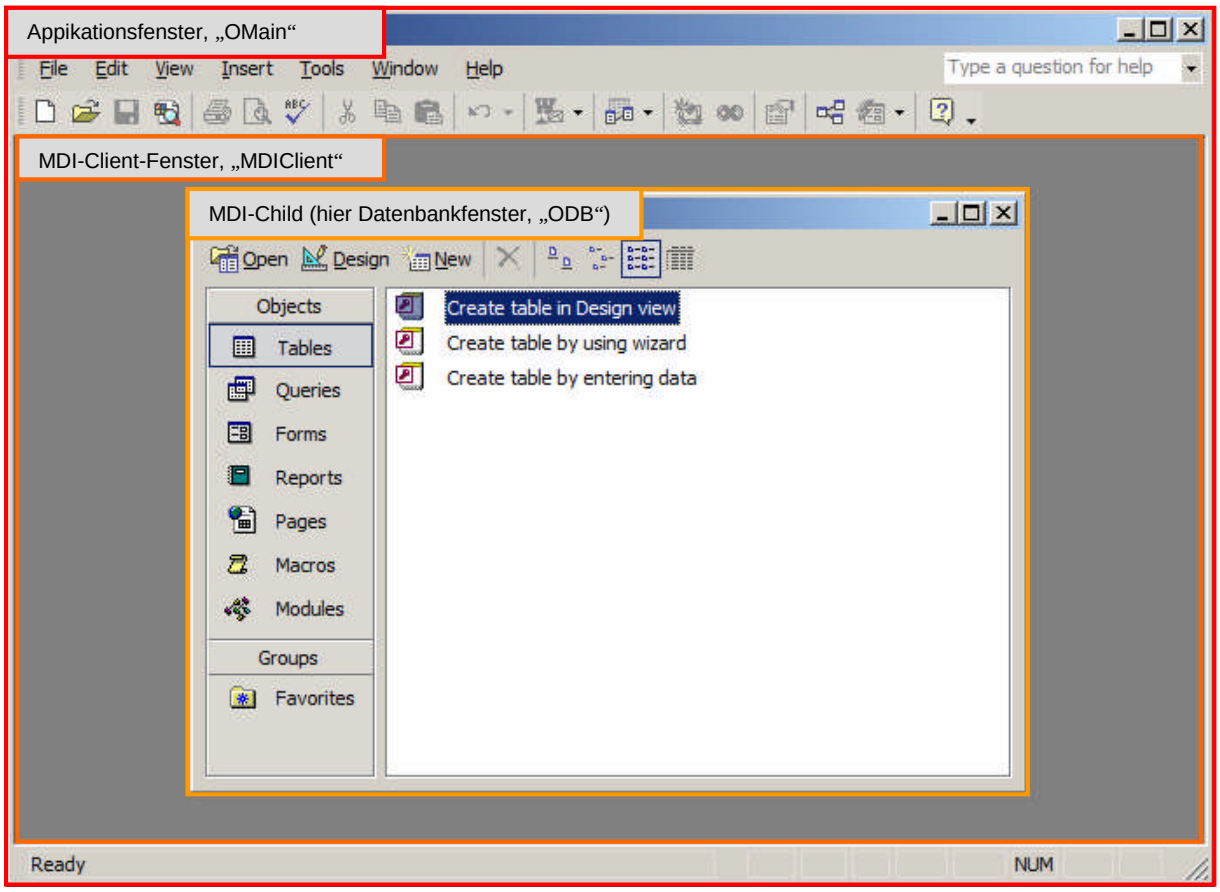
Fensterarchitektur von Access

MDI

Access ist eine Multiple Document Interface (MDI)-Anwendung. Hauptmerkmale dieses Anwendungskonzepts sind die spezielle Fensterarchitektur und damit verbunden auch eine spezielle Nachrichtenverarbeitung (s. DefFramProc, DefMDIChildWindowProc). Jede MDI-Anwendung besteht aus mindestens drei Fenstertypen:

1. dem Frame-Fenster
2. dem MDI-Client-Fenster
3. abgeleiteten MDI-Child-Fenstern (und/oder Popups)

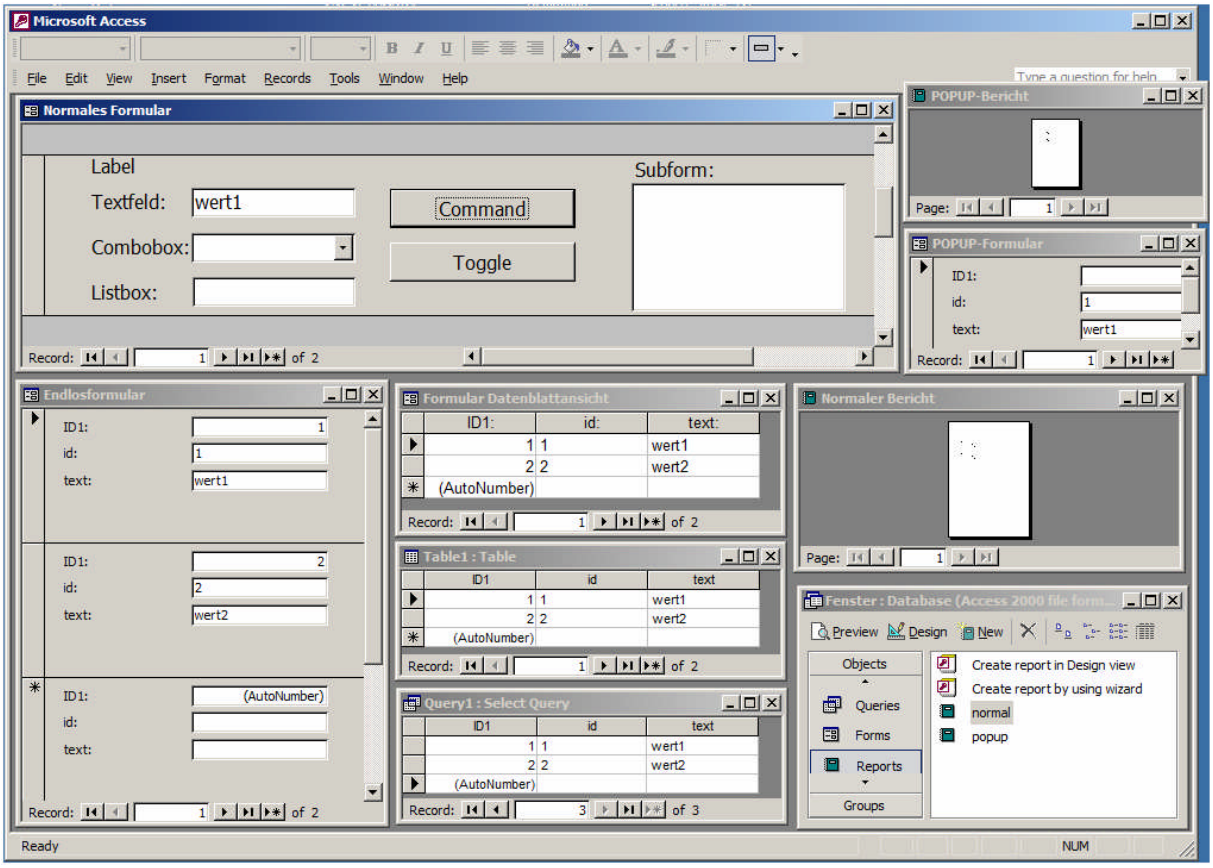
Das Frame-Fenster entspricht in Access dem Applikationsfenster (Fensterklasse "OMain"), das MDI-Client-Fenster ("MDIClient") ersetzt quasi den Clientbereich normaler Fenster einer Single Document Interface (SDI) Anwendung. MDI-Child-Fensters (verschiedene Klassen, Style = WS_CHILD, Exstyle = WS_EX_MDICHILD) sind, wie der Name schon sagt, Kinder des MDI-Clientfensters, können also nur innerhalb dieses Fensters bewegt werden. Popup-Fenster (Style = WS_POPUP) schließlich sind hierarchisch auf der gleichen Ebene wie das Frame-Fenster angesiedelt.



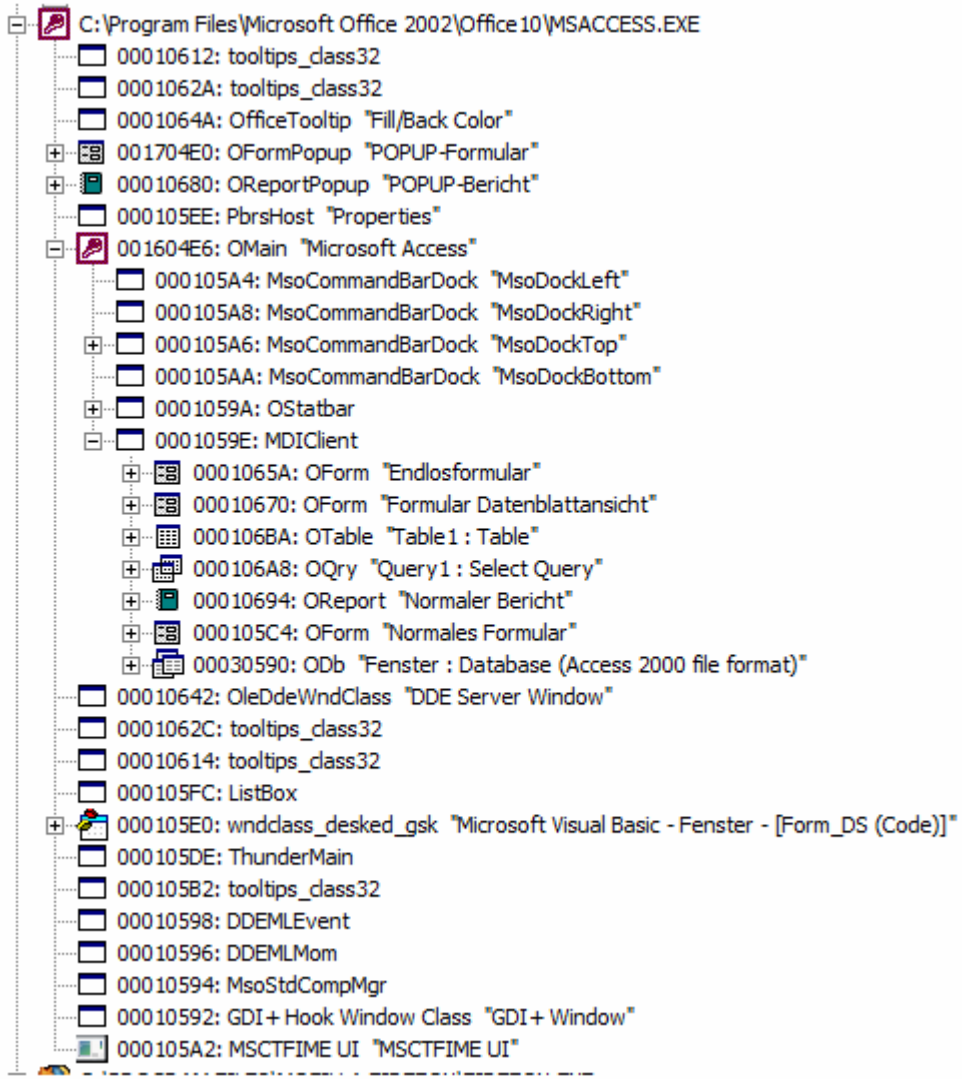
Access 2000-2003

Die nachfolgende Abbildung zeigt eine Access-XP-Datenbank, in der, mit Ausnahme von Pivots, alle möglichen Fenstertypen geöffnet sind:

1. das Datenbankfenster
2. ein normales Formular (mit allen Steuerelemente, außer Graph und OLE)
3. ein Endlosformular
4. ein Formular in Datenblattansicht
5. eine Tabelle
6. eine Abfrage
7. ein normaler Bericht
8. ein Popup-Formular
9. ein Popup-Bericht

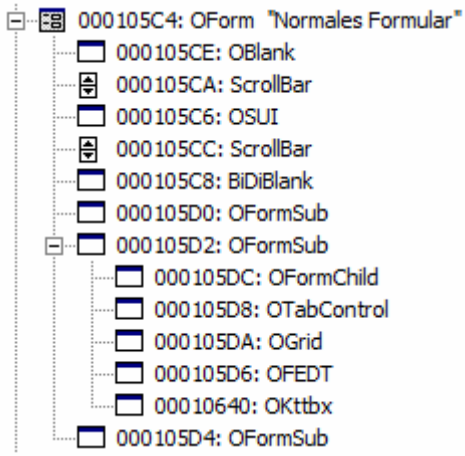


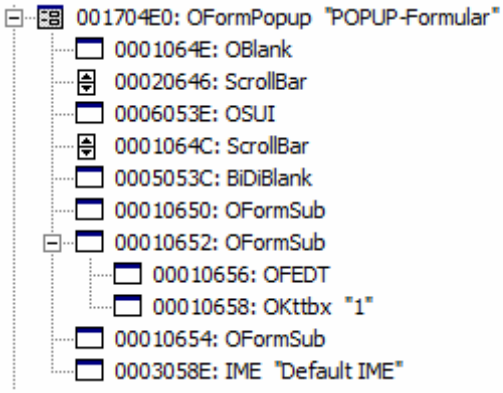
Ein Blick in einen Spyer (hier Winspector) zeigt folgende Fenster-Hierarchie (Übersicht):

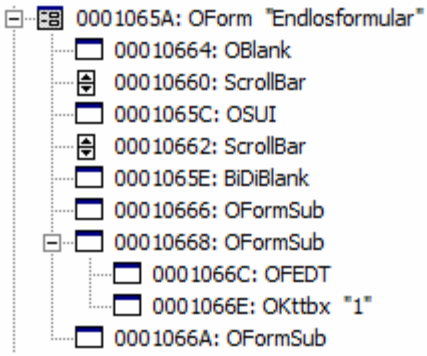


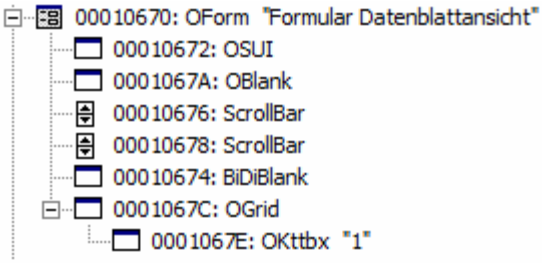
Wie man sieht, gibt es eine ganze Reihe an Nicht-Kind-Fenstern, von denen jedoch nur die beiden Popups (Bericht und Formular) sowie das Applikationsfenster von Interesse sind. Kindfenster des Applikationsfensters sind neben dem MDIClient auch noch eine Statusbar und vier msoCommandBarDock. Letzteres sind Container für die Commandbars, die es ermöglichen, diese an allen vierSeiten des Applikationsfensters anzudocken. Innerhalb des MDIClient schließlich finden sich die sieben MDI-Kindfenster (Endlosformular etc.).

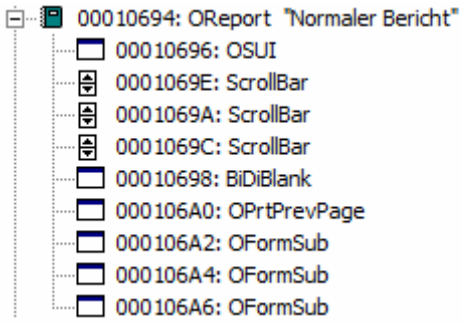
Access-Fenster im Detail:

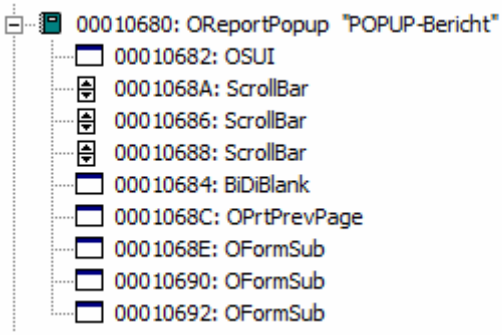


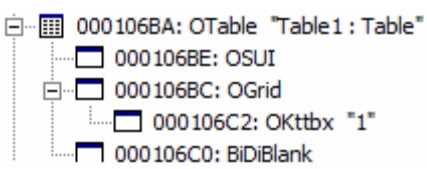


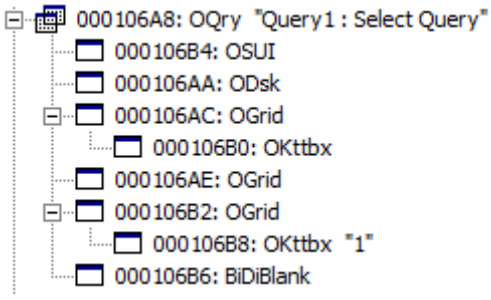












Was ist wichtig?

1. Es besteht seitens der Kindfenster kein Unterschied zwischen Popup, normalen und Endlosformularen (die beim normalen Formular sichtbaren Kindfenster im zweiten OFormSub sind die Steuerelemente, die nur auf diesem Formular eingefügt wurden). Formulare in Datenblattansicht gleichen hingegen eher Tabellen und Abfragen. Ebenfalls kein Unterschied besteht zwischen normalen und Popup-Berichten.

2. OFormSub: Mit Ausnahme von Datenblattansichten verfügen Formulare und Berichte jeweils über drei Kindfenster der Klasse OFormSub. Hierbei handelt es sich um ein Header-, ein Detail - und ein Footer-Fenster, und zwar unabhängig davon, ob diese in der Section-Collection von Access existieren oder nicht. Die Reihenfolge ist dabei immer gleich. (Obwohl auch Berichte diese Fenster besitzen, spielen Sie hier im Prinzip keine Rolle, sie sind in der Berichtsansicht minimiert. Sämtliche Grafische Darstellung werden auf der Klasse "OPrtPreview" ausgegeben). Formulare in Datenblattansicht besitzen anstelle der OFormSubs ein OGrid. Es ist daher auch wenig verwunderlich, warum man bei ihnen in Access keine Header und Footer dazuschalten kann.

3. Steuerelemente im normalen Formular: (i)Steuerlemente werden nie direkt auf dem Formular eingefügt, sondern immer nur auf einem Sektionsfenster. (ii)Viele Steuerelemente finden sich gar nicht in der Fenster-Auflistung wieder. Genauer gesagt gibt es drei verschiedene Gruppen:

   a) Von Access nur gezeichnete Steuerelemente, ohne Handle, kaum Zugriffsmöglichkeiten von außen. Hierzu gehören: Label, Images, Checkboxen, Buttons, Radiobuttons, Groupboxen, Togglebuttons, Frames, Lines, Logos (2007), Hyperlink (2007)und PageNumber & Co (2007)

   b) Fenster mit "temporärem" Handle. Es gibt ein echtes Fenster der Klasse "OKttbx". Dieses Fenster wird als Eingabeelement für mehrere Access-Steuerelemente verwendet, insbesondere innerhalb von OGrids, als Textfeld und als Kombinationsfeld. Bei einem Fokuswechsel wird dieses Fenster einfach unbemerkt an passende Position verschoben.

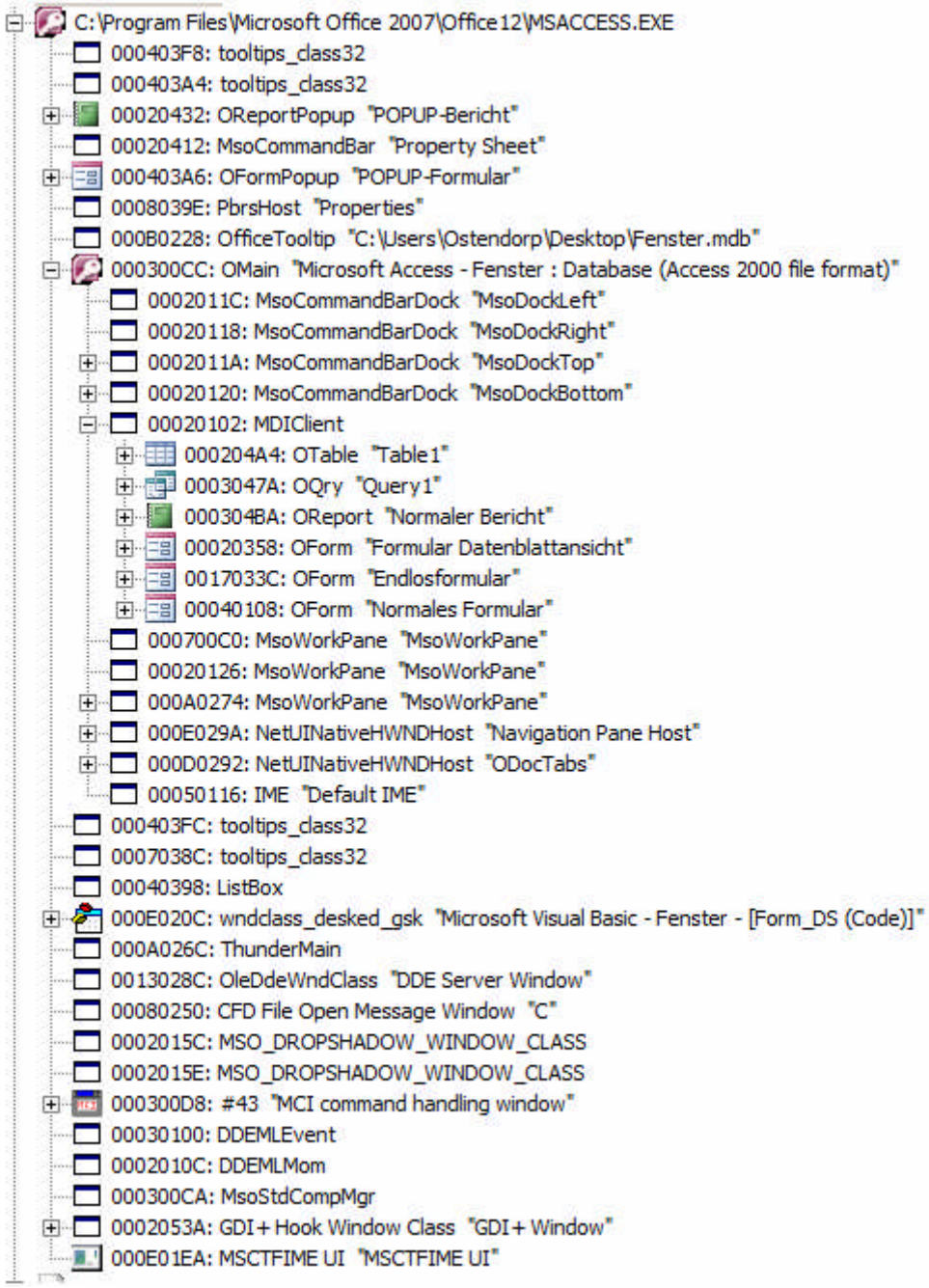
   c) Echte Fenster mit dauerhaftem Handle: Hierzu zählen

      - Subform, Klasse "OFormChild"
      - Listbox, Klasse "OGrid" (dieselbe Klasse wie bei Datenblattansichten!)
      - Register, Klasse "OTabControl„
      - ActiveX-Controls, div.

      Bei Listboxen ist zu beachten, das die Fenster beim Laden ggf. zerstört und neu angelegt werden, was beim Subclassing natürlich berücksichtigt werden muß (vgl. Beispiel "9 Drag and Drop" in der Demo-DB).

4. Am Rande: OSUI ist die Klasse für den Navigationsbereich, Scrollbars sind Scrollbars. (Der Verwendungszweck der anderen Fenster hat sich mir bislang leider noch nicht erschlossen).

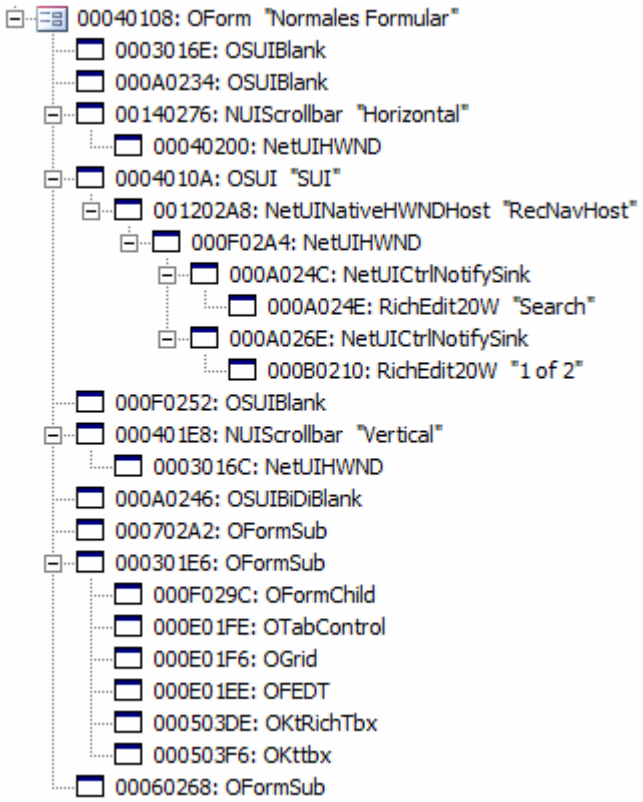
Ein Blick auf Access 2007



Bei Access 2007 hat sich, wie zu erwarten, einiges im Bereich des Applikationsfensters getan. Neben einer Reihe neuer Fensterklasse, wie dem MsoWorkpane und dem NetUINativeHwndHost samt Kindfenstern, sind andere Fenster auch verschwunden. Die gravierendsten Änderungen finden sich wohl sich im Bereich des Datenbankfenster und der Statusbar: Das Datenbankfenster ist nun kein MDIChild mehr sondern in einer etwas verzweigten Architektur innerhalb eines NetUINativeHwndHosts eigefügt (also auf gleicher Ebene wie das MDI-Clientfenster oder die msoCommandbardocks). Die Statusleiste wurde in Form eines CommandBar-Fensters realisiert.

Die Änderungen der Fenster, mit denen man als Access-Programmierer am meisten zu tun hat, fallen hingegen relativ milde aus, insbesondere sind die Klassennamen für die Fenster und Steuerelemente

immer noch dieselben. Größere Änderungen finden sich eigentlich erst auf Ebene der Scrollbars und des Navigationsfensters wieder, siehe nachfolgende Abbildung:

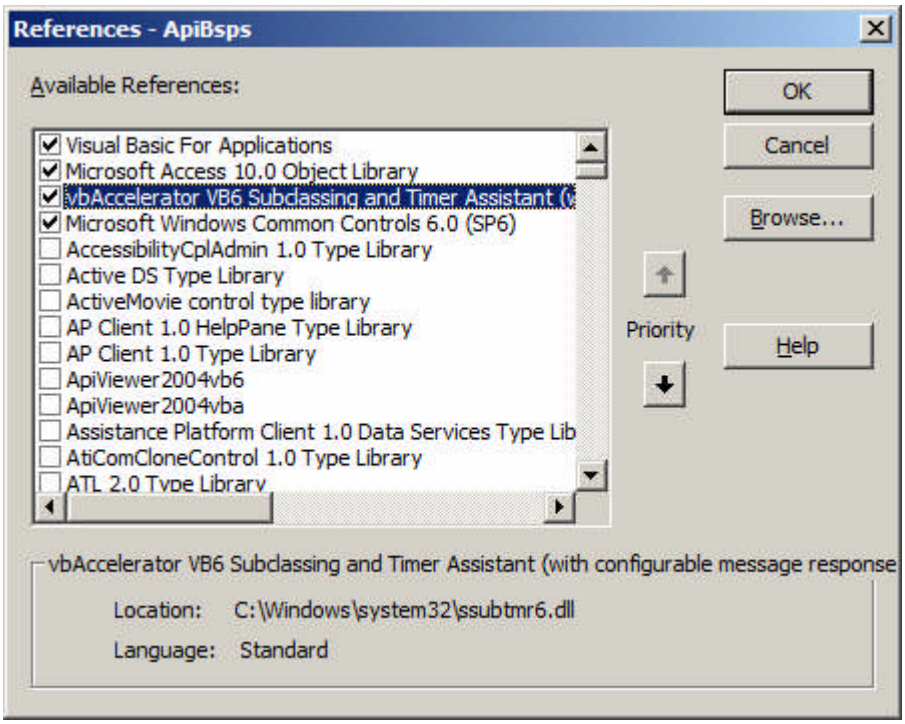


Anmerkung: Auch wenn sich die Fensterarchitektur in wesentlichen Punkten vielleicht nicht großartig geändert hat, gibt es aufgrund des Versuchs von Access 2007, eigene Skins durchzusetzen, trotzdem große Unterschiede im Verhalten zu vorherigen Versionen. So lassen sich z.B. Style-Änderungen im Bereich des Applikationsfensters so gut wie gar nicht mehr durchführen, da Access sofort versucht, seinen eigenen Style wieder herzustellen (vgl. z.B. auch Beispiel 3 in der Demo-DB).

Praxisbeispiele

Angesichts des Funktionsumfangs des WinApi ist in diesem Rahmen kaum mehr als ein Kratzen an der Oberfläche möglich. Für alle, die ein paar tiefergehende Beispiele suchen, möchte ich allen voran die Seiten von Steve McMahon und Randy Birch ans Herz legen (s. Linkliste).

Nachfolgend sind also ein paar einfache Beispiele aufgeführt, die zumindest überwiegend auch einen etwas spezifischeren Bezug zu Access haben. Für die Subclassing-Beispiele ist ein Verweis auf die SSubtmr6.dll erforderlich, für das hierarchische Kombinationsfeld in Beispiel 7 ein Verweis auf die Common Controls Version 6.



Nachfolgend noch ein paar zusätzliche Anmerkungen zu den Beispielen (bitte parallel die Demo-DB anschauen):

1. Handles ermitteln

Um auf Access-Fenster mittels API zugreifen zu können, benötigt man zunächst einmal deren Handles. Für Hauptfenster wie Formulare, Berichte oder das Applikationsfenster stellt Access entsprechende Properties bereit (Me.hwnd, Application.hwndAccessApp), alle anderen Handles darf man selbst herausfinden. Glücklicherweise hält sich der Aufwand hierfür in Grenzen:

Handles normaler Fenster lassen sich am einfachsten mit der Funktion `FindWindowEx` ermitteln Die vier Parameter (s. Deklaration im Code) bedeuten:

hwnd1 - das Handle des Elternfensters, **in** dem gesucht werden soll
hwnd2 - das Handle eines gleichklassigen Fensters **ab** dem gesucht werden soll
lpsz1 - der Klassenname des zu suchenden Fensters
lpsz2 - der Fenstertitel des zu suchenden Fensters

Durch einen mehrmaligen Aufruf der Funktion `FindWindowEx` ist es somit möglich, sich sowohl durch die Fensterhierarchie, als auch durch die Auflistung gleichklassiger Fenster derselben Ebene durchzuhangeln.

Handles von Steuerelementen lassen sich grundsätzlich auf die gleiche Weise abfragen, allerdings geht es meist noch eine Spur einfacher, indem man über die SetFocus-Methode den Fokus auf das Steuerelement setzt und anschließend mit der Funktion `GetFocus`, das Handle des aktiven Fensters ausliest, sprich das des Steuerelementes.

Dabei ist lediglich der Zeitpunkt des Aufrufes zu beachten. Eine Listbox ist im Form_Load-Event u.U. noch nicht erstellt, so daß ein GetFocus nicht das richtige Handle zurückgibt. Umgehen läßt sich dies dadurch, daß man im Load-Event einen Timer aktiviert und das Handle erst im Timer-Event ausliest.

Ein kleines Problem stellen auch Steuerelemente dar, die keine SetFocus-Methode besitzen und mehrfach, aber in keiner definierten Reihenfolge auf dem Formular bzw. einem Sektionsfenster platziert sind, etwa Registersteuerelemente oder ActiveXe. In diesem Fall ist es notwendig, alle diese Steuerelemente in einer Schleife zu durchlaufen und anhand eines weiteren Kriteriums zu überprüfen, ob das Fenster das richtige ist. Idealerweise verfügen die Formulare über eine undokumentierte accHittest-Methode, die, auch wenn ein Formular gescrollt vorliegt, anhand von Bildschirmkoordinaten das passende Objekt zurückgeben. Somit muß man innerhalb der Schleife lediglich noch die Fensterposition des Steuerelementes ermitteln und per accHittest den Objektnamen an dieser Position erfragen. Stimmt dieser mit dem gewünschten Steuerelement überein, so hat man auch bereits dessen Handle (siehe Funktion `GetControl` in der DemoDB).

2. Fensterpositionen

Für das Positionieren von Fenstern gibt es mehrere Möglichkeiten, die wichtigste und potenteste ist wohl die Funktion `SetWindowPos`. In der Demo-DB sind Beispiele für vier häufige Aufgaben rund um diese Funktion enthalten:

1. Formular auf MDI einpassen

Als Alternative zum Maximieren gedacht. Um die Größe anzupassen wird einfach per GetClientRect die Größe des Clientbereichs des MDI-Clientfensters ermittelt und anschließend per SetWindowPos die Position des Formulares festgelegt.

2. Zentrieren der Anwendung auf den Screen

z.B. für Anwendungen, die für eine kleinere Bildschirmauflösung entwickelt wurden, als die des aktuellen Monitors. Als Bezugsfläche für das Zentrieren wird dabei nicht der gesamte Monitor genommen, sondern nur die Desktop-Arbeitsfläche, sprich der Bereich auf dem Monitor ohne die Taskbar, die sich über die Funktion SystemParametersInfo ermitteln läßt.

3. Zentrieren eines normalen Formulares auf den MDI-Clientbereich

Als Alternative zur eingebauten Property "Auto Center" (bei der sich mir noch nicht erschlossen hat, auf welchen Bereich sich das "Center" wohl genau beziehen mag).

4. Zentrieren eines Popup-Formulares auf den MDI-Clientbereich

Ebenfalls als Alternative zur Property "Auto Center" gedacht, bei der Popups auf den Monitor zentriert werden. Ungetestet, sollte aber auch ganz nützlich sein, wenn die Anwendung auf einem zweiten Bildschirm läuft und es Probleme mit der Auto Center Funktion gibt, sprich Popups auf dem ersten Monitor angezeigt werden.

In der Demo-DB sind die Beispiele 2-4 in einer einzigen Funktion "CenterWindow " zusammengefaßt, der die Handles des zu zentrierenden Fensters sowie des Fensters das als Bezugsfläche dienen soll, übergeben werden. Ob es sich beim zu zentrierenden Fenster um ein Popup oder ein Kindfenster handelt wird automatisch mit der Funktion GetParent erkannt.

Die Funktion arbeitet mit dem flag SWP_NOSIZE. Soll z.B. eine Anwendung beim Zentrieren noch auf eine bestimmte Größe gebracht werden, muß das separat durchgeführt werden.

3. Fensterklassen manipulieren

Bevor eine Applikation ein Fenster erstellen kann, muß zunächst eine Fensterklasse registriert werden, von der das Fenster abgeleitet werden kann (z.B. "OForm"). Alternativ kann auch auf bereits von Windows bereitgestellte Klassen zurückgegriffen werden (z.B. ein TreeView oder ein Textfeld). Bei der Registrierung werden Basis-Angaben, wie sich Fenster dieser Klasse verhalten sollen in eine Klassenstruktur angegeben, die dann per Registerclass(Ex) registriert wird.

Von all dem bekommt ein Access-Entwickler natürlich nichts mit, da dies durch Access selbst erledigt wird. Interessant wird es jedoch dadurch, daß sich einige diese Klasseneigenschaften im Nachhinein per SetClassLong ändern lassen. In den Beispielen sind dies die Hintergrundfarbe und der Cursor.

Das Fenster einer TabControl-Klasse beschränkt sich allerdings nur auf eben jenen Bereich der Registerlaschen. Die Fläche darunter, in der man seine Steuerelemente platziert gehört bereits nicht mehr dazu. Hierbei handelt es sich um einen von Access selbst gezeichneten Bereich!

ACHTUNG: Die Änderung der Eigenschaften einer Klasse werden für **alle** abgeleiteten Fenster wirksam. Wenn man also die Hintergrundfarbe eines TabControls ändern, wird diese Änderung auch für die Registersteuerelemente auf anderen Formularen wirksam, sobald diese neugezeichnet werden.

Leider sind nicht alle Farben die man sieht, so einfach änderbar. Bei einem Formular in Datenblattansicht ist das Dunkelgrau z.B. nicht die Hintergrundfarbe der Klasse "OGRID", sondern eine bewußt gemalte Vordergrundfarbe.

4. Mausradbewegung per Peekmessage

Da das Mausrad im VBA_Editor nicht funktioniert, arbeiten viele Entwickler mit Zusatztools wie FreeWheel, welches ein Scrollen per Mausrad ebendort ermöglicht. Dabei sorgt diese Tool jedoch auch dafür, daß z.B. Textfelder problemlos gescrollt werden, was schnell vergessen wird. Wird die Applikation ausgeliefert, erfolgt auf anderen Rechnern u.U. jedoch ein Datensatzwechsel oder es passiert gar nichts, wenn z.B. Tools eingesetzt werden, die eben dies verhindern sollen (z.B. die MouseHook.dll von Stephen Lebans).

Um ein Textfeld dennoch dazu zu bewegen, sich scrollen zu lassen, kann man sogar ohne Subclassing auskommen, indem man mit PeekMessage arbeitet. Dabei wird (z.B. beim Fokus-Erhalt) ein PeekMessage-Loop aktiviert der spezifisch auf eine WM_MOUSEWHEEL-Nachricht wartet und diese in eine WM_VSCROLL-Nachricht übersetzt. Dadurch wird das Textfeld wie gewünscht gescrollt.

5. Platz sparen mit Kontextmenüs

Die Verwendung von Windows-Kontextmenüs in Access erscheint auf den ersten Blick vielleicht etwas abstrus. Schließlich lassen sich Kontextmenüs von Access sogar per GUI zusammenklicken. Für bestimmte Aufgaben finde ich sie aber extrem nützlich. Dazu gehört z.B., "überladene" Formulare,

bei denen im Lauf der Zeit immer mal wieder ein zusätzlicher Button dazugekommen ist, wieder etwas zu bereinigen. Durch ein Kontextmenü hinter einem Button, läßt sich einerseits viel Platz sparen, andererseits kann man sie prima räumlich bestimmten Steuerelementgruppen zuordnen, Einträge je nach Bedarf (z.B. Nutzerberechtigungen) aktivieren und deaktivieren und vor allem später gut ergänzen, wenn weitere Funktionalitäten hinzukommen sollen. Access-Kontextmenüs sind hierfür ungeeignet, da sie sich nicht (schlecht) entsprechend positionieren lassen.

Um ein Kontextmenü anzuzeigen, wird zunächst ein leeres Menüfenster per CreatePopupMenu erzeugt, per AppendMenu gefüllt und anschließend per TrackPopupMenu(Ex) sichtbar gemacht. Um zu ermitteln, welcher Eintrag angeklickt wurde, wird per GetMessage auf eine eingehende Nachricht gewartet. Handelt es sich um eine WM_COMMAND, erhält man im wparam die Item-Nummer und kann entsprechend reagieren.

Verschiedene Stile lassen sich einerseits durch die Flags beim AppendMenu-Aufruf einstellen (MF_POPUP, MF_GRAYED, MF_CHECKED etc.) andererseits durch weitere Funktionen wie SetMenuDefaultItem oder SetMenuItemBitmaps.

Bei der Positionierung (vgl. Trackpopupmenu) ist für Buttons, die im Detailbereich angelegt werden, der Scrollzustand zu beachten. Hierfür kann z.B. ein Hilfsfenster am Ursprung des Detailbereichs, also bei der Position 0/0 , ein unsichtbares Static-Fenster per CreateWindowEx angelegt werden. Wird der Detailbereich gescrollt, verschiebt sich neben dem Button auch das Fenster entsprechend nach oben. Die Positionierung des Kontextmenüs kann, unter Berücksichtigung der absoluten Position des Hilfsfensters auf dem Monitor, also immer relativ zu diesem erfolgen. (Für den Header und den Footer können hingegen direkt die Sektionsfenster verwendet werden).

6. Splitter

Die Möglichkeit, benachbarte Fenster durch einen Splitter zu verschieben, gehört eigentlich zum Standard eines jeden Programmes. Leider stellt Access hierfür keinerlei eingebaute Funktonalitäten bereit. Die einfachste Möglichkeit einen Splitter zu programmieren wäre ein Steuerelement, z.B. ein Frame, das beim MouseMove entsprechend verschoben wird. Gerade bei Formularen, in denen Grafiken oder komplexe Abfragen angezeigt werden, ist der Zeitaufwand für die Neuberechnung bei einem direkten Verschieben allerdings sehr hoch, was zu Flackern am Bildschirm und einfach zu einer insgesamt sehr trägen Bewegung des Splitters führt. Mit Access-Bordmitteln, läßt sich jedoch kein indirekter Splitter Programmieren, bei dem z.B. ein schwarzer Balken über andere Fenster wie Ufos, Listboxen oder ActiveXe bewegt wird.

Der Splitter in der Demo-DB ist so angelegt, daß zwischen einem direkten und indirekten Splitten gewählt werden kann (letztlich per Optionseinstellung durch den Nutzer). Im Load-Event wird eine entsprechende Splitter-Klasse (clsSplitter) instanziert, die ihrerseits zwei Events auslöst, die im Formular dazu genutzt werden können, die Steuerelemente neu zu positionieren: ein Event direkt nach jeder Mausbewegung, ein weiteres erst nach erfolgtem Mouseup.

Für das eigentliche Zeichnen des Splitterbalkens werden zwei Methoden verwendet, einmal InvertRect zum anderen PatBlt mit einem PatInvert-flag und unterschiedlichen Pinseln (solid und pattern). Bei beiden handelt es sich um XOR-Funktionen, d.h. beim ersten Aufruf wird der Splitter gezeichnet, beim zweiten Aufruf an der gleichen Position wird der Splitter gelöscht. Im Gegensatz zum Zeichnen eines soliden Pinsels und dem anschließenden Neuzeichnen des ursprüngliches Fensterbereichs, minimiert dies ein störendes Bildschirmflackern ganz enorm.

7. De/Activeate-Event auch für Popup-Formulare

Die Activate und Deactivate-Events von Access werden nur für normale Formulare gefeuert. Es gibt also keine reguläre Möglichkeit, den Wechsel zu und von einem Popupformular mitzubekommen.

Leider bleiben bei Popupformularen die Standardschaltflächen der Commandbars für ein eventuell im Hintergrund liegende, normales Formular aktiv. Dies kann dazu führen, daß Nutzer unbeabsichtigt Datensätze aus diesem Formular anstatt aus dem Popup-Formular löschen (und weil das Löschen scheinbar nicht funktioniert hat, versucht man es gleich noch einmal...)

Mit der Auswertung der WM_NCACTIVATE-Nachricht (NC steht übrigens für Nonclient), ist es möglich, ein entsprechendes Event auch für Popup-Formulare auszulösen und dabei z.B. Commandbars zu deaktivieren.

Ein weiterer Vorteil liegt darin, daß ein Deactivate-Event zum automatischen Schließen verwendet werden kann. Ein Beispiel, sich dies zu Nutzte zu machen ist der Nachbau eines hierarchischen Kombinationsfeldes. Hierfür wird in der Demo-DB ein rahmenloses Popup-Formular mit einem TreeView verwendet, daß bei Betätigung eines DropDown-Buttons einfach unterhalb eines entsprechenden Textfeldes platziert wird. Mit dem Deactivate-Event wird es nun möglich, das Formular automatisch zu schließen, sobald der Nutzer mit der Maus woanders hinklickt.

8. Größenänderungen von Formularen abfangen

Für die Gestaltung bildschirmunabhängiger Anwendungen ist es notwendig, einen Mechanismus zu entwickeln, wie Steuerelemente und Formulare bei Größenänderungen reagieren sollen. Im einfachsten (und nicht zu empfehlendem) Fall werden die Formulare proportional angepaßt, normalerweise erfolgt die Anpassung jedoch nach spezifischeren Regeln. Dabei steht man häufig vor dem Problem, daß das Resizen innerhalb eines gewissen Größenbereichs gut funktioniert, jenseits davon jedoch nicht (insbesondere bei Steuerelementen auf Registern). Eine Möglichkeit, dem entgegen zu wirken, besteht darin, ab einer bestimmten Größe nicht weiter zu resizen und stattdessen Scrollbars einzuschalten, was optisch jedoch häufig störend wirkt. Eine andere Möglichkeit ist es, die Größenänderung eines Formulares von vorn herein nur bis zu einem bestimmten Punkt zuzulassen. Da Access mit dem Resize-Event leider keinen Cancel-Parameter mitliefert, mit dem die Größenänderung abgebrochen werden könnte, ein nachträgliches Ändern der Formulargröße zudem zu erheblichem Flackern führt und abgesehen davon eben immer erst NACH

einer schon erfolgten Größenänderung ausgelöst wird, muß wiederum per Subclassing die passende Windows-Nachricht ausgewertet werden.

Hierbei handelt es sich um die WM_GETMINMAXINFO-Nachricht, die ein Formular vor der eigentlichen Größenänderung erhält. Die Nachricht beinhaltet einen Pointer auf eine GETMINMAXINFO-Struktur, in der u.a. die Werte für eine minimale Größe festgelegt werden können.

Da diese Angaben bei einem maximierten Zustand eines MDIChild-Formulares ignoriert werden, muß auch das Applikationsfenster überwacht und in die Berechnung miteinbezogen werden.

9. Drag and Drop

Bei diesem Beispiel handelt es sich um eine Adaption des Beispiels von Dev Ashish (http://www.mvps.org/access/api/api0032.htm), bei dem eine Listbox mittels der Funktion DragAcceptFiles dazu verführt wird, sich mit fremden Dateien einzulassen. Durch Überwachung der WM_DROPFILES wird ein eventuelles "Drop" registriert. Mit der Funktion DragQueryFile können die Anzahl und Namen der Dateien ermittelt und als RowSource-String an die Listbox übergeben werden.

Wenn man keine Scheu vor den Common Controls hat, ist es natürlich wesentlich einfacher, deren OLEDragDrop-Events mit dem Data-Object zu nutzen.

10. Accessibility

Abgesehen von der Ermittlung der Cursorposition hat dieses Beispiel zwar vordergründig wenig mit API zu tun, es verwendet jedoch die undokumentierte **accHittest**-Funktion, die verdeutlicht, daß Access das Accessibility-Konzept als API-Bestandteil implementiert.

Das übliche Vorgehen, um z.B. ein MouseOver über ein Steuerelement zu simulieren, ist es, beim MouseMove etwas bestimmtes zu tun, und es im MouseMove-Event des zugehörigen Sektionsfensters wieder zu lassen. Liegen die Steuerelemente eng beieinander, kann es dabei jedoch passieren, daß das MousMove-Event des Sektionsfensters übersprungen wird und Steuerelemente z.B. eingefärbt bleiben. Ein weiterer Nachteil ist, daß der gleiche Code für sämtliche Steuerelemente eingefügt werden muß.

Im Beispiel der Demo-DB erfolgt all dies mithilfe der Funktion accHittest in einer zentralen Klasse, die nur einmal zum Anwendungsstart instanziert werden muß. Der Vorteil liegt also einerseits in der Vermeidung von Code, andererseit in der Durchsetzung eines Designkonzeptes. Der Nachteil in der gegenwärtigen Implementierung ist die Verwendung eines Timers zur Abfrage der Mausposition (Flackern). Hierfür wäre zwar ein Mousehook (SetWindowsHookEx) leicht zu realisieren, leider führt dies jedoch zu ähnlichen Komplikationen, wie das Subclassen auf herkömmlichem Weg.

11. SHFileOperation

Warum soll man sich "API antun", um Dateioperationen durchzuführen? Zum einen gibt es doch genügend eingebaute Methoden wie FileCopy, Kill, Name etc. zum anderen auch noch das bequem zu programmierende FileSystemObject. Ganz einfach: Mehr Möglichkeiten, mehr Komfort und unterm Strich sogar weniger Aufwand.

SHFileOperation ist die Funktion, die auch der Windows-Explorer nutzt. Mit ihr ist es z.B. möglich, ganze Order zu kopieren oder sie in den Papierkorb zu verschieben anstatt ganz zu löschen, mit einem einzigen Flag die Anzeige von Rückfragen und Fehlermeldungen zu bestimmen und vor allem **betriebssystemspezifische** Progressbars anzuzeigen, die dem Nutzer zudem auch noch die Möglichkeit bieten, länger dauernde Vorgänge abbrechen zu können.

Links

A) Ressourcen und Tools

Accessibility-Tools
http://www.microsoft.com/downloads/details.aspx?FamilyID=3755582a-a707-460a-bf21-1373316e13f0&DisplayLang=en
ApiGuide
http://allapi.mentalis.org/agnet/apiguide.shtml
ApiViewer 2004
http://www.activevb.de/rubriken/apiviewer/index-apiviewer.html
ControlSpy
http://www.microsoft.com/downloads/details.aspx?familyid=19d4294d-0531-4ec2-8b27-2e463b315f16&displaylang=en
MSDN
http://msdn2.microsoft.com/de-de/default.aspx
SSubTmr6.dll
http://www.vbaccelerator.com/home/VB/Code/Libraries/Subclassing/SSubTimer/VB6_SSubTmr_Binary.asp
VB5-Runtime
http://support.microsoft.com/kb/q180071/
Windows Vista SDK
http://www.microsoft.com/downloads/details.aspx?FamilyID=c2b1e300-f358-4523-b479-f53d234cdccf&DisplayLang=en
Winspector
http://www.windows-spy.com/

Von den hunderten guter Webseiten rund um Access, VB und API sind dies diejenigen, die man in meinen Augen mindestens besuchen sollte:

B) API + Access:

Dev Ashish
http://mvps.org/access/api/index.html
Beispiele in der KnowHow.mdb
http://freeaccess.de/knowhow.asp
Stephen Lebans
http://lebans.com/

C) API + VB:

ActiveVB

http://www.activevb.de/

Hardcore Visual Basic

http://vb.mvps.org/hardcore/

Das Buch von Bruce McKinney als Online-Version!

Randy Birch

http://vbnet.mvps.org/index.html

Brad Martinez

http://btmtz.mvps.org/

vbAccelerator (Steve McMahon)

http://www.vbaccelerator.com/home/index.asp