

MV_MENU-TUTORIAL

Diese Anleitung zeigt, wie Sie die mv_Menu Klasse in dBASE verwenden können. Mv_Menu ist eine dBASE benutzerspezifische Klasse, mit der Sie ein Menü für Ihre dBASE-Forms erzeugen können.

Sie fragen sich, warum eine neue Menü-Klasse? dBASE besitzt eine eigene Menü-Klasse die mittels des Designers visuell erzeugt werden kann.

Aber MV_MENU hat einige entscheidende Vorteile gegenüber dem herkömmlichen dBASE-Menü.

- Sie können vor jeden Menübegriff ein Icon oder ein bmp-Bild setzen.
- Das Menü kann unten, oben, links oder rechts des dBASE-Forms angedockt werden.
- Nicht nur SDI Windows können ein Menü besitzen, auch alle MDI-Forms können ihr eigenes benutzerspezifisches Menü haben.
- Ein grosser Vorteil ist, dass Sie Ihrem Menü das Aussehen von Office2003, Office2007 als auch von Vista (benutzen Sie die Ribbon Option) geben können.

Allerdings gibt es einen Nachteil, den ich nicht verschweigen möchte : Sie müssen das Menü von Hand programmieren Es ist leider kein Designer dafür verfügbar....

Lesen Sie weiter um zu lernen, wie Sie ein modernes Menü erzeugen können.

A) Das Registrieren des ocx

Das Arbeitspferd des mv_Menu ist ein Activex-Objekt. Wie jedes andere COM-Objekt müssen Sie auch dieses registrieren bevor es verwendet werden kann. Dies muss auf Ihrer Entwicklungsmaschine und auch auf jedem Client getan werden.

Dazu benötigen Sie das Systemprogramm regsvr32 mit dem Pfad auf die ocx Datei als Parameter.

- regsvr32 c:\vdblogic\com\mv_menu.ocx

Ersetzen Sie "c:\vdblogic\com\" durch den Pfad auf dem Ihre ocx-Datei liegt. Wenn der Pfadname Leerschläge enthält schliessen Sie es mit Anführungszeichen ein z.B.

- regsvr32 "c:\Program Files\com\mv_menu.ocx"

Bitte beziehen Sie dies in Ihr Installationsprogramm (Inno, Installshield...) mit ein, damit die Registrierung automatisch bei der Installation Ihrer Applikation durchgeführt wird.

B) Einstieg

Der dBASE Teil der Komponente ist eine benutzerspezifische Klasse (geliefert als mv_Menu.co), der alle Klassen enthält, um ein fortschrittliches, modern ausschauendes Menü zu entwickeln. Die Basis der Hierarchie ist die mv_Menu Klasse selbst: Es ist der Behälter, der die Menübar an der Oberseite Ihrer Form enthält. Am Anfang ist diese Menübar völlig leer und wir werden zusammen diese Menübar mit Items, Sub-items und Sub- Sub-Items füllen.

Wir wollen zusammen nach und nach unseren Startpunkt bauen: Öffnen Sie den Source-Editor und tippen Sie den folgenden Code ein:

```
#include mv_menu.h
Class own_menu(oparent) of mv_menu(oparent) custom

Endclass
```

Gut, das ist nicht viel, aber wir haben gerade eine Unterklasse von unserem Hauptmenü Objekt erstellt. Jeder dBASE-Guru wird mir Recht geben, dass es eine gute Idee ist, die Originalsteuerelemente nicht zu verwenden sondern eher abgeleitete Controls zu nutzen. Wir sind also nur den allgemein anerkannten Regeln gefolgt. Die include Anweisung wird uns helfen, mit einigen vorbestimmten Konstanten den Code leichter lesbar zu machen.

Beachten Sie die Verwendung des "Custom" Deskriptors, dies wird verhindern, dass der Designer jedes Mal die Eigenschaften unseres Menüs verändert sobald wir etwas an unserer Form ändern.

Natürlich wollen wir dies auch in Natura sehen und darum werden wir als nächsten Schritt unser Menü in eine Form einbauen.

Schliessen Sie den Quellcode-Editor und speichern Sie den Code als own.cc. Wählen Sie den Reiter "Andere" im dBASE Navigator, Klicken Sie auf die Datei mv_menu.co und drücken sie <F2>, machen Sie mit own.cc dasselbe. Alternativ können Sie "set procedure <filename.ext>" ins Befehl-Fenster tippen und die ENTER Taste drücken. (Je nach Ihren Einstellungen sind Dateien mit der Endung *.co im dBASE Navigator eventuell nicht zu sehen). Sie haben jetzt gerade die Komponente in den Geltungsbereich des Form-Editors gebracht.

Starten Sie nun den Formular-Designer und erzeugen Sie ein neues Formular. Drücken Sie "F12" und Sie sehen etwa folgendes im Quellcode-Editor:

```
class NeuForm of FORM
  with (this)
    height = 16
    left = 38
    top = 1
    width = 40
    text = ""
  endwith
endclass
```

Gehen Sie zum Kopf des Codes und fügen Sie über der Zeile " ** END HEADER -- Diese Zeile nicht entfernen" folgendes hinzu...

```
Set procedure to mv_Menu. co additive
Set procedure to own.cc additive
```

Setzen Sie zwischen die Zeilen "endwith" und "endclass" folgendes ein:

```
This.my_menu = new own_menu(this)
```

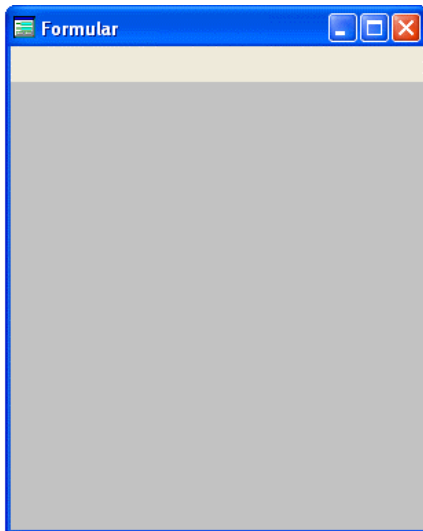
Wir haben ein Objekt der Klasse own_menu realisiert und verweisen mit der Eigenschaft "this" auf das Objekt "my_menu". Wir sind im Constructor der Form, so ist die Eigenschaft "this" nichts anderes als die Form selbst. Mit dieser Ziele haben wir das Menü erzeugt und es der Form zugeordnet.

Bemerkung: Der dem New Operator übergebene Parameter muss ein Formobjekt sein. Wenn Sie versuchen das Menü etwas anderem zuzuordnen (Entryfield oder Pushbutton) werden Sie eine Fehlermeldung erhalten

Bemerkung: Sie können kein Menü Control von der Teilpalette ziehen. Die meisten seiner Eigenschaften sind verborgen und die "streaming engine" wäre verwirrt, wenn Sie es trotzdem versuchen würden. Sie müssen das Menü im Quellcode hinzufügen

Bemerkung: Das "onopen" Event der Form ist zu spät, um das Menü zuzuordnen, dies muss im Constructor erfolgen.

Kehren Sie zurück zur Designer-Ansicht (F12), und was Sie sehen... ist gar nichts. Die ganze Zeichnungsarbeit wird zur Laufzeit durchgeführt und ist während des Designs unsichtbar. OK, dann wollen wir die Form starten. Speichern und starten Sie das Formular als test.wfm. Was für ein Mist, nichts. Gut, eigentlich ist etwas passiert, aber Sie können es nicht sehen. Öffnen Sie das Formular im Designer und ändern Sie die Colornormal Eigenschaft der Form auf "Silber" und starten das Formular nochmals. Jetzt sollten Sie folgendes sehen:



Hier können wir unsere leere Menübar sehen. Nichts Spezielles, ausser dass es an ein MDI-Formular gebunden wird. Wenn Sie es nützlich finden, könnten Sie leicht zwei Mdi-Formulare gleichzeitig, mit zwei verschiedenen Menübars öffnen (zum Beispiel von einigen im Formular gezeigten Datenobjekten abhängig). Diese Möglichkeit haben Sie mit dem herkömmlichen Menü-Objekt nicht.

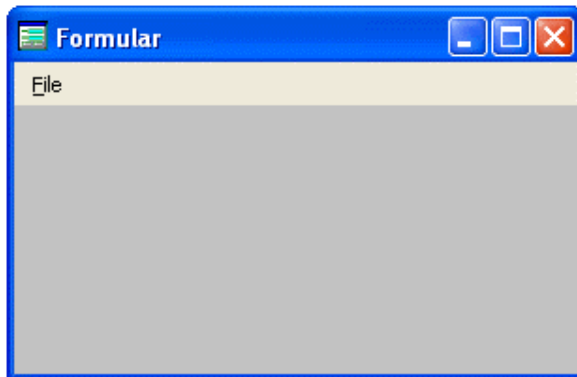
Im folgenden Schritt wollen wir das mv_menu ein wenig aufpeppen.

C) Hinzufügen von Items zur menu_bar

Wir wollen zwei verschiedene Objekte in das mv_Menu einbauen: das Menü-Item-Objekt und das Popup-Item-Objekt. Das Menü-Item-Objekt wird als ein Container für weitere Objekte verwendet (Menü-Item oder Popup-Item), und das Popup-Item-Objekt wird verwendet, um eine Funktion zu starten, wenn es angeklickt wird. Das Menü-Item öffnet Untermenüs, wenn darauf geklickt wird, hat aber keinen getrennten programmierbaren EventHandler. Öffnen Sie die own.cc Datei im Quellcode-Editor und ergänzen Sie den Code wie folgt:

```
#include mv_menu.h
Class own_menu(oparent) of mv_menu(oparent) custom
    Function initialize
        This.m_item1 = new menuitem(this, "&File")
        return
Endclass
```

Schliessen Sie die own.cc Datei und starten Sie Ihre test.wfm wieder. AHA, jetzt sehen wir einen Fortschritt:

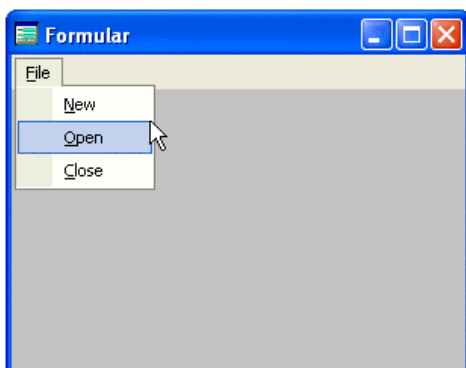


Wir haben zur own_menu Klasse eine Methode mit dem Namen "initialize" hinzugefügt. Diese Methode ist sehr wichtig, da sie beim Start des Menüs automatisch aufgerufen wird (es ist eine Art onopen EventHandler für die Menübar). Mit dieser Methode, können wir den ganzen Aufbau-Code für alle Objekte und Unterobjekte schreiben, wie: `this.m_item1 = new menuitem (this, "&File")`

Die Syntax folgt genau den Regeln der normalen dBASE OOP Sprache, der "New Operator" wird ein neues Objekt der Klasse menuitem instanziiert, dessen Parent "this" ist und dessen Text, "File" lautet. Mit "this" ist die Menübar gemeint und das MenuItem-Objekt ist jetzt ein Kind der Menübar. Wie gesagt, das menuitem selbst ist ebenfalls ein Behälter und kann andere Objekte enthalten; wir werden einige Popuitem-Objekte hinzufügen. Fügen Sie den folgenden Code in der initialisiertest Methode der own_menu Klasse hinzu

```
This.m_item1.p1 = new popuitem(this.m_item1, "&New")
This.m_item1.p2 = new popuitem(this.m_item1, "&Open")
This.m_item1.p3 = new popuitem(this.m_item1, "&Close")
```

Starten Sie das Formular erneut und Sie werden sehen, dass wir uns langsam dem Aussehen eines modernen Menüs nähern.



Als nächstes werden wir auf die Vorteile des "Objekt orientierten Programmierens" schauen.

D) Wie steht's mit der Vererbung:

Im vorausgehenden Kapitel lernten wir, wie wir unsere Menübar bestücken können. Ein Weg, ein Menü für eine Anwendung zu bauen, wäre, alle notwendigen Objekte in die *.cc zu legen, und mit New() zu erzeugen. Der Nachteil dieser Methode ist, dass Ihr Menü nur für diese eine Anwendung zu gebrauchen und somit einmalig wäre. Es wäre schwierig, den Code (Menü) für eine andere Anwendung zu verwenden.

Glücklicherweise ist dBASE eine echte Objekt orientierte Programmiersprache und wir werden diese Fähigkeiten nutzen um uns das Leben leichter zu machen.

Wir werden eine neue Klasse hinzufügen, die von unserem own_menu erbt. Öffnen Sie die own.cc Datei im Quelltext-Editor und fügen Sie nach dem "endclass" Befehl folgenden Text hinzu:

```
Class next_menu(oparent) of own_menu (oparent) custom
  Function initialize
  Super::initialize()
  This.m_item2 = new menuitem(this, "&Edit")
  return
Endclass
```

Hier erzeugen wir eine Unter-Klasse next_menu von own_menu, das bedeutet, dass aller Code und alle Objekte unseres own_menu jetzt ein Teil des next_menu sind, ausserdem können wir leicht weitere Funktionalität zu diesem next_menu hinzufügen.

Die Zeile: Super::initialize(), stellt sicher, dass der Initialisierungscode des Vaters richtig durchgeführt wird. Jedes Mal, wenn Sie solche eine Art von Vererbungs-Kette bilden, müssen Sie sicherstellen, dass alle Eltern und Grosseltern die Chance haben, richtig initialisiert zu werden, daher der Aufruf von Super::initialize(). Der nächste Schritt fügt ein weiteres Menü Objekt mit der Überschrift "Edit" zur Menübar hinzu.

Bemerkung: Das Und-Zeichen in Verbindung mit der Grundmenü Klasse. Wird einem Buchstaben das "&" Zeichen vorangestellt, erscheint dieser mit einem Unterstrich, was bedeutet, dass diese Option mit dem ALT-key aufgerufen werden kann.

Bevor wir uns das Menü im Formular anschauen, fügen wir noch folgende Zeilen hinzu:

```
This.m_item2.m1 = new menuitem(this.m_item2, "&Action")
This.m_item2.m1.p1 = new popupitem(This.m_item2.m1, "&Copy")
This.m_item2.m1.p2 = new popupitem(This.m_item2.m1, "&Paste")

This.m_item2.Cancel = new popupitem(This.m_item2, "Ca&ncel")
```

Schliessen Sie die Datei und starten Sie das Formular. Hmm, nichts hat sich geändert. Sie müssen die Klasse im Formular ebenfalls ändern: MY_MENU = New next_MENU (this)

Wenn Sie das Formular jetzt starten, werden Sie das aktualisierte Menü sehen. Damit ist jetzt auch klar, was es bedeutet, dass ein menuitem ein Container für andere Objekt ist. Das Objekt m_item2.m1 enthält zwei popupitems als Untermenüs. Es ist wichtig zu wissen, dass die Hauptmenübar nur menuitems enthalten kann. Ich überlasse es Ihnen, noch einige Objekte zum Menü (vielleicht "Werkzeuge" und ein "Hilfe" Menü) hinzuzufügen. Jetzt wo wir etwas haben, mit dem wir arbeiten können, werden wir einen näheren Blick auf die Eigenschaften und Methoden des mv_Menu werfen.

E) Die Eigenschaften und Methoden der Menübar

1) Die Theme Eigenschaften und die Set Theme Methode:

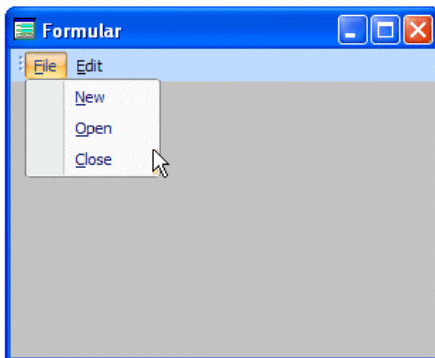
Mit dem mv_Menu haben Sie die Wahl zwischen sieben verschiedenen Stilen. Diese "Themen" ahmen Aussehen und Verhalten der Menüs der bekannten Office Pakete oder das neu eingeführte gläserne Aussehen von VISTA nach. In der Include Datei finden wir folgende theme Konstanten:

```
#define      theme_2000      0      // Office 2000
#define      theme_XP        1      // Office XP
#define      theme_2003      2      // Office 2003
#define      theme_WinXP     3      // Native WinXP
#define      theme_Whidbey   4      // Whidbey
#define      theme_2007      5      // Office 2007
#define      theme_Ribbon    6      // Vista-Look
```

Wenn Sie die Set_Theme Methode mit einer dieser Konstanten anrufen, wird sich das Menü im Aussehen entsprechend des ausgewählten Theme sofort ändern. Die Theme-Eigenschaft beinhaltet einen numerischen Wert zwischen 0 und 6, der dem zurzeit ausgewählte Thema entspricht.

Fügen Sie den folgenden Code am Ende der initialize Methode der next_menu Klasse in own.cc hinzu und sehen Sie das Ergebnis in Ihrem testform...

```
This.Set_Theme ( theme_ribbon )
```



2) Die attach_to framewin Eigenschaft

Diese Eigenschaft bindet das Menü ans _app.Framewin Objekt. Sein Default Wert ist false, was bedeutet, dass das Menü nicht ans framewin gebunden wird. Wenn Sie es auf true setzen, wird das Menü an der Oberseite vom framewin erscheinen. Jedoch braucht man noch ein normales Formular, das als Parent die Kontrolle hat. Dieses "Parent Formular" ist für die Lebenszeit des Menüs verantwortlich.

Wenn Sie dieses Formular schliessen, werden die mv_menu freigegeben und durch die üblichen dbase-Menüs ersetzt. Die Form-Eigenschaft des Objektes wird einen Verweis auf dieses Parent Formular enthalten.

```
This.attach_to_framewin = true
```

3) Die Set Docking Methode

Diese Methode erwartet zwei Parameter. Der erste ist ein logischer Wert, der anzeigt, ob die Menübar an eine Seite des Parent Window andockt werden kann. Wenn dieser Wert falsch ist, nimmt das Menü seine klassische Position an der Oberseite des Windows ein und kann nicht bewegt werden (in diesem Fall wird der zweite Parameter ignoriert). Wenn der übergebene Wert true ist, bestimmt der zweite (numerische) Parameter, wo die Menübar andockt werden kann. Die folgenden Konstanten werden in der include Datei definiert:

```
#define Dock_Top 0x1
#define Dock_Bottom 0x2
#define Dock_Left 0x4
#define Dock_Right 0x8
#define Dock_Float 0x10
#define Dock_All 0x1F
```

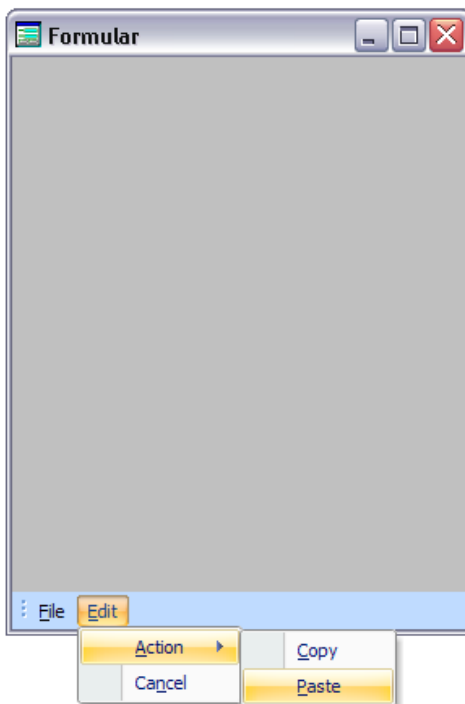
Wenn Sie zum Beispiel wollen, dass das Menü oben und unten an das Formular andockt werden kann, rufen Sie die Set_Docking Methode wie folgt auf:

```
This.Set_Docking(true, Dock_top + Dock_bottom )
```

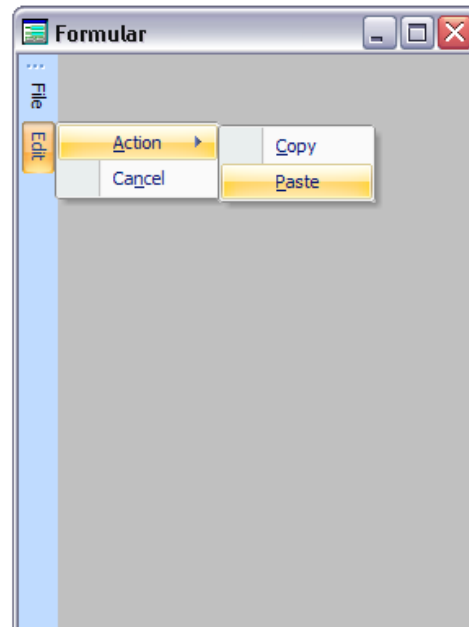
Wenn Sie zum Beispiel wollen, dass das Menü links und rechts an das Formular andockt werden kann, rufen Sie die Set_Docking Methode wie folgt auf:

```
This.Set_Docking(true, Dock_left + Dock_right)
```

Unten andockt



Links andockt



4) Die Get MenuHeight Methode

Manchmal ist es vorteilhaft, wenn man die exakte Höhe des Menus kennt (z.Bsp. wenn man berechnen möchte, wie viel Platz im Formular für andere Controls bleibt) Die `Get_MenuHeight`-Methode liefert diese Höhe in Pixels.

5) Die Add Toolbar Methode

Ein Teil des `mv_Menu` ist eine moderne Toolbar. Diese Toolbar hat ebenfalls eine ansehnliche Menge von Eigenschaften, Methoden und Events, wir werden diese später in dieser Anleitung noch kennenlernen.

Die Verbindung zwischen dem `mv_Menu` und der Toolbar wird durch obige Methode `add_Toolbar` hergestellt. Eine Toolbar kann nicht ohne ein `mv_Menu` erzeugt werden, Sie müssen erst immer ein Menu erstellen, an das die Toolbar dann gebunden wird (natürlich könnte dieses Menu völlig leer sein). `Add_toolbar` erwartet 2 Parameter : einen String, der den Namen der Toolbar enthält, sowie einen numerischen Wert, der die Startposition der Toolbar angibt. (siehe auch die `Set_Pos`-Methode der Toolbar)

```
This.t_bar = this.add_toolbar("File" , 0)
```

`This.t_bar` enthält nun eine Referenz auf die Toolbar und kann benutzt werden, um weitere Controls hinzuzufügen.

6) Die Set Visible Methode

Es kann vorkommen, dass Sie kein Menu in ihrem Formular haben möchte, aber trotzdem eine Toolbar brauchen. Für diese Fälle müssen wir die Toolbar verstecken können. Dies wird mit der `Set_Visible` Methode erreicht. Der Default-Wert ist `true` ; falls sie `set_visible(false)` aufrufen, wird die Menubar versteckt und die Toolbar wird das oberste Control im Form.

```
This.Set_Visible(false)
```

Zu dieser Methode gehört die `m_visible`-Eigenschaft. (kann nur gelesen werden und gibt an, ob die Menubar gerade zu sehen ist (`true`) oder nicht(`false`))

7) Die Set Iconsize Methode

Eine spezielle Eigenschaft der Toolbar ist, dass sie Icons enthalten kann, die nicht auf die 16x16 Pixel Größe der Original-Toolbar beschränkt sind.

Die Größe der Pixel wird vom Menu gesteuert und ist global gültig, das heisst eine Toolbar kann nicht Icons der Größe 16 und 24 gleichzeitig enthalten. Moderne Icons können Bilder für verschiedene Auflösungen enthalten. Die Toolbar wählt automatisch jene aus, die am besten zur aktuellen Icon-Größe passen. Falls nur eine Bildgröße vorliegt, dann erweitert bzw. reduziert die Toolbar die Bilder auf die passende Größe (was aber mit einem Qualitätsverlust verbunden ist)

```
This.Set_Iconsize(24)
```

8) Die Set State and Get State Methoden

Da Sie oder der User die Toolbar bzw. Das Menu an verschiedenen Stellen des Formulars andocken, oder frei schweben lassen können, brauchen wir eine Möglichkeit diese Positionen zu speichern. Auf diese Weise können wir dafür sorgen, dass der User seine Formulare immer wieder so vorfindet, wie er sie verlassen hat.

Dies wird mit der Get_State und Set_State- Methoden erreicht.

Get_State liefert einen String im XML-Format zurück, der die Positionsinformationen des mv_Menu enthält. Dieser String könnte in eine INI-Datei geschrieben werden oder, falls Sie viele Menus und Toolbars nutzen, als Memo-Feld in eine Tabelle. Sobald das Formular geöffnet wird, kann die Set_State –Methode mit dem XML-String aufgerufen werden und die Elemente befinden sich danach an der Position, an der sie waren als der User das Fenster geschlossen hat.

Typischerweise würden sie folgenden Code ins onopen des Formulars aufnehmen

```
Function OnOpen
  Local initpos,q
  Q = new query()
  Q.sql = "select * from init where id = 'men1'"
  q.active = true
  Initpos = q.rowset.fields["pos"].value
  Form.mv_menu.set_state(initpos)
  q.active = false
return
```

Men1 ist der Schlüssel, unter dem die Status-Information in der Tabelle abgespeichert wurde
Im canclose steht dann folgendes :

```
Function Form_canclose
  Local initpos
  Initpos = form.mv_menu.get_state()
  ** now store the string in the table ...
return
```

9) Das OnUpdate event

Wie die Original dbase toolbar hat auch die mv_toolbar ein onupdate-event. Dieses event wird alle 100 ms ausgelöst und kann dazu benutzt werden, das Erscheinungsbild der Toolbar anzupassen. Man könnte z.Bsp. Navigations-Buttons aktivieren oder deaktivieren, je nachdem ob das Rowset am Ende oder am Anfang ist.

Es reicht aus, in der Klassendefinition eine Funktion mit Namen "OnUpdate" zu schreiben, diese function wird dann automatisch alle 100 ms ausgeführt..

Damit haben wir alle Methoden und events der Basisklasse behandelt.

Werfen wir jetzt einen Blick auf die Mitglieder der „Menü“- und „Popup-Objekte“.

E). Die Mitglieder der Menü - und Popup-Objekte

1) Das „Clicked“ - Ereignis

Das wichtigste Mitglied der Objekt-Steuerung ist das Klick Ereignis. Der Zweck eines Menüs ist, dem Benutzer einen leichten und schnellen Zugang zu bestimmten Funktionalitäten zu geben. Diese Funktionalitäten müssen von Ihnen, dem Entwickler, zur Verfügung gestellt werden indem. Sie diese als Funktionen schreiben. Das Klick Event ist dazu da, die Funktionen die Sie geschrieben haben, in die verschiedenen Popup Objekte einzubinden

Sie verbinden das Ereignis mit seinem Steuerungsprogramm durch das Zuweisen eines Funktionszeigers zum Ereignis, genau wie bei irgendeinem anderen Ereignis in dBASE. Schreiben Sie das folgendes in die initialize-Methode der own_menu-Klasse:

```
This.m_item1.p3.clicked = class::closeform
```

Mit der eben geschriebenen Zeile fügten Sie gerade einen Zeiger auf einen sogenannten Eventntheadler dem "Close" Schalter des Menüs hinzu. Jetzt müssen Sie diesen Eventhandler schreiben. Da Sie das Schlüsselwort "Class" mit dem Operator zur Auflösung des Gültigkeitsbereichs "::" verwendeten, erwartet dBASE die Funktion closeform in derselben Klassendefinition.

Setzen Sie den Cursor kurz vor den endclass Befehls der own_menu Klasse und tippen Sie folgende Zeile ein:

```
Function closeform  
    this.form.close( )  
return
```

Dieses Stück des Codes wird genau das tun was das Menü Objekt aussagt: Nämlich das Formular einfach schliessen.

Bemerkung: Sowohl das Popup Objekt als auch das Menü Objekt haben ein Klick-Ereignis, aber nur das Steuerungsprogramme des Popup Objektes wird automatisch aufgerufen, wenn Sie auf das entsprechenden Objekt klicken, Wenn Sie auf einen Menü Objekt klicken, wird es nur sein Untermenü öffnen, aber nicht die dem geklickten Ereignis zugeordnete Funktion ausführen. Dies geschieht, um die zwei Objekte so symmetrisch wie möglich zu halten, und es gibt Ihnen die Möglichkeit, das Ereignis mittels Programmcode aufzurufen: this.m_item1.clicked ()

Bemerkung: Nur "echte" Funktionen können dem Klick-Ereignis zugeteilt werden, Codeblocks sind nicht erlaubt. So wie für den Einzeiler oben müssen wir einen Funktionskörper schreiben

2) Die Form-Eigenschaft

Die Funktion, die die Form oben schliesst, verwendet diese Codezeile: `This.form.close ()`. Jedes Menü Objekt hat eine Eigenschaft, genannt "Form", die einen direkten Verweis auf das Formular ist, an das die Menubar angehängt wurde. Mit dieser Eigenschaft können Sie einen Verweis auf jedes Objekt auf der bestimmten Form anlegen, wie: `This.form.entryfield1.copy ()`, das den Inhalt von `entryfield1` in die Zwischenablage kopieren wird.

Innerhalb des Eventhandlers können Sie "Form" nicht direkt verwenden (`form.close ()` wird einen Fehler ausgeben). Die Items sind Subklassen von nicht sichtbaren dBASE Objekt Klassen, diese wissen nichts von einem Ding Namens "Form". Es ist ähnlich dem Rowset-Objekt, wo Sie `this.parentparent` verwenden müssen, um Form anzusprechen. Mit der Form-Eigenschaft des Items, können Sie die `parents` auslassen und einfach `this.form` schreiben.

3) Die "enabled" und die "visible" Eigenschaft

Diese Eigenschaften hängen zusammen mit den `Set_Enabled` und `Set_Visible` Methoden. Sie können diese Eigenschaften lesen d. h. `this.enabled` (wird wahr oder falsch zurückgeben), aber um einen Wert zu ändern, müssen Sie die passende Methode verwenden. Um sich zu überzeugen, wie sie wirken, schreiben Sie die folgende Zeile nach der Instantiierung des open Item:

```
This.m_item1.p2.set_enabled(false)
```

Wenn Sie Ihre Form starten, werden Sie sehen, dass das open-item noch vorhanden ist, aber dass Sie es nicht mehr auswählen können. Verwenden Sie dieselbe Syntax mit `Set_Visible`, um ein item völlig zu verbergen. Das ist besonders nützlich, wenn Sie Ihre Menüs abhängig von Benutzerrechten kundenspezifisch anfertigen wollen. Definieren Sie das ganze Menü und verbergen Sie jene Teile (mit `Set_Visible (false)`) die der zurzeit geloggt Benutzer nicht sehen darf.

4) Die Text-Eigenschaft

Diese Eigenschaft geht zusammen mit der `Set_Text` Methode. Jederzeit können Sie die Überschrift eines bestimmten Items leicht ändern. In der `next_menu` Klasse, schreiben Sie das Folgende:

```
this.M_item1.set_text("&Data")
```

Dieser Code wird die Überschrift unseres ersten menuitems auf "Data" ändern: Die anfängliche Überschrift wurde während der Instantiierung als zweiter Parameter des New Operator übergeben.

5) Die Typ-Eigenschaft

Dieses Eigenschaft ist Read-only und zeigt den Typ des Items an:

```
Menuitem = 0  
Popupitem = 1
```

6) Die Shortcut Eigenschaft

Tatsächlich gibt es zwei mit Shortcut verbundene Eigenschaften: die Shortcut Eigenschaft selbst und die Shortcut-Modifikator-Eigenschaft. Die erstere identifiziert einen Tastenanschlag, der das geklickte Ereignis des entsprechenden Items direkt starten wird; der Letztere zeigt an, ob ein Modifikator-Schlüssel (shift, control, alt) gleichzeitig gedrückt werden muss.

Die Include Datei definiert die folgenden Konstanten:

```
#define          FNONE          0
#define          FSHIFT         4
#define          FCTRL          8
#define          FALT           16
```

Wie mit den anderen Eigenschaften benutzen wir die Set_Shortcut-Methode, um den Eigenschaftswert tatsächlich zu verändern. Die Methode benötigt zwei Parameter: der Erste ist der Keycode für die shortcut Eigenschaft und der Zweite, der den Wert für den Modifikator darstellt. Keycodes werden als ASCII-Codes der Taste definiert. Für spezielle tasten können Sie die ganze Liste in der include Datei finden. Zum Beispiel, die <ENDE>-Taste wird definiert als:

```
#define          VK_END          0x23
```

Fügen Sie die folgende Zeile zur next_menu Klasse hinzu

```
this.m_item1.p3.Set_Shortcute (VK_END,FCTRL)
```

Der shortcut wurde so entworfen, dass die lokalen Abkürzungen für die Tastaturanschläge gezeigt werden, auf englischen Systemen wird er als: Ctrl + End, auf deutschen Systemen als Strg + Ende dargestellt. Die Wirkung ist auf allen Betriebssystemen dieselbe, wenn Sie die Endtaste zusammen mit der Kontrolltaste drücken, wird die Form geschlossen.

7) Die checked Eigenschaft

Die Eigenschaft zusammen mit der Set_Checked Methode wird verwendet, um eine Kontrollmarkierung (Häkchen) vor das entsprechende Item zu setzen. Setzen Sie folgende Zeile direkt nach der Instantiierung des cancel-item in der next_menu Klasse

```
This.m_item2.cancel.set_checked(true)
```

Wenn Sie die Form starten, werden Sie eine Kontrollmarkierung (Häkchen) vor dem Cancel-item sehen

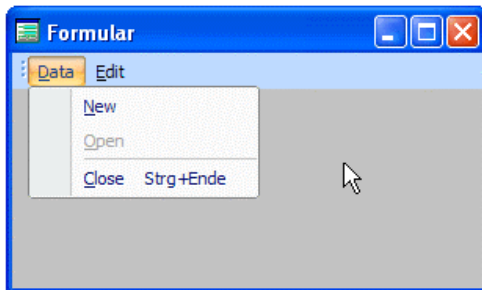
Es gibt auch einige Methoden, die nicht mit Eigenschaften verbunden werden. Sehen wir uns die einmal an:

8) Die Set-Separator-Methode

Diese Methode wird verwendet, um ein Item vom Rest der Items abzutrennen. Eine Linie wird zwischen dem getrennten Item und dem Rest gezogen. In der `own_menu` Klasse, fügen Sie die folgende Zeile hinzu:

```
this.m_item1.p2. set_separator ( true )
```

Wenn Sie die Form starten werden Sie folgendes sehen:



9) Die Set Icon Methode

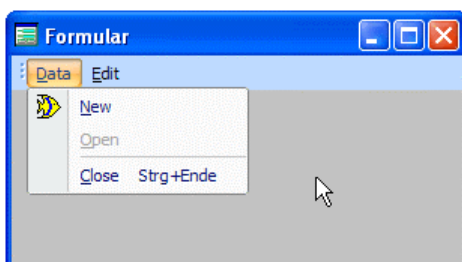
Wie der Name schon sagt, wird diese Methode ein Icon einem Menü-Item beifügen. Sie müssen einen gültigen Dateinamen als Parameter übergeben. Der Dateiname kann ein relativer oder absoluter Pfad sein. Der relative Pfad zeigt auf die Position wo Ihr `*.cc` abgelegt ist. Der absolute Pfad beginnt da wo Ihre `*.exe` abgelegt ist. Im Allgemeinen sind absolute Pfade keine gute Idee, also konzentrieren wir uns auf den relativen Pfad.

Wenn Sie schreiben: `Set_Icon ("fish.ico")`, sucht das Steuerelement im Ordner in der die `*.cc` abgelegt ist nach einer Datei mit Name `fish.ico`. Wenn es sie nicht finden kann, wird kein Icon angezeigt. Es ist eine gute Angewohnheit, Bilder und Icons in einen getrennten Ordner zu legen. Sie können auf diese Dateien wie folgt zugreifen: `Set_Icon (".\images\fish.ico")`. Der Pfadname folgt den weithin bekannten Regeln für relative Pfade: `."` ist der aktuelle Ordner und `.."` ist das übergeordnete Verzeichnis.

Wollen wir das in unserer Demoklasse versuchen, und folgendes hinzufügen

```
This.m_item1.P1.set_icon ("fish.ico")
```

Wenn Sie die Datei `fish.ico` aus dem Beispiel Ordner in Ihr eigenes Verzeichnis kopiert haben sollten Sie folgendes sehen:



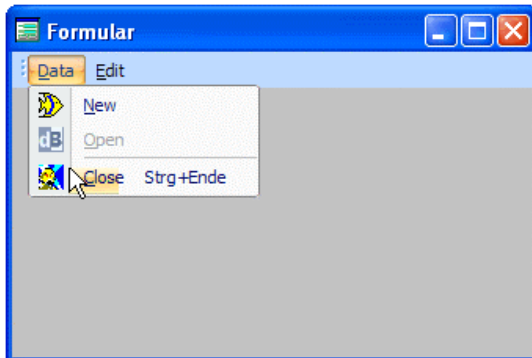
10) Die Set_Resource Methode

Wenn Sie viele Icons und/oder Bitmaps in Ihrer Anwendung haben, ist es eine gute Idee, sie in einer Resource dll zu gruppieren. Sie müssen nicht Dutzende von kleinen Dateien während der Installation kopieren. Dasselbe gilt, wenn Sie einige der Icons durch neue ersetzen wollen; Sie ändern Ihre DLL und senden diese an Ihren Kunden. Die Set_Resource Methode wird verwendet, um Icons aus dieser Resource herauszuziehen.

Diese Methode benötigt zwei Parameter: Der erste ist der Dateiname der Resource-DLL, der zweite ist der Name oder die Ordnungszahl des Icon in der Resource-Datei (beide sind vom Typ String). Der Dateiname kann den ganzen Pfad zur Datei enthalten, wenn nicht, wird die Klasse zuerst im aktuellen Verzeichnis, dann im Runtime-Verzeichnis und zuletzt im Verzeichnis Windows\system32 suchen. Wenn die Datei nicht gefunden wird, wird das Icon nicht angezeigt.

```
This.m_item1.p2.set_resource("plus_en.dll", "31233")
This.m_item1.p3.set_resource("resource.dll", "ROSWELL")
```

Die erste Zeile wird das Icon auf die Icon Resource mit der Ordinalzahl 31233 in der plus_en.dll im dBASE Homeverzeichnis setzen (dies ist das dBi Icon) die zweite Zeile wird das Icon mit Namen "ROSWELL" verwenden. Das Ergebnis sollte jetzt so aussehen:



10) Das OnShowPopup event

Menu items (nicht popup items) besitzen ein spezielles event, genannt : onshowpopup.

Diese Ereignis feuert immer dann, wenn der User auf ein Menu Item klickt, aber bevor das zugehörige Popup aufklappt. Die ist nützlich, in den Fällen, wo gewisse Positionen im Popup aktiviert oder deaktiviert werden sollen, etwa wie im „Bearbeiten“ Menu, wo das „Kopieren“ Item nur aktiviert ist, wenn auch tatsächlich ein Textbereich markiert ist.

Der komplette Sourcecode für ein „Bearbeiten“ Menu befindet sich in der demonstrations-cc

Wenn Sie diesem Tutorenkurs bis hierhin gefolgt sind, wird Ihre own.cc Datei ähnlich dem Script auf der folgenden Seite aussehen.

```
#include mv_menu.h
Class own_menu(oparent) of mv_menu(oparent) Custom
  This.attach_to_framewin = true
  Function initialize
    This.m_item1 = new menuitem(this, "&File")
    This.m_item1.p1 = new popupitem(this.m_item1, "&New")
    This.m_item1.p2 = new popupitem(this.m_item1, "&Open")
    This.m_item1.p2.set_enabled(false)
    This.m_item1.p3 = new popupitem(this.m_item1, "&Close")
    This.m_item1.p3.clicked = class::closeform
    This.m_item1.p3.set_separator(true)
    This.m_item1.p1.set_icon("fish.ico")
  return

  Function closeform
    this.form.close()
  return
endclass

Class next_menu(oparent) of own_menu (oparent) custom
  Function initialize
    Super::initialize( )
    This.m_item2 = new menuitem(this, "&Edit")
    This.m_item2.m1 = new menuitem(this.m_item2, "&Action")
    This.m_item2.m1.p1 = new popupitem(This.m_item2.m1, "&Copy")
    This.m_item2.m1.p2 = new popupitem(This.m_item2.m1, "&Paste")
    This.m_item2.Cancel = new popupitem(This.m_item2, "Ca&ncel")
    This.m_item2.cancel.set_checked(true)
    This.set_theme (theme_ribbon)
    This.set_docking(true, dock_top + dock_bottom)
    This.m_item1.set_text("&Data")
    This.m_item1.p3.Set_Shortcut(VK_END, FCTRL)
    This.m_item1.p2.set_resource("plus_en.dll", "31233")
    This.m_item1.p3.set_resource("resource.dll", "ROSWELL")
  return
Endclass
```

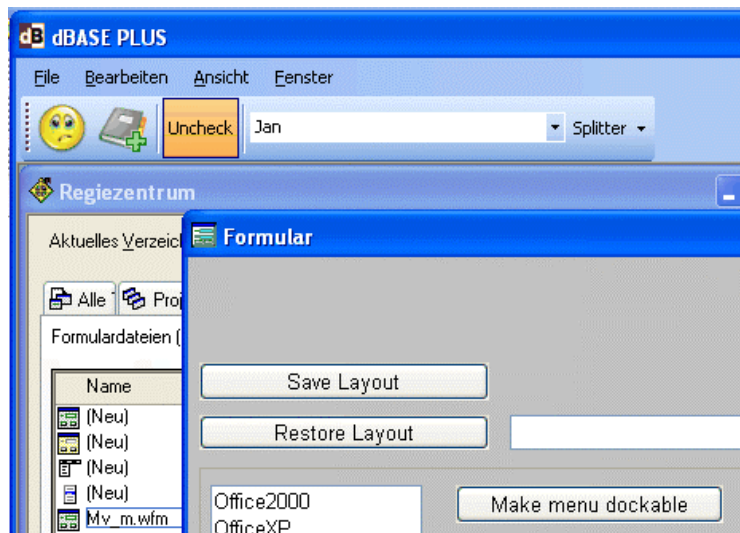
Schauen wir uns nun gemeinsam die Toolbar etwas genauer an.

F) Die mv_Toolbar

Die mv_toolbar hat viele Gemeinsamkeiten mit der Original dbase Toolbar, trotzdem gibt es die eine oder andere Verbesserung :

- Zuerst ist da natürlich der wesentlich modernere Look. Die mv_Toolbar stellt sich im gleichen Gewand dar, wie das mv_Menu
- Man kann den Toolbuttons Icons in beliebiger Grösse zuordnen, die Beschränkung auf 16x16 Pixel entfällt.
- Die Toolbar Buttons können Icons, Text oder beides enthalten.
- Zu guter Letzt, ist die Toolbar nicht auf Buttons beschränkt, sie kann auch Comboboxen und sogenannte Split-Buttons aufnehmen.

Unten sehen sie das mögliche Aussehen einer mv_Toolbar...



Die Benutzung der mv_Toolbar ist ähnlich wie die des mv_Menus, man kann controls einfach mit Hilfe des new-Operator hinzufügen.

```
This.t_bar = this.add_toolbar("File" , 0)
this.tbar.t1 = new mv_toolbutton(this.tbar, "FILE")
this.tbar.cb1 = new mv_toolcombo(this.tbar, "Combobox")
this.tbar.sl = new mv_splitbutton(this.tbar, "Splitter")
```

Der oben gezeigte Code fügt der Toolbar drei verschiedene Steuerelemente hinzu, einen normalen Toolbutton, eine Combobox sowie ein Splitbutton. Ein Splitbutton ist nichts anderes als ein Toolbutton mit einem kleinen Pfeil an der rechten Seite. Klickt der User auf den „Button-Teil“, so wird das „clicked“ event ausgelöst. Klickt er hingegen auf den Pfeil, so wird ein separates Popumenu aufgeklappt.

In der folgenden Zusammenfassung werden noch einmal alle Eigenschaften, Methoden und Ereignisse der verschiedenen Controls aufgelistet

G) Zusammenfassung

Menubar :

- **Theme / Set_Theme** : holt/setzt das Erscheinungsbild des Menüs..
- **Set_Docking** : gibt an, wie das Menu in seinem Formular gedockt werden kann.
- **Version** : diese Eigenschaft enthält die Versionsnummer der aktuellen mv_Menu-Klasse
- **Attach_to_framework** : falls diese Eigenschaft auf true gesetzt wird, dann kann das Menu an das _app.framework gebunden werden (default = false)
- **Add_toolbar(ctitle,npos)** : Fügt dem Formular, das die Menubar enthält, eine Toolbar hinzu
- **Set_Visible** : Versteckt / zeigt die Menubar (default = true)
- **Get_State / Set_State(cstate)** : ermöglicht es die genauen Positionsangaben des mv_Menu zu lesen bzw. zu setzen
- **Set_IconSize(nsize)** : bestimmt die Größe der Icons in der Toolbar. Diese Methode wirkt sich auf alle Icons aus.
- **OnUpdate** : Ereignis der Menubar, das alle 100 ms feuert.
- **Get_MenuHeight** : gibt die exakte Höhe in Pixel des Menus zurück

Menuitem/Popupitem :

- **Clicked** : sie können diesem Ereignisse eine normale dbase Funktion zuordnen. Diese Funktion wird immer dann aufgerufen, wenn der User das entsprechende Item anklickt.
- **Form** : jedes Menu- und jedes Popup-Objekt besitzt eine Referenz auf das parent Formular.
- **Enabled / Set_Enabled** : steuert, ob ein Element aktiviert ist oder nicht.
- **Visible / Set_Visible** : Steuert die Sichtbarkeit eines Elements.
- **Text / Set_Text** : holt/setzt den Text eines Menu-Eintrags.
- **Type** : nur-Lesen-eigenschaft, bestimmt den Typ des Objekts (menu/popup)
- **Shortcut / Shortcut_Modifier / Set_Shortcut** : steuert die Tastaturbelegung.
- **Checked / Set_checked** : steuert, ob ein Eintrag ein Häkchen bekommt oder nicht.
- **Set_Separator** : mit set_separator kann man eine waagrecht Linie über einem Eintrag erzeugen..
- **Set_Icon** : lädt ein Icon aus einer Datei.
- **Set_Resource** lädt ein Icon aus einer Resource-Datei, nach Namen oder Ordinalzahl.
- **OnShowPopup** : dieses Ereignis wird ausgelöst, wenn der User auf ein Item klickt, das ein Popup öffnet, aber bevor dieses Popup tatsächlich gezeigt wird.

Toolbar :

- **Set_Closeable** : steuert, ob eine schwebende Toolbar geschlossen werden darf oder nicht
- **Set_Gripper** : steuert den kleinen senkrechten Strich links in der Toolbar
- **Set_Pos** : steuert die Position der Toolbar, 0 = oben, 1= unten, 2= links, 3 = rechts , 4 = float
- **Set_Visible** : zeigt bzw. versteckt die Toolbar als Ganzes.
- **Set_Text** : diese Methode gibt der Toolbar einen Titel, mit dem sie bei einem Rechtsklick ins Menu, identifiziert werden kann.

Toolbar Button:

- **Set_Width : (Methode)** hängt zusammen mit der **width** eigenschaft, erwartet einen numerischen Parameter, der die Breite des Controls in Pixel angibt
- **Set_Height : (Methode) / height** Eigenschaft, erwartet einen numerischen Parameter, der die Höhe in Pixel angibt
- **Set_Tooltip : (Methode) / tooltip** Eigenschaft, erwartet einen String als Parameter. Dieser String wird als Tooltip angezeigt, wenn der Cursor eine Weile über dem Control verweilt.
- **Set_Text : (Methode) / Text** Eigenschaft, erwartet einen String als Parameter. Dieser String wird im Toolbutton angezeigt, wenn dessen Stil 1 oder 3 ist (siehe auch `set_style`)
- **Set_Separator : (Methode)** true oder false: zeigt an, ob vor dem Toolbutton ein senkrechter Strich erscheinen soll. Dies wird benutzt, um Bereiche innerhalb der Toolbar voneinander abzugrenzen..
- **Set_enabled : (Methode) / enabled** Eigenschaft (default = true). Falls `Set_enabled(false)` aufgerufen wird, so wird das Element deaktiviert und reagiert nicht mehr auf User-Aktionen, ausserdem wird es in einer anderen Farbe dargestellt.
- **Set_Checked : (Methode) / checked** Eigenschaft, (default=false). Erlaubt es, einen visuellen Effekt zu zeigen, der angibt, ob ein Element einen Wert besitzt oder nicht.
- **Set_Icon : (Methode)** , wie bei den Menu-Elementen, kann hiermit ein Icon geladen werden, das dargestellt wird wenn der Buttonstil auf 2 gesetzt ist. Falls eine Icon-Datei das gleiche Bild in mehreren Größen enthält, wird immer jenes ausgewählt, das am besten zu der mit `Set_IconSize` voreingestellten Größe passt. Liegt nur ein Icon vor, so wird dieses entweder vergrößert oder verkleinert.
- **Set_Resource : (Methode)** , entspricht der Set-Icon Methode, nur dass hier die Bilder aus einer Resource – Datei geholt werden (siehe auch die gleichnamige Methode bei den Menus)
- **Set_Shortcut : (Methode) / shortcut and shortcut_modifier** Eigenschaft, erwartet zwei Parameter : der erste gibt den Ascii-Wert des Tastaturkürzels an, der zweite enthält einen Modifikator (SHIFT , ALT , CONTROL) der gleichzeitig gedrückt werden muss, damit das zum Toolbutton gehörige „clicked“ event feuert. (siehe auch die Erklärung bei den Menus).
- **Set_Style : (Methode) / style** Eigenschaft , steuert das Aussehen des Toolbuttons. Wird hier 1 übergeben, so enthält der Button nur den Text, der mit `Set_text` voreingestellt wurde. Der Wert 2 bedeutet, dass nur ein Icon im Button zu sehen ist und 3 bedeutet, dass sowohl ein Icon als auch Text zu sehen ist.
- **Clicked : (event)** jedes Mal wenn der toolbutton gedrückt wird, wird die Funktion aufgerufen, die diesem Ereignis zugeordnet wurde (Codeblocks sind nicht erlaubt).

ToolbarSplitbutton:

Der Toolbar-Splitbutton hat genau die gleichen Eigenschaften und Methoden, wie der normale Toolbutton. Man kann aber dem Split-Button ein Popup-Menü zuordnen. Dieses wird aufgerufen, wenn der Benutzer den kleinen Pfeil des Split-Buttons drückt.

Wie Menu-Items, so besitzt auch der Split-Button ein `onshowpopup` Ereignis, das feuert bevor das Popup aufklappt.

ToolbarCombobox:

Die folgenden Methoden entsprechen denen der Toolbuttons : Set_Width, Set_Height, Set_Tooltip, Set_Separator, Set_Enabled und Set_Visible.

Die folgenden Methoden sind speziell für die Combobox

- **Set_Datasource : (Methode)** diese Funktion erwartet ein Array als Parameter. Die einzelnen Bestandteile des Array werden dann in der Combobox dargestellt.
- **Set_canEdit : (Methode) / canedit** Eigenschaft steuert, ob der User Eingaben in der Combobox machen kann. Wenn hier false übergeben wird, dann dürfen nur die Werte ausgewählt werden, die im datasource array angegeben sind.
- **Get_Data : (Methode) / cursel** Eigenschaft. Diese Funktion gibt den Text zurück, der gerade im Edit-Feld der Combobox angezeigt wird.
- **Get_Index : (Methode) / index** Eigenschaft. Der Rückgabewert dieser Methode enthält den Index des Textes im Array, der gerade angezeigt wird. Wenn canedit wahr ist, dann kann hier auch der Wert 0 zurückgegeben werden.
- **Set_Index : (Methode)** Falls man einen bestimmten Wert voreinstellen will, so kann man den Index auf diesen Wert im Array an diese Methode übergeben.
- **Clicked : (event)** siehe toolbutton. Hier kann das Ereignis auch als ongotfocus betrachtet werden
- **LostFocus : (event)** Dieses Ereignis feuert, wenn die Combobox den Fokus verliert.
- **OnKey : (event)** jedes Mal, wenn ein Tastenanschlag im Edit-Teil der Combobox geschieht, wird dieses Ereignis ausgelöst.

All diese Eigenschaften, Methoden und Ereignisse können in Natura im beigefügten Demo Formular betrachtet werden. Dort findet sich auch der komplette Sourcecode für ein vollständiges „Bearbeiten“ und „Fenster“ Menu.

Besonderer Dank geht an René Debrunner, der den größten Teil dieses Dokuments verfasst hat.

Ich versuche ständig, das mv_menu zu erweitern.

Falls Sie Fragen oder Anregungen haben, erreichen sie mich unter : info @ vdblogic . de

Marc Van den Berghen

www.vdblogic.de/dbl
