

Vortrag zum Thema:

**Tabulator-Reihenfolge in einer Eingabemaske dynamisch ändern
und
Ersatz des " Control Creation Order"-Fensters von VO**

Vorgetragen in der "Visual Objects Usergroup Bodensee" am 30.9.2010

© 2010: Jürgen Asal, Asal Informatik GmbH, CH 4146 Hochwald.

Dieses Script darf, unter Nennung des Urhebers, ohne Einschränkungen
verwendet und kopiert werden.

1. Ziel

Es soll eine Möglichkeit eingebaut werden, die Reihenfolge der Controls, welche durch Betätigen der TAB-Taste angesprochen werden, komfortabel während der Laufzeit der Applikation ändern zu können.

Zusätzlich soll die TAB-Reihenfolge im Window-Editor komfortabel eingegeben werden können ohne das "Control Creation Order"-Fenster (CTRL-K) benutzen zu müssen.

2. Einbau im Window-Editor

2.1. Ergänzungen der CAVOWED.INF

Zuerst wird ein neuer Assign eingebaut:

```
[CONTROL]
Assign01=Name,HyperLabel (%Name%, %Caption%, , )
Assign02=Caption,HyperLabel (, lstring, , )
Assign03=Description,HyperLabel (, , string, )
Assign04=Help Context,HyperLabel (, , , string)
Assign05=Background Color,Background (COLOR)
Assign06=Context Menu,ContextMenu (CLASS:Menu)
Assign07=Tooltip,TooltipText (string)
Assign08=Owner Alignment,OwnerAlignment (OWNERALIGNMENT)
Assign09=Group,Group (numeric)
Assign10=Owner Alig hhbbyyxx,OwnerAlignmentEx (string)
Assign11=Order,Order (numeric)
WindowState01=Disabled,WS_DISABLED (BOOL)
WindowState02=Tab Stop,WS_TABSTOP (BOOL)
:
:
```

(Die Doppelpunkte sind "Auslassungszeichen")

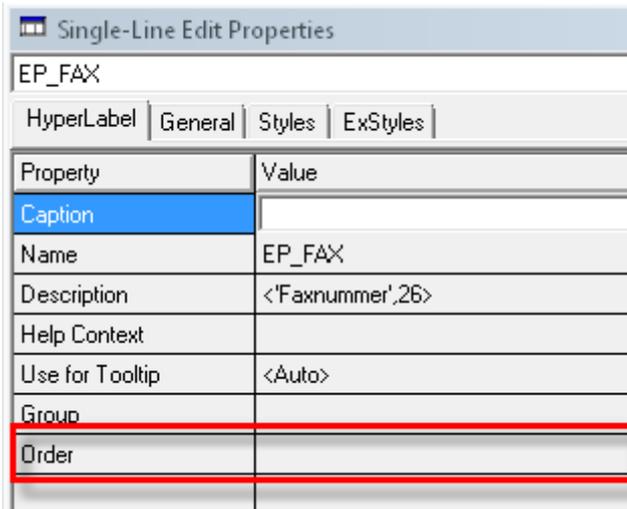
In der Gruppe "AssignMap" wird "Order" unten angefügt:

```
[AssignMap]
BackgroundColor=Color
BarColor=Color
Background=Brush
TextLimit=
:
:
OwnerAlignment=
FileName=
Group=
OwnerAlignmentEx=
Order=
```

Der Assign wird nun (z.B.) im Tab "Hyperlabel" des Window-Editors eingebaut. Der Einbau ist prinzipiell in jeder TAB des Editors möglich!

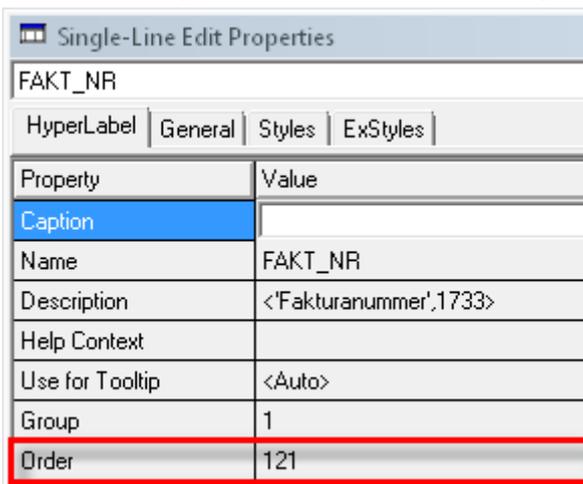
```
;**** Association between Tabs and methods/assigns/styles
HyperLabel=(HyperLabel Info)Caption,Name,Description,Help Context,Use for Tooltip,Group,Order
DataWindow=(DataWindow Properties)Inherit from Class, DBInhFrom, DCInhFrom,Class
Declaration,No Access/Assign,Export Controls,Pixel Positions,Automated,Large Icon,Small
Icon,Accelerator Table,Background Color,Menu,Context Menu,Help File Name,View As,Clipper
Keys,Tab Key,Data Server,Prevent Auto Layout,Allow Server Close,Defer USE,Quit On
Close,Font,StatusBar,Notify,PreValidate,PreInit Actions,PostInit Actions, Width, Height,Owner
Alignment
:
:
```

Bim nächsten Start von VO ist der neue Assign nun sichtbar



2.2. TAB-Reihenfolge in der Eingabemaske definieren

Nun muss für jedes Control die Reihenfolge angegeben werden:



Hier helfen zwei AutoHotkey-Makros ungemein, welche einen automatischen Zähler darstellen::

```

;=====
;== Automatischer Counter ==
;=====
;Win s      ==> Startwert für automatischen Counter eingeben
#s::
InputBox, nStart,Autocounter,Startwert eingeben,,175,120
;nStart := nStart+0
MsgBox Startwert wurde auf %nStart% gesetzt
return

;Counter abrufen und incrementieren
;Win +(num) ==> Automatischer Counter abrufen und inkrementieren
#NumPadAdd::
send %nStart%
nStart := nStart+1

```

Das Makro, welches mit den Tasten "Windows" + S aufgerufen wird, legt den Startwert des Counters fest:



Das zweite Makro setzt die Zahl ein und erhöht den Wert um eins:

The image shows a 'Single-Line Edit Properties' dialog box for a control named 'FAKT_NR'. The 'General' tab is selected. The 'Order' property is highlighted in blue and has the value '120'.

| Property | Value |
|-----------------|------------------------|
| Caption | |
| Name | FAKT_NR |
| Description | <'Fakturanummer',1733> |
| Help Context | |
| Use for Tooltip | <Auto> |
| Group | 1 |
| Order | 120 |

Einfach den Cursor in Feld "Order" setzen und die Tasten "Windows" und "+" auf dem numerischen Keyboard drücken. Die nächste Zahl des Counters wird eingesetzt und der Counter um 1 erhöht.

Hinweis: Damit noch "Luft" für nachträglich eingesetzt Controls bleibt, sollten ab und zu Lücken in den Nummern gelassen werden!

3. Implementierung in den Window-Klassen

3.1. Modifizierung der Window-Klassen

3.1.1. Protect Array einbauen

Alle Controls, bei welchen die TAB-Reihenfolge geändert werden sollen, werden in das Protect-Array `_aOrderControl` der Window-Klasse (DataWindow, DialogWindow wtc.) geladen.

Da es leider nicht möglich ist, eine Protect-Variable in eine VO-Klasse nachträglich einzufügen, bleibt uns nur den Weg über eine abgeleitete Klasse:

```
CLASS _DATAWINDOW INHERIT DATAWINDOW

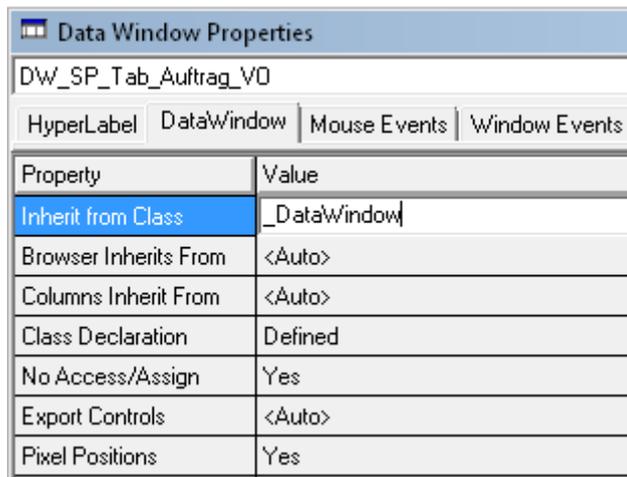
    // Array mit Controls, welche sortierbar (Tab-Reihenfolge) sein müssen
    PROTECT _aOrderControl AS ARRAY
```

In der INIT-Methode muss noch ein leeres Array zugewiesen werden. Ohne dies wäre `aOrderControl` mit `NULL_ARRAY` initialisiert und nicht mit einem leeren Array, und das spätere `AADD()` würde crashen.

```
METHOD Init (oOwner, oSource, nResourceID) CLASS _DATAWINDOW
:
    _aOrderControl := {}
:
```

3.1.2. Vererbung im Maskengenerator einbauen

Die Maske muss jetzt natürlich von `_DataWindow` vererbt werden!



| Property | Value |
|-----------------------|-------------|
| Inherit from Class | _DataWindow |
| Browser Inherits From | <Auto> |
| Columns Inherit From | <Auto> |
| Class Declaration | Defined |
| No Access/Assign | Yes |
| Export Controls | <Auto> |
| Pixel Positions | Yes |

3.1.3. Control registrieren

Nun benötigen wir noch eine Methode, welche das Control in das Array anfügt:

```
METHOD RegisterOrder (oControl AS Control, nOrder AS DWORD) AS VOID PASCAL CLASS _DATAWINDOW
    //s Anmelden eines Controls mit seiner Anzeigereihenfolge
    //p oControl \tab Anzumeldendes Control
    //p nOrder \tab Reihenfolge
    AAdd ( _aOrderControl, { oControl, nOrder } )
```

Hinweis: Es wird natürlich nicht das ganze Control im Array eingefügt, sondern nur einen Pointer. Somit ist kein Performance-Nachteil spürbar.

Hinweis: Diese Methode muss natürlich in allen Klassen "DataWindow, DialogWindow" etc. eingebaut werden!

Das Registrieren dieses Controls kann man einfach automatisieren:

In der VO-Klasse CONTROL wird ein zusätzlicher Assign eingebaut. Dies ist im Gegensatz zum Hinzufügen einer Protect-Variable problemlos möglich:

```
ASSIGN Order (nOrder) CLASS Control

    IF IsMethod(SELF:owner, #RegisterOrder)
        SELF:owner:RegisterOrder (SELF, nOrder)
    ENDIF

RETURN nOrder
```

Die Abfrage "IsMethod" soll nur verhindern dass das Programm abstürzt, falls die Methode "RegisterOrder" in der Supperklasse nicht vorhanden ist (z.B. weil man Schritt 3.1.1 vegessen hat).

3.1.4. Assign "Order" aufrufen

Wie wird nun das Assign "Order" aufgerufen? Dies erledigt VO über den Maskengenerator.

Schauen wir mal den generierten Code an:

```
oDCF PAY ACC := SINGLELINEEDIT(SELF,ResourceID(DW SP TAB AUFTRAG VO F PAY ACC,
GetInst()),Point{77, -2678},Dimension{80, 20})

oDCF_PAY_ACC:HyperLabel := HyperLabel{#F_PAY_ACC,"PAY_ACC",LoadResString("Who pays the
transport ?",683),NULL_STRING}

oDCF PAY ACC:Group := 6
oDCF PAY ACC:OwnerAlignmentEx := "00140000"
oDCF_PAY_ACC:Order := 421
```

Die letzte Zeile ist die Zuweisung der Reihenfolge via das Assign, generiert vom Maskengenerator von VO.

3.2. Code für "TAB-Reihenfolge ändern"

Das eigentliche Ändern der TAB-Reihenfolge wird in folgender Methode vorgenommen:

```
METHOD SetTabOrder() AS VOID PASCAL CLASS DATAWINDOW
    //p Ordnet die Tab-Reihenfolge neu an

    LOCAL nII AS DWORD
    LOCAL nLen AS DWORD

    // Nach Ordernummer aufsteigend sortieren
    ASort(_aOrderControl,,{|x, y| x[2] <= y[2]})

    // Neue Reihenfolge
    nLen := ALen(_aOrderControl )
    FOR nII := 2 UPTO nLen
        SetWindowPos( aOrderControl[nII,1]:handle(), aOrderControl[nII-1,1]:handle(),;
            0,0,0,0,SWP_NOMOVE+SWP_NOSIZE)
    NEXT
```

3.2.1. Windows-API-Funktion "SetWindowsPos"

Die alles entscheidende Funktion ist "SetWindowsPos" aus dem Windows API:

SetWindowPos

The **SetWindowPos** function changes the size, position, and Z order of a child, pop-up, or top-level window. Child, pop-up, and top-level windows are ordered according to their appearance on the screen. The topmost window receives the highest rank and is the first window in the Z order.

```
BOOL SetWindowPos(
HWND hWnd,           // handle of window
HWND hWndInsertAfter, // placement-order handle
int X,               // horizontal position
int Y,               // vertical position
int cx,              // width
int cy,              // height
UINT uFlags          // window-positioning flags
);
```

Parameters

hWnd

Identifies the window.

hWndInsertAfter

Identifies the window to precede the positioned window in the Z order. This parameter must be a window handle or one of the following values:

| Value | Meaning |
|----------------|--|
| HWND_BOTTOM | Places the window at the bottom of the Z order. If the hWnd parameter identifies a topmost window, the window loses its topmost status and is placed at the bottom of all other windows. |
| HWND_NOTOPMOST | Places the window above all non-topmost windows (that is, behind all topmost windows). This flag has no effect if the window is already a non-topmost window. |
| HWND_TOP | Places the window at the top of the Z order. |
| HWND_TOPMOST | Places the window above all non-topmost windows. The window maintains its topmost position even when it is deactivated. |

For more information about how this parameter is used, see the following Remarks section.

X

Specifies the new position of the left side of the window.

Y

Specifies the new position of the top of the window.

cx

Specifies the new width of the window, in pixels.

cy

Specifies the new height of the window, in pixels.

uFlags

Specifies the window sizing and positioning flags. This parameter can be a combination of the following values:

| Value | Meaning |
|------------------|---|
| SWP_DRAWFRAME | Draws a frame (defined in the window's class description) around the window. |
| SWP_FRAMECHANGED | Sends a WM_NCCALCSIZE message to the window, even if the window's size is not being |

| | |
|--------------------|--|
| | changed. If this flag is not specified, WM_NCCALCSIZE is sent only when the window's size is being changed. |
| SWP_HIDEWINDOW | Hides the window. |
| SWP_NOACTIVATE | Does not activate the window. If this flag is not set, the window is activated and moved to the top of either the topmost or non-topmost group (depending on the setting of the hWndInsertAfter parameter). |
| SWP_NOCOPYBITS | Discards the entire contents of the client area. If this flag is not specified, the valid contents of the client area are saved and copied back into the client area after the window is sized or repositioned. |
| SWP_NOMOVE | Retains the current position (ignores the X and Y parameters). |
| SWP_NOOWNERZORDER | Does not change the owner window's position in the Z order. |
| SWP_NOREDRAW | Does not redraw changes. If this flag is set, no repainting of any kind occurs. This applies to the client area, the nonclient area (including the title bar and scroll bars), and any part of the parent window uncovered as a result of the window being moved. When this flag is set, the application must explicitly invalidate or redraw any parts of the window and parent window that need redrawing. |
| SWP_NOREPOSITION | Same as the SWP_NOOWNERZORDER flag. |
| SWP_NOSENDCHANGING | Prevents the window from receiving the WM_WINDOWPOSCHANGING message. |
| SWP_NOSIZE | Retains the current size (ignores the cx and cy parameters). |
| SWP_NOZORDER | Retains the current Z order (ignores the hWndInsertAfter parameter). |
| SWP_SHOWWINDOW | Displays the window. |

Return Values

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

3.2.2. API-Aufruf aus Library

Dieser API-Aufruf ist in der Library "WIN32 API Library" enthalten. Falls man diese Lib nicht einbinden will, kann der Aufruf der DLL-Funktion auch in der Applikation eingebaut werden:

```
_DLL_FUNCTION SetWindowPos( hwnd AS PTR, hWndInsertAfter AS PTR, x AS INT, y AS INT, ;
  cx AS INT, cy AS INT, uFlags AS DWORD);
AS LOGIC PASCAL:USER32.SetWindowPos
```

4. Verwendung in der Anwendung

4.1. Aufruf von SetTabOrder

In der Init-Methode jeder Maske, welche die Tabreihenfolge ändern soll, muss nun die Methode SetTabOrder aufgerufen werden. Dies muss aber nicht in der INIT-Methode passieren, der Aufruf kann jederzeit erfolgen, auch nachdem die Maske bereits initialisiert und angezeigt wird!

```
METHOD INIT(oWindow, iCtlID, oServer) CLASS DW SP Tab Auftrag
    SUPER:INIT(oWindow, iCtlID )
    SELF:SetTabOrder ()
```

Der Rest geschieht nun automatisch.

Auf diese Weise kann nun die unselige und nervige Fuktion "Control Creation Order" (CTRL-K) des Window-Editors eliminiert werden.

4.2. Dynamische Veränderung der TAB-Reihenfolge

Die TAB-Reichenfolge kann nun sehr einfach zur Laufzeit der Applikation verändert werden, indem im Array `_aOrderControl` die Nummer der TAB-Reihenfolge verändert wird. Anschliessend braucht nur noch SetTabOrder erneut aufgerufen werden, und die geänderte Reihenfolge ist aktiv.

Das folgende Beispiel verdeutlicht dies. Alle Controlls, welche eine TAB-Reihenfolge zwischen 20 und 50 haben, werden nun ans Ende geschoben, indem einfach der Wert 400 zur TAB-Reihenfolge addiert wird:

```
METHOD SetOrderOfDimensionFields() CLASS DW_SP_TAB_Posit
    //p Setzt den Block mit den Dimmensionen ans Ende der TAB-Reihenfolge
    // (8) J.Asal, 23.08.2010

    LOCAL nLen          AS DWORD
    LOCAL nII           AS DWORD

    nLen := ALen(SELF: aOrderControl)

    FOR nII := 1 UPTO nLen
        IF SELF:_aOrderControl[nII,2] >=20 .AND. SELF:_aOrderControl[nII,2] <=50
            SELF:_aOrderControl[nII,2] += 400
        ENDIF
    NEXT

    SELF:SetTabOrder ()
```