VFX9 Tipps und Tricks

Rainer Becker

Seit der Übernahme des Frameworks Visual Extend durch die dFPUG c/o ISYS GmbH erstelle ich mehr und mehr Anwendungen mit VFX. Mittlerweile sogar fast ausschließlich, da sich diese Vorgehensweise in vielerlei Hinsicht bewährt hat. Unter anderem durch schnellere Abwicklung und bessere Kalkulierbarkeit von Projekten sowie regelmäßigere Lieferung von Updates im Rahmen von Wartungsverträgen mit niedrigerem Aufwand.

Im Rahmen der praktischen Arbeit sammeln sich eine Vielzahl von kleinen Hilfsroutinen und nützlichen Funktionen. Im gedruckten Handbuch zu VFX 8.0 hatten wir dazu eine Tipps & Tricks-Sammlung von Stefan Zehner veröffentlicht. Das gedruckte Handbuch zu VFX 9.0 wurde aber wesentlich umfangreicher, so dass für Anhänge kein Platz mehr war. In diesem Beitrag deshalb eine separate Tipps & Tricks-Sammlung zu VFX 9.0 und früher.

Hinweise zum Einstieg

Bestimmt ist es Ihnen auch schon einmal passiert. In der Headerdatei USERDEF.H stehen die benutzerdefinierten Konstanten, die von VFX nicht überschrieben werden wie zum Beispiel:

```
#DEFINE INTROFORM_LOC
   && Startbild für Splash-Formular
#DEFINE DESKTOP_LOC
   && Hintergrundbild für den Desktop
```

Aber nach Änderung der Bildnamen erscheinen weiterhin die alten Bilder. Da hilft auch das mehrfache komplette neu Erstellen des Projektes nichts. Abhilfe schafft hier nur das Löschen aller .FXP-Dateien im PROGRAM-Verzeichnis.

Das war jetzt wirklich kein neuer Tipp, denn dieser Hinweis und einige weitere kurze Hinweise (wie zum Beispiel „Projektpfad immer richtig setzen“ oder, sehr wichtig!, „OneToMany in 1:n-Relationen nicht auf .T. setzen“) finden sich im Kapitel 16.25 im gedruckten VFX9-Handbuch (Seite 236ff.)!

Das nur vorab, aber nun lassen Sie uns mit ein paar Hooks beginnen, die jeder ohne Anpassung von Basisklassen oder Programmen in seine Anwendung integrieren kann:

Grundlagen von Hooks

Hooks sind eine einfache Möglichkeit, um das Verhalten von Klassen und Formularen von Visual Extend auf den eigenen Bedarf anzupassen, ohne direkt in den Quellcode eingreifen zu müssen. Dadurch entfallen mögliche Probleme bei der Installation neuer Builds oder neuer Versionen des Frameworks, da man keine Codeänderungen nachziehen muss.

An diversen Stellen in den Klassen von Visual Extend werden Hooks aufgerufen und in VFXHook.Prg findet sich der Eventhookhandler zum Platzieren der eigenen Funktionen:

```
FUNCTION eventhookhandler(tcevent, ;
         toobject, toform)
LOCAL lcontinue
lcontinue = .T.
*-- your code goes here
RETURN lcontinue
ENDFUNC
```

Diesen rudimentären Handler kann man sich beim Erstellen des ersten Hooks auch gleich noch ein wenig anpassen, um die weitere Programmierung von Hooks zu vereinfachen.

```
LOCAL lcBaseClass
tcEvent = UPPER(ALLTRIM( tcEvent ))
WITH toObject
lcBaseClass = UPPER( .baseclass )
```

Damit entfällt bei allen weiteren Hooks das Beachten der exakten Schreibweise des auslösenden Events (z.B. „OnPrint“) für den Hook sowie ggf. der exakten Schreibweise der Basisklasse (z.B. „Textbox“) der aufrufenden Klasse. Außerdem kann man das aktuelle Objekt durch die Änderung einfach mit einem „.“ am Zeilenanfang referenzieren. Noch weiter vereinfachen kann man sich die weitere Hookprogrammierung durch ein geschachteltes CASE-Statement folgender Art:

```
DO CASE
   CASE tcEvent=="INIT"
      DO CASE
         CASE INLIST( lcBaseclass, ;
         "TEXTBOX", "COMMANDBUTTON", ;
         "LABEL", "COMBOBOX" )
```

Auf der ersten Ebene wird das Ereignis abgefragt und auf der zweiten Ebene die entsprechende Basisklasse. Natürlich könnte man es auch umgekehrt staffeln, aber oft reagiert eine Gruppe von Klassen auf gleiche Art und Weise auf ein Ereignis.

Einfache Hook-Beispiele

Gut verwenden kann man Hooks für das generelle Einstellen von Eigenschaften von Steuerelementen im Init. Dadurch muss man die Basisklassen nicht anfassen. Zum Beispiel Abschalten des Tabstops für Textboxen und Commandbuttons, Einschalten von Hyperlinks in Editboxen, Abschalten der Größenänderbarkeit von Zeilen und Headern in Grids sowie schönere Darstellung von Containern:

```
IF lcbaseclass =="TEXTBOX" AND ;
                           .readonly
       .tabstop = .F.
ENDIF
IF lcbaseclass =="COMMANDBUTTON"
       .tabstop = .F.
       .caption = STRTRAN( .caption, ;
                  "\<", "" )
ENDIF

CASE lcbaseclass =="EDITBOX"
       .enablehyperlinks = .T.

CASE lcbaseclass =="GRID"
       .allowrowsizing=.F.
       .allowheadersizing=.F.
       .recordmark = .F.

CASE lcbaseclass =="CONTAINER"
       .style = 3    && -Themed
       IF UPPER( .name ) != "RECORD"
          .specialeffect = 1
                 && -sunken
       ENDIF
```

Gerne vergisst man auch mal das Eintragen von wahlweise einem Tooltip-Text oder einem Statusbar-Text, was mit folgender Zeile erledigt ist (sofern nur ein Text von beiden fehlt):

```
.tooltiptext  = EVL( .tooltiptext , ;
                     .statusbartext )
.statusbartext = EVL( .statusbartext,;
                      .tooltiptext )
```

Und gerne schalte ich auch mal Elemente temporär ab, ohne alle notwendigen Eigenschaften dafür alle einzeln umzusetzen – es wird einfach die Schriftart auf durchgestrichen geändert und die folgenden Zeilen erledigen den Rest:

```
IF .FontStrikethru
       .visible = .F.
       .enabled = .F.
ENDIF
```

Korrektur von DisplayCount

Besonders praktisch finde ich außerdem einen Hook für Comboboxen zur Festlegung der Anzahl der angezeigten Zeilen. Dies ist z.B. besonders praktisch im Filterformular für die Anzeige der auswählbaren Felder, denn der Standard von 7 Elementen ist meist nicht ausreichend. Mit der neuen Funktion SYS(2910) kann man die Anzahl der angezeigten Elemente in Auswahllisten einstellen, aber das bezieht sich nicht auf Comboboxen. Gerade im Filterformular ist das störend, da dort in der aktuellen Version von VFX nicht mit der Tastatur gescrollt werden kann, da direkt in das Wertfeld gesprungen wird. Dies kann man im Formular hardcodiert einstellen:

```
thisform.grdExpressions.colFields.;
   cmbFields.DisplayCount = lnitem+1
```

Oder man setzt global über einen Hook die Anzahl der angezeigten Einträge z.B. von 7 auf 15 hoch. Wesentlich höher sollte man den Wert nicht setzen, da sonst die Combobox nach oben oder unten aus dem Bildschirm wandert.

```
CASE lcbaseclass =="COMBOBOX"
       IF .DisplayCount = 0
          .DisplayCount = 15
       ENDIF
```

Wenn der Wert höher als die Anzahl der anzuzeigenden Elemente sein sollte, stört dies nicht weiter. Man kann DisplayCount bei einer Combobox mit 10 Elementen auch auf 99 setzen, ohne das Fehler auftreten oder mehr als die 10 vorhandenen Elemente gezeigt würden.

Defaultfarben für benötigte Felder

Komplizierter werden die Hookdefinitionen natürlich, sobald es sich um die Einstellung von Eigenschaften handelt, die nicht schon in der Basisklasse vorhanden sind, sondern erst ab einer bestimmten Subklasse in der Vererbungshierarchie. Da muß man natürlich erstmal prüfen, ob die Eigenschaft überhaupt vorhanden ist, um Fehlermeldungen zu vermeiden. Ein Beispiel dafür wäre die zentrale Einstellung der Farben von Steuerelementen für benötigte Felder (.crequiredfields) in Visual Extend ab der Version 9.0 statt individuell Einstellung in jeder Maske wie folgt:

```
IF tcevent="INIT" AND ;
   lcbaseclass="FORM" AND ;
   pemstatus( toobject, ;
   "crequiredfieldfailureprops", 5)

   .crequiredfieldfailureprops = ;
   EVL(.crequiredfieldfailureprops, ;
   "backcolor=rgb(255,255,0)")

   .crequiredfieldinitprops = ;
   EVL(.crequiredfieldinitprops ,;
   "backcolor=rgb(255,255,255)")
ENDIF
```

Man beachte insbesondere die Verwendung der Funktion EVL, um zu vermeiden, das im Formular bereits vorhandene abweichende Definitionen überschrieben werden. EVL eignet sich auch besonders für die Festlegung von Defaultwerten für übergebene Parameter in Methoden und Funktionen. Aufwändiger zu schreiben aber im Ablauf evtl. performanter wäre eine IF-ENDIF-Prüfung auf EMPTY().

Childformulare aus Childgrids aufrufen

Aber lösen wir doch einmal ein kompliziertes Problem! Ich möchte z.B. gerne aus Childgrids in einem VFX-cOneToMany-Formular auf Doppelklick oder Returntaste korrespondierende Childformulare aufrufen. Sofern ich über den Parent/Child-Builder die entsprechenden Child-Formulare verknüpft habe, müsste ich nur die Eintragsnummer übergeben und mehr wäre nicht zu tun. Wenn, ja wenn, die VFX-OnKeyEnter-Methode des VFX-Childgrids denn genauso aufgerufen werden würde, wie dies in den normalen VFX-Grids passiert. Dies ist nur leider nicht der Fall! Wollte ich es auf die abwärtskompatible Art von VFX tun, müsste ich nunmehr in sämtliche Textboxen entsprechenden Aufrufcode platzieren. Das ist für die Builder einfach, da die einfach die Vorlage aus der VFXCODE.DBF kopieren, aber manuell ist das doch etwas zu aufwändig – insbesondere bei der Pflege, wenn weitere Spalten hinzukommen. Man könnte das leicht bei einer neuen Spalte vergessen…

Deshalb nehmen wir lieber die neue BINDEVENT-Funktion von Visual FoxPro und platzieren diese in einem Hook. Allerdings möchten wir die Funktionalität nicht unbedingt bei jedem Childgrid, weshalb wir die Eigenschaft TAG mit irgendeinem Wert belegen (in diesem Fall die Klausel „ONMORE“). Das sieht dann wie folgt aus:

```
IF tcevent="INIT" AND ;
   lcbaseclass="GRID" ;
   AND UPPER(.class) = "CCHILDGRID" ;
   AND "ONMORE" $ UPPER(.tag )

LOCAL lnCounter
FOR lnCounter = 1 TO.ColumnCount

=bindevent(.columns(lnCounter).text1,;
"KeyPress", toObject, "onkeyenter")

=bindevent(.columns(lnCounter).text1,;
"DblClick", toObject, "onkeyenter")

NEXT
ENDIF
```

Und schon leiten sämtliche Textboxen der markierten Childgrids Ihre Keypress und DblClick-Events an die OnKeyEnter-Methode des Childgrids weiter. Statt an OnKeyEnter kann man es natürlich auch an die Methode DblClick binden, das ist Geschmackssache außer das die Grid.DblClick-Methode natürlich auch direkt durch den Anwender aufrufbar wäre.

In der Methode DblClick or OnKeyEnter müssen wir unbedingt eine LPARAMETER Anweisung mit einbauen, damit das gebundene Ereignis auch empfangen werden kann! Desweiteren wollen wir für den Fall der Returntaste den Tastendruck mit CLEAR TYPEAHEAD verwenden und unsere eigene Funktion aufrufen wie folgt:

```
LPARAMETERS nKeyCode, nShiftAltCtrl
CLEAR TYPEAHEAD
THISFORM.OnMore( <n> )
```

Das ist jetzt allerdings eine etwas sehr minimalistische Variante. Leider bietet das VFX-Childgrid noch keine leere Methode mitsamt Hook für diesen Fall an (was aber verbessert werden wird), so dass wir an dieser Stelle ohnehin keinen Hook einsetzen können. Also müssen wir in jedem Childgrid die entsprechende Methode ausprogrammieren wie folgt:

```
LPARAMETERS nKeyCode, nShiftAltCtrl
IF THISFORM.nformstatus=0 or ;
             THIS.ReadOnly
       CLEAR TYPEAHEAD
ENDIF
=DODEFAULT()
IF ( THISFORM.nformstatus=0 OR ;
          THIS.ReadOnly ) AND NOT ;
          EOF( THIS.RecordSource )
       THISFORM.OnMore( 1 )
ENDIF
```

Das Childformular wird nur aufgerufen, wenn die Maske nicht im Bearbeitenmodus ist bzw. wenn das Childgrid ohnehin Readonly ist und wenn ausserdem überhaupt Daten im Childgrid vorhanden sind. Im Beispiel wird die OnMore-Methode mit dem Parameter 1 aufgerufen. In einer separaten Methode am Childgrid könnte man das natürlich auch generisch ausformulieren und den Parameter ebenfalls aus der Eigenschaft .Tag oder einer zusätzlich definierten Eigenschaft auslesen. Aber zumindest bis hierhin hat uns ein einfacher Hook gebracht.

OnMore-Funktionen auf Shortcut-Menü

Wo wir gerade beim Wiederverwenden von OnMore-Funktionen sind, noch ein anderes kleines Beispiel. Die Auswahlmaske für Zusatzfunktionen auf der Funktionstaste F6 ist zwar sehr hilfreich, aber manche mögen es gerne moderner und hätten Zusatzfunktionen gerne auf der rechten Maustaste, auch wenn man den OnMore-Dialog etwas modernisieren kann (z.B. Borderstyle=Fixed Dialog, Closable=.T., Maske größer). Nachfolgend deshalb eine Funktion die auf Formularebene (Objektreferenz ToForm) die alle OnMore-Funktionen als Rechtsklick-Menü zur Verfügung stellt:

```
FUNCTION onmore2shortcut
LPARAMETERS ToForm
LOCAL lnCount, lcCmd, lnMax
LOCAL ARRAY laArray(1)
lnCounter = 0
lcCmd = ""
lnMax = TOFORM.onmore( -1, @laArray)
IF lnMax > 0
       DEFINE POPUP onmore SHORTCUT ;
          RELATIVE FROM MROW(),MCOL()
       FOR lnCounter = 1 to lnMax
          DEFINE BAR lnCounter OF ;
          onmore PROMPT laArray;
          [lnCounter,1] MESSAGE ;
          laArray [lnCounter,2]

          lcCmd = "ON SELECTION BAR" ;
          + ltrim(str(lnCounter)) + ;
          "OF onmore =_screen.;
          activeform.onmore(" + ;
          LTRIM(STR(lnCounter)) +")"

          &lcCmd
       NEXT
       ACTIVATE POPUP onmore
ENDIF
ENDFUNC
```

Dieses Beispiel basiert allerdings darauf, dass die OnMore-Methode bei Übergabe von -1 als Aufrufparameter als Rückgabewert die Anzahl der verfügbaren Funktionen liefert und im zweiten Parameter per Referenz das Array zur Verfügung stellt. Dies macht es derzeit leider noch etwas problematisch, dieses Beispiel zusammen mit dem Parent/Child-Builder einzusetzen.

Hinweis: Bestimmte Einträge in der OnMore-Methode kann man mit einer Abfrage auf den Übergabeparameterwert -1 klammern und somit nicht zur Verfügung stellen, wenn die Rechtsklickfunktion das Array abholt. Dann muß man allerdings mit einer Zählvariablen a la lnCounter=lnCounter+1 arbeiten und diese als Offset im Array benutzen, statt feste Arraydimensionen. Dies sollte man aber sowieso tun, weil man dann problemlos die Reihenfolge von Einträgen ändern kann (steht übrigens auch auf unserer Wunschliste…).

Zusätzlich ist es sinnvoll, das Rightclick-Event von Labels und Container an den Parent weiterzuleiten sowie von Pages an den Parent.Parent.Rightclick bzw. direkt an Thisform.Rightclick, damit bei diesen Elementen ebenfalls das Rechtsklickmenü des Formulars zur Verfügung steht.

Standardberichte nicht immer als Standard

Kommen wir zu einem weiteren ganz anders gelagerten Beispiel: Wenn man in der Eigenschaft .cReportName eines beliebigen VFX-Datenformulars einen Standardbericht einträgt, wird dieser IMMER verwendet und als Liste ausgedruckt. Damit steht die Selbsterstellungsmöglichkeit von Listen leider nicht mehr zur Verfügung. Manchmal möchte man aber gerne beides haben, einen Standardbericht und den VFX-Listendesigner.

Jetzt könnte man in der OnPrint-Methode einfach folgendes schreiben:

```
LPARAMETERS tlpreview
IF this.pgfpageframe.activepage # ;
                    this.npagelist
   this.creportname="meinsuperbericht"
ELSE
   this.creportname=""
ENDIF
RETURN DODEFAULT(m.tlpreview)
```

Wenn die Suchseite aktiv ist, erscheint daraufhin der Standard-VFX-Listendesigner. Auf allen anderen Seiten hingegen wird direkt der Standardbericht für das Formular gedruckt. Das verstehen sogar einfache Anwender. Damit hätten wir eine erste schnelle Lösung.

Nur für Entwickler ist das eigentlich nicht ganz so toll. Denn leider müsste man diesen Code in sämtliche Formulare in die OnPrint-Methode kopieren, welche einen Standardbericht zur Verfügung stellen sollen. Dies wäre für die Wartung sehr ärgerlich, insbesondere wenn man auf die zusätzliche Idee kommt, dass man auf den normalen Bearbeitungsseiten mit dem Standardbericht nur den aktuellen Datensatz ausdrucken möchte (z.B. Bestellung, Rechnung usw.).

Deshalb lösen wir das Problem über einen selbstdefinierten Hook, der auf das Ereignis OnPrint bzw. OnPostPrint lauert. Sie müssen allerdings am Ende der OnPostPrint-Methode der cDataForm-Klasse noch folgenden Hookaufruf einfügen:

```
*-- Ergänzung cDataForm.OnPrint:
*-- Zusätzlicher OnPostPrint-Hook
IF THISFORM.lusehook
       LOCAL luhookvalue
       luhookvalue = THISFORM.;
       oneventhook("OnPostPrint",;
       THIS,THISFORM)

       DO CASE
       CASE VARTYPE(luhookvalue)="L"
          IF !luhookvalue
             RETURN .T.
          ENDIF
       CASE VARTYPE(luhookvalue)="N"
          RETURN luhookvalue=0
       ENDCASE
ENDIF
```

In einem zukünftigen Build von Visual Extend wird dieser weitere Hook bereitstehen. Schauen Sie also erst nach, ob die obigen Zeilen in Ihrer Version nicht ohnehin enthalten sind. Falls nicht, ist das Hinzufügen kein Problem, da es in einem weiteren Build ja sowieso dazukommen wird…

Kombinierter On(Post)Print-Hook

Und hier der Quellcode für den eigentlichen Hook, der die gesamte Arbeit für uns übernimmt. Sofern in der VFX-Standardeigenschaft .cReportName ein Vorlagebericht definiert ist, wird dieser von Einzelseiten aus für den aktuellen Satz gedruckt; in der Übersicht hingegen steht weiterhin der VFX-Listendesigner zur Verfügung. Benötige Eigenschaften werden notfalls schnell zum laufenden Formular addiert:

```
CASE lcBaseclass == "FORM" AND ;
( tcevent == "ONPRINT" or ;
  tcevent == "ONPOSTPRINT" )

*-- Ausführung wenn man drucken darf und kann
IF NOT .lCanPrint OR .lEmpty
       lcontinue = .F.
ELSE
IF tcevent == "ONPRINT"

       *-- Erster Eventaufruf
*-- Fehlende Eigenschaften anlegen
       IF NOT Pemstatus( toObject, "cPrintReportName", 5)
          .addproperty( "cPrintReportName", .creportname )
       ENDIF
       IF NOT Pemstatus( toObject, "cPrintFilterSave", 5)
          .addproperty( "cPrintFilterSave", FILTER() )
       ENDIF

       *-- Druck aus Übersicht
*-- Defaultreport abschalten
       IF .pgfpageframe.activepage = .npagelist
          IF NOT EMPTY( .creportname )
             .cPrintReportName = .creportname
             .creportname = ""
          ENDIF
       ELSE

       *-- Druck aus Default
*-- Filter und ggf. Defaultreport setzen
          IF NOT EMPTY( .cPrintReportName )
             .creportname = .cPrintReportName
          ENDIF
          .cPrintFilterSave = FILTER()
          LOCAL lnRecno
          lnRecno = STR(RECNO())
          SET FILTER TO RECNO() = &lnRecno
       ENDIF

ELSE   && tcevent == "ONPOSTPRINT"

       *-- Nach Druck aus Übersicht
*-- Defaultreport wiederherstellen
       IF .pgfpageframe.activepage = .npagelist
          IF NOT EMPTY( .cPrintReportName )
             .creportname = .cPrintReportName
          ENDIF
       ELSE

       *-- Nach Druck aus Detail
*-- Filter wiederherstellen
          IF EMPTY( .cPrintFilterSave )
             SET FILTER TO
          ELSE
             LOCAL lcFilter
             lcFilter = .cPrintFilterSave
             .cPrintFilterSave = ""
             SET FILTER TO &lcFilter
          ENDIF
ENDIF
ENDIF
```

Wie Sie sehen, kann man mit einem etwas komplexeren aber immer noch nicht wirklich komplizierten Hook das Standardverhalten einer mit Visual Extend generierten Anwendung in ganz erheblichem Maße anpassen! Kopieren Sie einfach den obigen Code in Ihren Eventhandler und schon können auch Sie diese Funktionalität verwenden.

Comboboxen zurücksetzen

Comboboxen sind in Visual FoxPro stellenweise etwas trickreich. Als Vorgabe ist der Style=1 (Dropdown Combo) voreingestellt, der sowohl die Auswahl eines Wertes aus einer aufklappenden Liste oder die Neueingabe eines Wertes ermöglicht. Neu eingegebene Werte aber müssen dann in die Auswahlliste transferiert werden, damit sie bei der nächsten Anzeige des Elements auch noch sichtbar sind. Und wenn man den Style=2 (Dropdown List) einstellt, kann der Anwender nur noch Auswählen, was die Combobox unfreiwillig in ein Pflichtfeld wandelt, welches nie wieder ohne Wert sein kann, sofern man nicht in sämtlichen Listen einen Leerwert zusätzlich definiert, was sehr unpraktisch sein kann.

Deshalb wollen wir im Rechtsklickmenü der Combobox für diesen Fall eine zusätzliche Möglichkeit anbieten, den Wert zurückzusetzen. Dafür müssen wir in die Methode Rightclick der Basisklasse. Unser Fall macht natürlich nur bei Style=2 einen Sinn. Zusätzlich sollte aber eine selbstdefinierte Eigenschaft (hier lResetValue) abgefragt werden, da die Funktionalität bei Pflichtfeldern z.B. unerwünscht wäre.

Nach DEFINE BAR 5 / ON SELECTION BAR 5 fügen wir in der Rightclick-Methode folgenden Code ein:

```
WITH THIS
IF .style = 2 AND ;
       .lresetvalue = .T. AND ;
       NOT EMPTY(.Value ) AND ;
       NOT thisform.lempty AND ;
       thisform.lcanedit

DEFINE BAR 6 OF shortcut PROMPT "\-"
DEFINE BAR 7 OF shortcut ;
       PROMPT "Reset Value"
ON SELECTION BAR 5 OF shortcut ;
       lothis.REQUERY()
ON SELECTION BAR 7 OF shortcut ;
       lothis.resetvalue()
ENDIF
```

Und in die neue Methode Resetvalue an der Comboboxbasisklasse kommt dann folgender Codeabschnitt:

```
WITH THIS
.Value = ""
.DisplayValue = ""
.InteractiveChange()
ENDWITH
```

Wenn man sich im Gotfocus-Event in einer Eigenschaft den DisplayValue beim Betreten des Feldes in einer Eigenschaft merkt, kann man alternativ oder zusätzlich das zurücksetzen auf den ursprünglichen Wert anbieten.

Mit ein bißchen leider noch fehlender Verfeinerungsarbeit kann man den ersten Aufruf natürlich wiederum in einen Hook einbauen, der auf Combobx.Rightclick reagiert und statt einer Methode eine Funktion aufruft. Den zusätzlichen Schalter würde man aber vermutlich trotzdem benötigen.

Ort aus Postleitzahl vorbelegen

Soviel erstmal zu Hooks. Kommen wir zu einem weiteren typischen Beispiel, welches in praxisnahen Anwendungen immer wieder vorkommt. Wenn der Anwender eine Postleitzahl eintippt, kann man in vielen Fällen daraus problemlos den entsprechenden Ort ableiten ohne dass der Anwender diesen nochmals tippfehlerbehaftet eintippen müsste oder wirklich separate Steuertabellen benötigt werden. IntelliSense oder eine Combobox helfen dabei aber nicht wirklich weiter. Deshalb platziere ich im Valid-Ereignis der PLZ-Textbox folgenden Code:

```
WITH THIS
IF NOT EMPTY(.value )
WITH .Parent.<txtort>
       IF EMPTY( .Value )
          LOCAL lcValue
          lcValue = getcityforzip( ;
       <adressen.land>, THIS.Value )
          IF NOT EMPTY( lcValue )
              .value = lcValue
              .interactivechange()
          ENDIF
       ENDIF
ENDWITH
ENDIF
ENDWITH
```

Bei Eingabe einer Postleitzahl wird damit mit nachfolgender Funktion nach anderen Datensätzen gesucht, die als PLZ oder als Postfach-PLZ bereits die gleiche PLZ verwenden und einen Ortseintrag haben. Alle anzupassenden Werte stehen in eckigen Klammern.

```
FUNCTION getcityforzip
LPARAMETERS tcCountry, tcZip

LOCAL lnSelect, lcReturn, lcAlias
```

```
lnSelect = SELECT()
lcReturn = ""
tcCountry = ALLTRIM( tcCountry )
if EMPTY( tcCountry )
      tcCountry = <"Deutschland">
ENDIF
tcZip = ALLTRIM( tcZip )

IF NOT EMPTY( tcCountry ) ;
      AND NOT EMPTY( tcZip )
      lcAlias  = "findcity4zip"
      IF NOT USED( lcAlias )
         USE <adressen> ;
         ALIAS ( lcAlias ) ;
         IN 0 AGAIN SHARED
      ENDIF
      SELECT (lcAlias)
      LOCATE FOR <land> = tcCountry ;
         AND <plz> = tcZip AND ;
         NOT EMPTY( <ort> )
      IF FOUND()
         lcReturn = <ort>
      ELSE

         LOCATE FOR <land> = ;
            tcCountry AND :
            <plz_pf> = tcZip AND ;
            NOT EMPTY( <postfach> )
         IF FOUND()
            lcReturn = <postfach>
         ENDIF
      ENDIF
      SELECT (lnSelect)
      lcReturn = ALLTRIM( lcReturn )
ENDIF
RETURN lcReturn
ENDFUNC
```

Daraus könnte man natürlich auch einen vollständig parametrisierten Funktionsaufruf machen, der im Valid der PLZ-Textbox (oder PLZ-Postfach-Textbox) leicht aufgerufen werden kann. Das wäre allerdings für das Verständnis des Ablaufs wenig hilfreich. Ansonsten benötigt die obige Funktion das Framework VFX nicht sondern kann generell eingesetzt werden.

VFX-PDF-Installation ohne Internet

Zurück zu VFX und zum Thema Berichte: Um die PDF-Ausgabe von Visual Extend auch bei Anwendern installieren zu können, die keinen Internetzugang haben, so dass die automatische Installation nicht funktioniert, gibt es einfache Möglichkeit. Man kann in das Verzeichnis von SYS(2023), normalerweise C:\Dokumente und Einstellungen\ <Username> \Lokale Einstellungen\ Temp, einfach die entsprechende Installationsdatei von Ghostscript unter ftp:// mirror.cs.wisc.edu/pub/mirrors/ghost/AFPL/gs814/gs814w32.exe kopieren. Sofern die Datei dort vorhanden ist, wird seitens VFX der automatische Internetdownload gar nicht erst versucht. Dies beschleunigt natürlich die Installation auch wenn ein Internetzugang tatsächlich verfügbar wäre.

VFX-Berichte versenden

Sofern ein unterstützer Faxtreiber vorhanden ist, kann man mit folgendem Pseudocode aus VFX heraus einen Bericht an eine Faxnummer versenden:

```
loFax = NEWOBJECT("cFax")
loFax.SendFax(<alias>,;
       <reportname>,<faxnummer>)
```

Dabei wird für FritzFax ein temporärer Registry-Eintrag mit der Telefonnummer erzeugt und WinFax wird per OLE angesteuert, damit die jeweilige Faxsoftware nicht erneut in einem Dialog nach der Empfängernummer fragt.

Sofern die PDF-Unterstützung konfiguriert ist, können Sie auch einen Bericht per eMail versenden. Leider muss man da etwas mehr wissen als beim reinen Faxversand. Hier ebenfalls nur Pseudocode für das Prinzip und die Ausführung innerhalb von VFX:

```
*-- Deklarationen
lnSelect=select()
lcDefault=GetDefaultFolder()
lcreportname = <berichtsdatei>+".frx"
lcAlias = Alias()
loEmail = CreateObject("CEmail")

*-- Maildetailsabfrage
DO Form vfxEmailDetails WITH ,,loEmail to oEmailDetails
lcEmail = oEmailDetails.cEmail
```

```
lcSubject = oEmailDetails.cSubject
lcText = oEmailDetails.cText
lcCCRecipient = oEmailDetails.cCCRecipient
lcBCCRecipient = oEmailDetails.cBCCRecipient
lcFor = []    && ggf. Bericht filtern..

*-- Berichtserstellung und Versand
lcFileName = ForceExt(lcreportname, "pdf")
loEmail.AddAttachment(lcAlias, lcFileName, lcreportname, lcFor)
loEmail.send_email_report(lcEmail, lcSubject, lcText, lcCCRecipient,
lcBCCRecipient)

*-- Aufräumen
SELECT(lnSelect)
SET DEFAULT TO (lcDefault)
```

Weitere Details zur Umsetzung finden Sie in der Methode OnPrint in der Klasse cDataForm.

Komprimieren der VFX-Messagetabelle

Sofern man in einer Anwendung nur einsprachig oder mit wenigen Sprachen arbeitet, und seine Projekt-Entwicklungsumgebung sichern oder versenden möchte, empfiehlt sich das Komprimieren der VFXMSG.DBF, da diese eine nicht unbeträchtliche Größe hat. Am einfachsten löscht man den Inhalt der nicht benötigten Sprachspalten mit einem einfachen Replace-Befehl a la

```
REPLACE ALL esp with "", fre with "",;
       ita with "", bul with "", ;
       gre with "", cze with "", ;
       nl with "" , por with "", ;
       ru with "" , fin with "", ;
       pl with "" && Türkisch fehlt!
```

und führt danach ein PACK MEMO aus. Bei der Installation eines neuen Builds von VFX werden neue Messagetexte natürlich wieder für alle Sprachspalten übernommen und man muss die entsprechenden Befehle erneut absetzen.

Weglassen von VFX-Komponenten

Was man unter VFX unbedingt vor einer Auslieferung der eigentlichen Anwendung tun sollte aber gelegentlich gerne vergisst:

- ZAPen der VFXLOG.DBF, denn der Kunde soll ja nicht alle unsere Fehlereinträge vom Testen sehen. Außerdem wächst die Datei bei Einschaltung der kompletten Protokollierung (siehe VFX Application Wizard/Builder) recht schnell an!
- ZAPen des AuditTrails in VFXAUDIT.DBF, sofern dieses über den VFX AuditTrigger-Wizard eingeschaltet wurde. Den Anwender interessieren unsere Testdatenänderungen eher weniger. Ohne Verwendung des AuditTrails kann man die Tabelle auch ganz weglassen.
- ZAPen der VFXUSERLOG.DBF, da das Protokoll der Useranmeldungen bei der Auslieferung ebenfalls leer sein sollte.
- ZAPen der VFXRES.DBF, da die bisherigen Voreinstellungen nicht mit ausgeliefert zu werden brauchen.
- Weglassen von VFXHELP.DBF, da diese nur in der Entwicklungsumgebung benötigt werden.
- Weglassen von VFXMSG.DBF, sofern keine Laufzeitlokalisierung verwendet wird. In letzterem Falle sollte die Tabelle inkludiert werden und nicht separat mitgeliefert.
- Weglassen von VFXPATH.DBF und ausschliessen der Formulare VFXCLIEN und VFXPATH, sofern man nicht mit Mandanten arbeitet.
- Weglassen von VFXREP.DBF, VFXRTEMP.DBF und VFXRTYPE.DBF, sofern man nicht mit der Klasse CRSELECTION zur Berichtsauswahl aus VFXREP.VCX arbeitet.
- Sofern man die Listendefinitionen nicht verwendet, kann man die Tabellen VFXPDEF und VFXPLIST sowie das Formular VFXPLIST ausschließen. Der Menüpunkt „Extras->Verwaltung von

Auswahllisten“ ist natürlich zu löschen. Bringt aber nicht viel.

- Sofern man die Toolbox nicht verwendet, kann man die Tabellen VFXTOOLBOX und VFXTOOLTYPE sowie 6 Formulare VFXTOOL* und die dazugehörige Klassenbibliothek ausschließen. Der Menüpunkt „Extras->Werkzeugkasten“ ist natürlich zu löschen.
- Sofern man die Fernwartung nicht verwenden möchte, kann man die Memofelder in Datensätzen mit dem Typ RADMIN in VFXINTERNFILES.DBF löschen.
- Sofern man keine Aktivierungsschlüssel verwenden möchte, kann man die Formulare VFXREGISTER UND VFXREQUESTREPLACEMENTKEY ausschliessen.
- Sofern man den neuen Berichtsgenerator von VFP 9.0 nicht verwendet, kann man die Berichts-Apps weglassen sowie die Klassenbibliotheken _FRXCURSOR und _REPORTLISTENER ausschliessen
- Sofern man keine Client/Server-Anwendung erstellt und auch nicht mit CursorAdaptor-Klassen arbeitet (auch nicht auf VFP-Tabellen), könnte man darüberhinaus einige Formulare ausschliessen (vfxclientdataaccess, vfxcreateconnstr, vfxdun, vfxgetdbconnection, vfxservers, vfxsqlstr,

Da es aber sehr häufig vorkommt, dass man zu einem späteren Zeitpunkt doch wieder entsprechende Funktionalitäten in der Anwendung aktivieren möchte, empfehlen wir einfach das Zappen bzw. Weglassen der oben aufgeführten Tabellen.

Installation von VFX-Anwendungen

Mit dem RuntimeInstaller der dFPUG kann man die Laufzeitumgebung für Visual FoxPro problemlos installieren. Dazu kommen dann noch die lokalisierten Applikationen für Berichtsausgabe, Vorschau und Bearbeitung. Damit wäre alles bis auf die eigentliche Anwendung und die Mitlieferung der VFX.FLL soweit vorbereitet, aber sofern man alle Möglichkeiten von Visual Extend in seiner Applikation verwendet, kommen dann doch noch ein paar weitere Dateien hinzu:

- TreeView - MSComctl.ocx (v6.0 SP6) für Treeview-Formulare
- ImageList - MSComctl.ocx (v6.0 SP6) (Microsoft Windows Common Controls 6.0) für z.B. die Favoritenanzeige
- Month View - MSComct2.ocx (v6.0 SP6) für die Kalenderanzeige in Datumsfeldern
- SOAP - mssoap30.dll (v3.0) für die Verarbeitung von XML
- MSMAPI.ocx (es sollte jede Version gehen) für die Mailansteuerung
- MSCHRT20.ocx für die Businessgrafiken

Diese Dateien sind zwar meist auf den Kundenrechnern vorhanden, aber dessen kann man sich nicht sicher sein und kommt deshalb um eine entsprechende Installationsroutine nicht herum.

Zusatzdateien für Hilfeaufruf

Sofern man mit dem VFX Help Wizard seine Helpcontext-Ids in allen Formularen gesetzt und aus der generierten und gefüllten VFXHELP.DBF eine Hilfedatei über den HelpWorkshop erstellt hat, liegt zwar eine schön .CHM-Datei vor, aber leider benötigt die generierte Anwendung noch die Dateien Foxhhelp9.exe und foxhhelpps9.dll (gleiches gilt für die Version 8), um die Hilfe aufrufen zu können. Da unter Windows XP der Suchen-Dialog nicht automatisch alle Verzeichnisse durchsucht, schauen Sie am besten direkt unter „C:\Programme\ Gemeinsame Dateien\ Microsoft Shared\ VFP“ nach zwecks Mitlieferung. Dort finden sich übrigens auch die verschiedenen Runtimebibliotheken und die gdiplus.dll. Bitte beachten Sie, dass die beiden Dateien entsprechend registriert werden müssen (regsvr32 foxhhelpps9.dll bzw. foxhhelp9.exe /regserver).

Berichte exkludieren

Sofern man Anwendern Berichte für die Bearbeitung freischalten oder aktualisierte Berichte ohne Gesamtauslieferung der kompilierten Anwendung versenden möchte, muß man die Berichtsdateien in der Projektdatei

via rechte Maustaste exkludieren. Zumindest aus meiner Sicht „leider“ werden Berichte defaultmäßig mit in die EXE einbezogen und blähen diese wahnsinnig auf. Am Stück ändern kann man das mit folgenden zwei Zeilen:

```
use <project>.pjx
replace all exlude with .T. ;
       for type = "R"
```

Das Verfahren kann man natürlich auch auf alle anderen Arten von Einträgen in Projektdateien anwenden und es ist nicht spezifisch für Visual Extend.

In Metadateien suchen

Und das gilt natürlich für alle weiteren FoxPro-Metadateien wie Formulare (.scx), Berichte (.frx), Menüs (.mnx) und so weiter, da es ja auch alles nur DBF-Tabellen sind. Um zum Beispiel einen vom Projekt als fehlerhaft gemeldeten Eintrag zu suchen, braucht man nur mit USE die Projektdatei öffnen und mit GOTO auf den entsprechenden Datensatz gehen.

oder bestimmte Wertbelegungen zu ändern. Elemente mit bestimmten Werten in Eigenschaften hingegen sucht man am besten wie folgt:

```
use <formular>.scx
browse for ;
       '<Eigenschaft>="<Wertanfang>' ;
       $ properties
```

Sofern man Klassen umbenennt und über das Codereferenztool nach weiterem Vorkommen des alten Namens zwecks Auffinden von abgeleiteten Klassen sucht, wird man um das Hacken der Metadateien von VFP nicht herumkommen. Im Codereferenztool findet man die Klasse nämlich, aber im Eigenschaftsfenster wird witzigerweise der neue Name angezeigt, obwohl im Namensfeld der Metatabelle immer noch der ursprüngliche Klassenname drinsteht!

VFX-INI-Datei und VFX-Splash

Zwecks Erhalt des Microsoft-Logos „Verified for Windows XP“ war es notwendig verschiedene Speicherpfade von Dateien zu ändern, so auch der INI-Datei von Visual Extend. Diese findet man nunmehr unter „C:\Dokumente und Einstellungen\ All Users\ Anwendungsdaten\ dFPUG\ Visual Extend\ 9.0“.

Dort kann man übrigens unter [VFX] mit dem Eintrag SPLASHSCREEN=NO den Startbildschirm von VFX einfach abschalten – nur falls es jemanden stören sollte.