Inhalt

Dieses Dokument beschreibt Abläufe und Funktionen von Programmteilen, die anlässlich des VFX Usertreffens vom 15. Mai 2009 vorgestellt und demonstriert worden sind.

©Author: Fritz Maurhofer, Maurhofer Informatik AG
murmi@maurhofer-informatik.ch
www.maurhofer-informatik.ch

1. Druckersteuerung perfektioniert

Die Internas der Druckersteuerung wurden bereits am VFX Anwendertreffen vom 19. Mai 2006 vorgestellt. Das entsprechende Dokument AnatomieReportSteuerung.pdf ist nach wie vor im dFPUG Portal verfügbar.

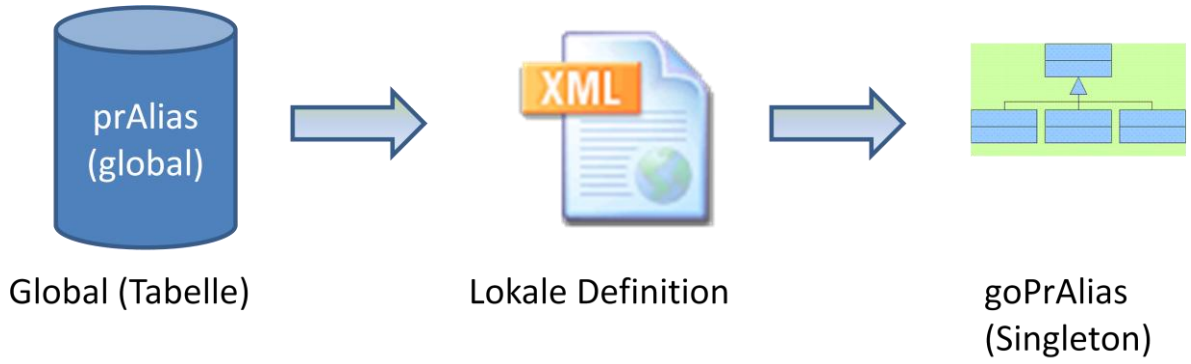
Es geht nun hier darum, dass Berichte, die auf speziellen Druckern oder sich aus Papier in bestimmten Schächten bedienen, nun nicht mehr direkt einem Drucker zugeordnet sind. Sondern je nach Benutzer und Standort auf den entsprechenden Geräten die sich z.B. in der Nähe befinden. Dabei muss berücksichtigt werden, dass sich der Papiervordruck beim Drucker im Erdgeschoss in Schacht 2, bei dem im Dachgeschoss aber in Schacht 1 befindet.

Weiter ist in der Zwischenzeit Windows Vista ein Thema, das uns freundlicherweise bei der Funktion SYS(1037) eine Schaltfläche „geklaut" hat. Mit Fox Bordmitteln ist es nun nicht mehr möglich innerhalb einer Applikation den Drucker mit seinen Eigenschaften einzustellen, damit diese Werte z.B. mit PRTINFO() abzufragen. Mit den in VFX nun enthaltenen Features kann aber auch dieses Problem behoben werden.

1.1. Konzept – Drucker Alias

Grundprinzip ist, dass nun bei einem Bericht nicht festgehalten wird „drucke auf HP 4711" sondern es wird ein Alias Begriff zugewiesen wie „Rechnung", „Briefdokument" etc.

Der Administrator definiert nun eine Applikationsweit gültige Alias Definition, die im einfachsten Fall einen Drucker bezeichnet, bei Berichten, die fixe Formulare verwenden sollen, sprechende Alias-Namen wie eben „Rechnung".



prAlias	Tabelle, die in der Applikationsdatenbank enthalten ist und die alle verwendbaren Aliasse enthält. Kann/sollte nur durch den Administrator erstellt und geändert werden.
Lokale Definition	Die Definitionsdatei, die sich im Profil des Users, bzw. lokalen Rechners befindet und die bei einem Alias abweichende Druckerangaben (Drucker/Schacht) enthalten können. Mit CursorToXML erstellte Datei.

goPrAlias Beim Programmstart erzeugtes Singleton, dass
a) die aktuell gültige „Übersetzungstabelle“ (Alias → Drucker) enthält und
b) die in diesem Kontext notwendigen Methoden um das Objekt zu benützen, editieren und zu speichern.

Bei einem Programmstart werden folgende Aktionen durchgeführt:

- Bestimmen des Pfades mit der lokalen Definitionsdatei
- Erzeugen des Objektes goPrAlias
  - Globale Alias laden
  - Eine lokale Definitionsdatei existiert
    - Laden der lokalen Definition
    - Sofern neue globale Alias Definitionen noch nicht vorhanden sind, werden diese hinzugefügt
  - Eine lokale Definitionsdatei existiert nicht
    - Auf Basis der globlaen Alias wird eine lokale Definitionsdatei erstellt

Wenn nun ein Bericht gedruckt werden soll, dann wird die aktuell gültige Zuordnung Alias→Drucker aufgelöst und dem Benutzer, der Benutzerin angezeigt, auf welchen Drucker/Schacht die Ausgabe erfolgen wird. Sofern dem Bericht kein Alias zugeordnet ist, wird der aktuell VFP Drucker/Schacht verwendet.

1.1.1. Beispiele

Instanziieren des goPrAlias beim Programmstart:

```
* Objekt für lokalen PrinterAlias
PUBLIC goPrAlias
goPrAlias = CREATEOBJECT("cLocPrAlias", ;
                         goProgram.oConnMgr.getConnection(), ;
                         goProgram.cLocalSetDir)
```

cLocPrAlias Diese Klasse finden Sie in \Lib\VFXTreff.vcx

.getConnection Wird in der Klasse z.Zt. nicht benützt, da die globalen Alias mit einem CursorAdaptor geladen werden.

.cLocalSetDir Applikationseigenschaft, die auf das Ressourcen-Verzeichnis verweist. Die Datei heisst fix LocalPrAlias.xml

LocalPrAlias.scx Dieses Formular (in \FORM) dient zum bearbeiten der lokalen Alias

prAlias Die Tabellenstruktur für die globalen und lokalen Printer Alias finden Sie im Anhang 3.1

1.2. Hasta la Vista

Oder als etwas bösartige Übersetzung: wer hat den Durchblick!

Bisher (d.h. bis XP) konnte man ja problemlos SYS(1037) aufrufen, mit der Druckerschaltfläche alles einstellen und das ganze dann mit PRTINFO() im gewünschten Detaillierungsgrad anschauen. Da man uns nun diese Schaltfläche „geklaut" hat, können einem auch die anderen Bordmittel

GETPRINTER()
APRINTERS( )
SYS(1500, "_mfi_sysprint", "_mfile")

nicht wirklich und vor allem nicht vollständig weiter.

Als weitere Möglichkeit wäre API-Calls zu nennen, mit denen man den Druckerdialog aufrufen kann. Der Funktionen sind viele, die aber aus VFP nicht gerade einfach zu handhaben sind.

Zum Glück ist in VFX Rettung eingebaut:

- Vfx.fll
- vfxPrintDialog.scx

VFX-Berichte benützen standardmässig diese Funktionalität und sie kann auch relativ einfach für eigene Zwecke benützt werden wie das folgende, stark reduzierte, Beispiel zeigt:

```
SET LIBRARY TO vfx.fll ADDITIVE
* Parameterobjekt erstellen
loPrinterObj = fmGetprinterObj()

DO FORM vfxPrintDialog WITH loPrinterObj

IF loPrinterObj.lReturnType
   * OK knopf wurde gedrückt
   * das Parameterobjekt ist bereits befüllt
   llRetVal = .T.
ENDIF
```

Ein nicht zu unterschätzendes Goodie noch gegenüber SYS(1037): beim vfxPrintDialog weiss man zuverlässig, ob der Benutzer OK oder Abbruch gedrückt hat!

Wenn nun beim vfxPrintDialog die OK Schaltfläche gedrückt wird, dann wird das Parameter Objekt mit den entsprechenden Werten befüllt. Das von VFX verwendete Objekt habe ich um ein paar Eigenschaften erweitert, damit wirklich alle vorhandenen Parameter verfügbar sind.

1.2.1. Beispiele

fmGetPrinterObj	Erstellt das Parameterobjekt
fmGetPrinterProps	gekapselter Aufruf des vfxPrinter-Dialoges
vfxPrintDialog.scx	Zusätzlicher Code in cmdOk.Class()

2. InBox

Moderne Scanner (insbesonder auch low cost All-in-One Geräten) können scans gemäss Job-Definitionen in bestimmten Verzeichnissen ablegen. Der Zugriff auf die so erzeugten Dokumente sollte innerhalb der Applikation auf einfache Art möglich gemacht werden.

Die Inbox soll einerseits als normales Formular zur Verfügung stehen (in screen), aber auch als top level Form, z.B. in einer Multi Monitor Umgebung.

Schwierigkeit: ein Formular kann nicht zur Laufzeit von „in screen“ auf „toplevel“ umgestellt werden und vice versa.

Lösung: zwei Klassen

- cInbox mit allen Eigenschaften und Methoden mit ShowWindow = 0 (in screen)
- cInboxTopLevel als Subklasse mit einer Abweichung: ShowWindow = 2 (as toplevel)

Das ganze wird über ein Programm gemäss übergebenem Parameter gestartet:

```
LPARAMETERS tcStartMode

LOCAL lnShowWindow, llGoOn, lcClass
lnShowWindow = 0
llGoOn       = .F.

IF ATC(tcStartMode,"L") > 0
   * in Screen
   lnShwoWindow = 0
   lcClass      = "cInbox"
ELSE
   * TopLevel
   lnShowWindow = 2
   lcClass      = "cInboxTopLevel"
ENDIF
```

```
IF VARTYPE(goProgram.oInbox) = "O"
   IF goProgram.oInbox.ShowWindow = lnShowWindow
      * bereits im richtigen Modus vorhanden
      goProgram.oInbox.show()
   ELSE
      * im aktuellen Modus beenden
      goProgram.oInbox.release()
      llGoOn = .T.
   ENDIF
ELSE
   * auf jeden Fall neu
   llGoOn = .T.
ENDIF

IF llGoOn
   goProgram.oInbox = CREATEOBJECT(lcClass, goProgram.cInboxPath)
   goProgram.oInbox.show()
ENDIF
```

Im Release Event des Formulares wird die Referenz in goProgram.oInbox jeweils wieder auf NULL gestellt.

Weitere Besonderheiten der Inbox: Die Daten werden mit dem Treeview Control von DBI angzeigt. Dieses ermöglich auf einfach Art, Daten spaltenweise darzustellen. Mit diversen Methoden kann die Tabelle einfach definiert werden:

```
.addColumn()
.ColumnDataType[]
.ColumnLock[]
.ColumnPicture[]
.ColumnPictureAlign[]
.ColumnSortable[]
.ColumnTextAlign[]
.aTreeColumns[)
.ColumnBackColor[]
.ColumnBackColor[]
.ColumnCheckBox[]
.ColumnCheckAlign[]
```

Anschliessend können mit der Methode `.doAddNode()` bei jeder Zeile die Wert der Spalten mit Semikolon getrennt übergeben werden.

Per Definition enthält das Inbox-Verzeichnis nur eine Stufe Unterverzeichnisse und nur darin sind die Dateien enthalten. Dies beschränkt den Aufbau des Trees auf zwei Stufen und macht das ganze ein bisschen einfacher. Auf eine Eigenheit des DBI-Treevies sei auch noch hingewiesen: die NodeCargo-Eigenschaft. Es handelt sich um einen nicht sichtbaren Text, dem wir einen beliebigen Wert zuweisen können. In unserem Fall den vollen Pfad und Dateinamen der zum Node gehörenden Datei. Da wir bei einer beliebigen Funktion jeweils den Index des Nodes kennen, kann der Dateiname mit einem einzigen Zugriff ermittelt werden:

```
lcFile = this.ocxInbox.NodeCargo(lnNode)
```

Eine kleine, aber etwas fiese Eigenheit hat das Control beim Drag/Drop. Startet man den Dragvorgang dann wird als Drag Symbol das ganze Control mitgezogen und nicht nur die gewählte Zeile. Wenn mann's weiss, ist einfach Remedur zu schaffen: Der Eigenschaft DragIcon ist schlicht ein Cursor-File zuzuordnen (z.B. dragmove.cur) und schon zeigt sich das erwartete Verhalten. Thanks to Oliver Hunziker für diesen Tipp!

Drag/Drop aus dem DBI-Treeview in ein FoxPro Formular wird wie folgt implementiert:

```
StartDragOut Event
*** ActiveX Control Event ***
this.Drag(1) && begin drag
```

Obschon der Drag aus einem ActiveX Control heraus gestartet wird, handelt es sich um einen FoxPro Drag/Drop und nicht um einen OleDragDrop. Im DragDrop des Grid, den wir als Drop-Ziel zulassen, haben wir nun eine Referenz auf das „liefernde" Objekt und können daraus (im Beispielfall gemäss Namenskonvention) die Referenz des Host Formulares ermitteln:

```
loSourceForm = oSource.parent
```

Diese Referenz ergeben wir einer Methode, die

beim Ausgangspunkt die notwendigen Angaben holt	`lcSrcFile = loSourceForm.dogetDragInfo()`
die Speicherung gem. Usanzen im aktuellen Form veranlasst	`onInsert() ….. onSave()`
und im Erfolgsfall das Aufräumen der Inbox veranlasst	`toSourceForm.doPostDrag()`

In der Inbox kann der oder die BenutzerIn den Explorer starten um ggf. „Irrläufer" aufzustöbern. Gewünscht wurde, dass aber nicht so einfach über das Inbox-Stammverzeichnis hinauf gesurft werden kann. Mit dem Parameter /root verhindert der Explorer genau das:

```
lcPara = "/root, " + lcPath
fmOpenFile("explorer.exe", lcPara, lcPath, "open")
```

Die Funktion fmOpenFile kapselt den SHELLEXECUTE Befehl und ist bei den Beispielen im Verzeichnis Program vorhanden.

2.1. Dokumente

2.1.1. PDF drucken

Dem Acrobat kann (m.Wissens ab Version 6 oder 7) der zu verwendende Drucker als Parameter mitgegeben werden:

```
LPARAMETERS toPrinterObj, tcFile, toForm
* Para's passed: <ExpO1> Printer-Objekt
*                <ExpC>  zu druckendes PDF (Pfad + Dateiname)
*                <ExpO2> Referenz a/aufrufendes Form
* Para returned: Nothing, bzw. immer .T.
* 26.03.2009/murmi

LOCAL lcFile, lcParam

* welches Acro-Exe ist zu verwenden:
lcFile  = "AcroRd32.exe"
lcParam = [/t "] + tcFile + [" "] + toPrinterObj.cPrinterName + ["]

fmOpenFile(lcFile, lcParam, "", "print")

RETURN
```

2.1.2. fmOffice

Die Word und Excel Klasse wurden um einige Funktionen erweitert:

fmOffice	Klassenbibliothek im Verzeichnis LIB
fmWord	
.find	Words Find/Replace Funktion vollständig umgesetzt
.openDocument	Erweitert um die ReadOnly und Passwort Variante
.openWord	Etwas sicherer gemacht…
fmExcel	
.activateExcel	Excel im gewünschten Status aktivieren, bzw. sichtbar machen
.openExcel	Etwas sicherer gemacht…
.openWorkbook	Ebenfalls um die Möglichkeiten ReadOnly und Passwort erweitert
.replace	Search/Replace vollständig umgesetzt

NB: include\usertxt.h enthält die zusätzlichen Konstanten für Word und Excel.

3. Anhang

3.1. Tabelle prAlias

Feldname	Typ	Länge	Dez.	Null	Bemerkungen
NPRALIASID	I	4		N	Key
CALIASNAME	C	50		J	Alias-Name (z.B. Rechnung)
CPRNAME	C	80		J	Druckername gem. Windows
CPRLOCAT	C	80		J	Ort gem. Windows
CPRCOMMENT	C	80		J	Kommentar gem. Windows
NLOCTRAY	I	4		J	Nr. des zu verwendenden Schachtes

3.2. Etwas Code…

fmCopyFile — Kopiert, im Gegensatz zum internen Copy File auch geöffnete Dateien

fmFormCatchErr — In Try/Catch kann Catch ein Fehlerobjekt erzeugen. Mit dieser Funktion kann daraus eine nett formatierter Text für eine Fehlermeldung erstellt werden.

Beispiel:

```
CATCH TO loError
    MESSAGEBOX("Fehler: " +;
                fmFormCatchErr(loError,2))
```

fmGetCharVal — Für SPT müssen ja alle Werte in Strings umgewandelt werden. Die Funktin nimmt sich dieses Problems an, insbesondere auch der Falle, wenn in einem Textstring Apostrophe vorhanden sind (z.B. `cName = 'O'Hara'`)

fmSQLInf — Funktion, mit der in einer C/S Umgebung jederzeit ein beliebiges SQL-Statement abgesetzt und ein bestimmter Wert zurückverlangt werden kann. Besonders hilfreich auch in Reports, wenn eine bestimmte Feldinformation nicht im Resultat set vorhanden ist, wohl aber der Key-Wert, um diese Information zu ermitteln.

fmVFXUsrRights — Ermittelt eine bestimmte Berechtigung (gem. vfxgrouprights) ausserhalb des VFX Automatismus

Stru2Clip — Formatiert die Tabellenstruktur des im aktuellen Arbeitsbereich geöffneten Alias und kopiert sie anschliessend ins Clipboard. In Wort kann dieser Textteil mit „Test in Tabelle umwandeln“ zu Dokumentationszwecken verwendet werden
(siehe Beispiel in 3.1)