Anwendungsentwicklung am Beispiel einer Software für Pflegedienste

In meinem Vortrag zur dBKON 2002 habe ich bereits einige Grundzüge meiner Softwareentwicklung dargestellt. Die dort enthaltenen Komponenten sind nach wie vor aktuell und auch in den neuesten Versionen wieder auf der Konferenz-CD enthalten.
Dies sind u.a.

DBCmd	Kommandointerpreter
DBFView	Tabellenbetrachter
KillNull	Modul zum Setzen von definierten Feldwerten
LDrivers	Anpassung der Sprachtreiber von Tabellen an den globalen Sprachtreiber
Memofix	Überprüfung und Reparatur von Dateistrukturen

Im ersten Teil des Beitrags werden einige Möglichkeiten zur Programmierung anhand konkreter Quellcodebeispiele dargestellt.

Im zweiten Teil wird eine Alternative zur in dBASE integrierten Report-Engine aufgezeigt.

Grundsätze der Anwendungsentwicklung

Eine umfangreiche Anwendung besteht sollte immer im MDI-Design gestaltet werden. Dabei wird zunächst ein Startprogramm benötigt. Dies ist die Routine, die vom Befehlsfenster mit DO aufgerufen oder vom Regiezentrum per Doppelklick gestartet wird. Außerdem enthält diese Datei den Einstiegspunkt beim Aufruf der fertigen Anwendung (EXE).
Anmerkung: Der Befehl DO sollte nur noch zum Aufruf von einzelnen Modulen bei der Anwendungsentwicklung verwendet werden. In einem fertigen Programm hat er nichts zu suchen!

Im Startprogramm werden nur die globalen Bestandteile der Anwendung angelegt. Dies ist das Anwendungsobjekt selbst, die (Haupt-)Menüleiste und die (Haupt-)Toolbar. Dabei ist es zweckmäßig, für die letztgenannten Komponenten jeweils separate Quellcodedateien vorzusehen.
Ein Beispiel für eine einfache MDI-Anwendung ist auf der CD unter SimpleMDI enthalten.

Diese Struktur bildet die Basis für den weiteren Aufbau der MDI-Anwendung. Das Anwendungsfenster enthält eine Menüleiste und eine Toolbar. In der Menüleiste sind die Menüs Datei, Bearbeiten und Fenster angelegt. Die Funktionalität dieser Menüs wird automatisch von dBASE übernommen.
Beim Klick auf das Schließen-Symbol der Anwendung wird man feststellen, dass sich das Programm nicht mehr auf diese Weise schließen lässt. Dies wird über ein verborgenes Formular erreicht, dessen canClose-Handler ein *false* zurückgibt, solange die (selbstdefinierte) Eigenschaft `_app.ShutDown` den Wert *false* hat.

```
CLASS DummyForm OF FORM
   with(this)
      visible=false
      canClose = CLASS::CANCLOSE
   endwith

   FUNCTION canClose
   LOCAL lClose
   lClose=_app.ShutDown
   RETURN lClose
ENDCLASS
```

Wenn der Benutzer den ‚regulären' Weg über den Menüeintrag Datei/Beenden oder die entsprechende Funktion in der Toolbar wählt, wird `_app.ShutDown` auf *true* gesetzt und die Anwendung verlassen. Mit diesem Weg wird ein definiertes Schließen der Applikation erzwungen.

Grundsätzlich sollten alle Formulare einer Anwendung als MDI-Formulare vorgesehen werden. Die Eigenschaft mdi ist nur dann auf false zu setzen, wenn der Benutzer zielgerichtet den aktuellen Dialog nicht verlassen darf, andernfalls würde der Benutzer in seiner Handlungsfähigkeit zu sehr eingeschränkt.
Solange kein Non-MDI-Formular aktiv ist, kann der Benutzer auf alle verfügbaren MDI-Formualre zugreifen, diese sogar mehrfach öffnen und beliebig zwischen den Formularen wechseln. Dies bedarf weiterer Überlegungen beim Anwendungsdesign.

Oft ist es zweckmäßig, auch hier Beschränkungen aufzuerlegen. Im vorliegenden Beispiel wird dies anhand von zwei Formularen demonstriert. Während das Formular `AloneForm` nur einmal geöffnet werden kann, kann das Formular `MultiForm` bis zu fünfmal geöffnet werden.
Diese Beschränkungen werden durch folgenden Code erreicht:

```
FUNCTION OpenAloneForm
LOCAL f
f=FINDINSTANCE("AloneForm")
IF f=NULL
   SET PROCEDURE TO Alone.WFM ADDITIVE
   f=new AloneForm()
   f.app=this
   f.Init() //Formular vor dem Öffnen IMMER initialisieren
   f.open()
ELSE
   f.windowState=0
   f.setFocus()
ENDIF
RETURN

FUNCTION OpenMultiForm
LOCAL f
f=FINDINSTANCE("MultiForm")
IF f=NULL .OR. this.MultiCounter<5
   this.MultiCounter++
   SET PROCEDURE TO Multi.WFM ADDITIVE
   f=new MultiForm()
   f.app=this
   f.Init() //Formular vor dem Öffnen IMMER initialisieren
   f.open()
ELSE
   MSGBOX("Dieses Formular kann nur 5 mal geöffnet werden","Maximale Formularanzahl
erreicht",16)
ENDIF
RETURN
```

In beiden Aufrufen wird zunächst mit FINDINSTANCE geprüft, ob das Formular bereits im Speicher steht (geöffnet ist). In der ersten Funktion wird das Formular angelegt, wenn es nicht vorhanden ist, andernfalls wird es auf Normalgröße gebracht (windowstate=0) und der Focus gesetzt.
In der zweiten Funktion wird das Formular angelegt, wenn es noch nicht vorhanden ist oder der Zähler noch nicht seinen Maximalwert erreicht hat. Beim jedem Anlegen eines Formulars wird der Zähler erhöht. Ist die Maximalanzahl erreicht, erfolgt eine Meldung. Der Benutzer kann zur Auswahl des gewünschten Formulars in diesem Falle das Menü Fenster nutzen.

Dialogprogrammierung

Grundsätzlich empfehle ich, ein Formular mit einem Notebook auszustatten. Die erste Seite kann hierbei eine Übersicht der Daten in Grid-Form enthalten, die weiteren Seiten enthalten die Details.
Bis auf einige Ausnahmen lasse ich die direkte Editierung von Daten im Grid der Übersichtsseite nicht zu. Zur schnellen Suche werden für die hauptsächlisten Kriterien Indexeinträge verwendet. Zum Umschalten wird ein Doppelklick auf den Spaltenkopf ausgeführt. Zusätzlich besteht die Möglichkeit zur Schnellsuche im aktuellen Index nach Klicken mit der rechten Maustaste.
Per Doppelklick auf einen Eintrag gelangt man auf die zweite Formularseite. Grundsätzlich wird für jeden Dialog ein kontextsensitives Popup-Menü angeboten, in dem die wichtigsten (momentan verfügbaren) Funktionen des jeweiligen Dialogs sofort aufgerufen werden können.

Wie funktioniert das nun im Einzelnen?
Für alle Spalten, die das Umschalten des Index bewirken, wird der onLeftDblClick-Handler des Spaltenkopfes definiert.

```
FUNCTION HEADING_NAME_onLeftDblClick(flags,col,row)
form.rowset.indexname="NAME"
CLASS::SetColors()
RETURN

FUNCTION SetColors
form.Notebook1.Grid1.columns[<Spaltenname 1>].headingControl.colorNormal=APP_INDEXVORHANDEN
...
form.Notebook1.Grid1.columns[<Spaltenname n>].headingControl.colorNormal=APP_INDEXVORHANDEN
DO CASE
   CASE form.rowset.indexname="NAME"
     form.Notebook1.Grid1.columns[<Spaltenname 1>].headingControl.colorNormal=APP_INDEXAKTIV
ENDCASE
RETURN
```

Das Grid der 1. Formularseite erhält folgenden onRightMouseDown-Handler:

```
FUNCTION GRID1_onRightMouseDown(flags,col,row)
IF BITSET(flags,2) //Umschalttaste gedrückt
   CLASS::OpenPopUp(flags,this.left+col,this.top+row)
ELSE
   CLASS::Finden("")
ENDIF
RETURN

FUNCTION Finden(cVorgabe)
//Diese Funktion wird aus dem Formular von GRID1_onRightMouseDown aufgerufen
//Außerdem von den onClick-Event-Handlern der Popup wobei hier die Vorgabe gesetzt ist
IF .NOT. EMPTY(cVorgabe)
   DO CASE
      CASE cVorgabe="N"
         form.rowset.indexname="NAME"
      .....
   ENDCASE
   CLASS::SetColors()
ENDIF
DO CASE
   CASE form.rowset.indexname=="NAME"
     form.Suchen.text="Suchen nach Name"
     ....
ENDCASE
form.Suchen.readModal()
RETURN

FUNCTION OpenPopup(flags,col,row)
form.LLDesign=BITSET(flags,3) //STRG-gedrückt, ermöglicht das Bearbeiten von Drucklisten
form.Popup1.top=col+10 //Positioniert das Popup rechts unterhalb des Mauszeigers
form.Popup1.left=row+1
form.Popup1.Speichern=form.rowset.modified
form.Popup1.Verwerfen=form.rowset
form.Popup1.open()
RETURN
```

Das Umschalten auf die 2. Formularseite erfolgt über einen Grid1_onLeftDblClick-Handler, der mit dem canSelChange des Notebooks synchronisiert wird:

```
FUNCTION GRID1_onLeftDblClick(flags,col,row)
IF CLASS::SelectPage2()
   form.Notebook1.cursel=2
ENDIF
RETURN

FUNCTION NOTEBOOK1_canSelChange(nNewSel)
LOCAL lChange
DO CASE
   CASE nNewSel=2
      lChange=CLASS::SelectPage2()
   ...
ENDCASE
RETURN lChange

FUNCTION SelectPage2
LOCAL lOK
lOK=true //Bedingung zum Wechsel der Formularseite
RETURN lOK
```

Listenerstellung

Nach Datenerfassung und Bearbeitung im PC möchte man die Daten zu Papier bringen und per Fax oder Email versenden. Hierzu stellt dBASE eine eigene Reportengine als Two-Way-Tool zur Verfügung. Abgesehen davon, dass man tatsächlich fast jeden Report damit programmieren kann, stehen Aufwand und Ergebnis in keinerlei Verhältnis. Der Designer ist schlicht nur dazu zu verwenden, einen groben Entwurf des Reports herzustellen. Der Versuch eines exakten Positionieren von Controls ist zum Scheitern verurteilt.
Als Ausgabeformat stehen lediglich Drucker, ein Vorschaufenster oder HTML-Format zur Verfügung, letzteres erzeugt jedoch Dateien, deren Druckansicht mit dem eigentlichen Report nichts mehr zu tun hat. Meine alten Reports habe ich grundsätzlich (nach vorherigem Entwurf mit dem Designer) im Quellcodeeditor erstellt – der zeitliche Aufwand war entsprechend.
Auf der dBKON 2002 wurde durch Hans-Peter Neuwirth Combit List&Label 8 vorgestellt. Bei aller Skepsis habe ich einige interessante Alternativen zum dBASE-Reportgenerator gesehen. Einige Zeit später konnte ich an einer Roadshow für List&Label 9 teilnehmen, auf der ich mich entschlossen habe, unverzüglich umzusteigen.
Entscheidend für meinen Wechsel waren folgende Aspekte:

- die erzeugten Reports werden exakt so gedruckt, wie sie angezeigt werden
- bei der Vorschau kann schnell zwischen einzelnen Seiten umgeschaltet werden, es ist ohne Verzögerung möglich von der 1 auf die letzte Seite und zurück zu wechseln, besonders bei umfangreichen Listen ist dies von großem Vorteil
- listen-, benutzer- und computerabhängige Speicherung der Druckereinstellungen sind möglich
- formatierte Textobjekte (wie z.B. in Word) werden direkt unterstützt
- vielfältige Exportmöglichkeiten ohne zusätzliche Software und Lizenzgebühren (*.PDF, *.RTF, *.XLS)
- direkter Versand der erstellten Listen per Mail/Fax ist möglich
- Listen können durch den Kunden individuell angepasst werden (Professional Version)

Aber ganz ohne Programmieraufwand geht es nicht! Es muß eine Schnittstelle zwischen dBASE und List&Label definiert werden, über die folgende Vermittlung realisiert wird:

- Anmeldung der Datensatzobjekte bei List&Label
- Namentliche Definition der Felder, Variablen und Parameter
- Regelung der Datenversorgung der List&Label Engine
- Konvertierung der HTML-Tags von Memo-Feldern

Hierzu wurde die benutzerdefinierte Klasse LLReport.CRP angelegt. Die Dateierweiterung ist hier nur aus Kompatibilitätsgründen gewählt, die Klasse selbst ist nicht von `Report` sondern von `Object` abgeleitet. Im Objektinspektor wird unter `baseClassName` der Eintrag `LLReport` angezeigt.

Die Klasse LLReport verfügt über folgende Eigenschaften:

Eigenschaft	**Standard**	**Verwendung/Hinweise**
baseclassname	LLReport	
directory	SET("DIRECTORY")	Verzeichnis, das beim Aufruf des Reports aktiv ist
lastWindow	0	Fensterhandle des aufrufenden Fensters (Controls)
LL	Objekt	List&Label Objekt
LL.Application	Name der EXE	Ist eine Variable _app.oInfo.ListAndLabel vom Typ "C" definiert, wird diese als Standard verwendet
LL.AutoFaxVar	""	Name des Feldes bzw. der Variablen mit Faxnummer des Empfängers
LL.AutoMailVar	""	Name des Feldes bzw. der Variablen mit Email-Adresse des Empfängers
LL.ChooseDest	true	Flag, ob Zielauswahl möglich
LL.Copies	0	Anzahl der Kopien, 0 mit Auswahl
LL.CopyCountField	""	Feld, mit dem die Anzahl der zu druckenden Etiketten festgelegt wird
LL.CopyCountValue	1	Standardvorgabe, falls CopyCountField leer oder nicht vorhanden
LL.DataObjects	NULL	Objectarray mit weiteren Datenzugriffsobjekten (rowsets)
LL.Debug	false	Debugmodus
LL.DefDir	""	Verzeichnis mit List&Label Definitionen (Vorlagen) [1)]
LL.DeletePreview	true	Temporäre Vorschaudateien nach Ausgabe löschen
LL.Design	false	Flag für Designermodus

LL.Destinations	""	Mögliche Ausgabeziele, Leerstring für alle
LL.Destination	3	Ausgabeziel, dBASE-Äquivalent (Vorschau)
LL.EMail	""	EMail-Empfänger
LL.EMailBCC	""	EMail-ZusatzEmpfänger
LL.EMailBody	"Dies ist der gewünschte Report"	EMail-Text
LL.EMailCC	false	EMail-Zusatzempfänger
LL.EmailSubjectl	0	EMail-Betreff
LL.Error	0	Fehlernummer
LL.Extension	".LST"	Unterstützt werden: Listenprojekte: ".LST"," Karteikartenprojekte ".CRD",".BRF","PLN" Etikettenprojekte ".LBL",".ETI",".BAR"
LL.ExportDir	""	Hauptverzeichnis für Exportdateien [2)]
LL.ExportFilename	"Export"	Vorgabename für Exportdatei
LL.Fax	""	Faxnummer des Empfängers
LL.FileCanAdd	false	Flag zum Hinzufügen von Dateien
LL.FileCanSelect	false	Flag zum Auswählen von Dateien
LL.Filemask	""	Maske für Dateinamen bei Auswahlmöglichkeit
LL.Filename	""	Name der List&Label Vorlage
LL.Firstpage	0	Startseite, 0 mit Auswahl
LL.IntCopies	1	Anzahl der Etiketten mit gleichen Daten (nur für Etikettenprojekte)
LL.LizenzInfo	""	Lizenzschlüssel
LL.NoDialogs	false	Keine Benutzeraktion, nur Standards oder Vorgaben verwenden
LL.Params	ARRAY	Array mit benutzerdefinierten Konstanten (L&L Variablen)
LL.Params[1]	ARRAY	{"Lizenznummer",<Lizenznummer>} Ist die Variable _app.oLizenz.Lizenz vom Typ "C", wird diese für <Lizenznummer> verwendet, andernfalls der Text "**** Diese Software ist nicht lizenziert ****"
LL.Params[2]	ARRAY	{"Lizenznehmer",<Lizenznehmer>} Ist die Variable _app.oLizenz.Lizenznehmer vom Typ "C", wird diese für <Lizenznehmer> verwendet, andernfalls der Text "Testversion "
LL.Params[3]	ARRAY	{"Zielfaxnummer",""}
LL.PreviewDir	""	Verzeichnis für Vorschaudaten [2)]
LL.PrintersDir	""	Verzeichnis mit Druckerdefinitionen [2)]
LL.Projektart	LL_PROJEKT_LIST	wird automatisch in Abhängigkeit vom Projekt gesetzt
LL.ResetPage	true	Seitenzählung zurücksetzen (nur Karteikartenprojekte)
LL.ResetList	false	Seitenzählung zurücksetzen (nur Listenprojekte)
LL.ResetUser	false	Variable (Konstanten) nach jedem Datensatz neu lesen
LL.SetOffset	false	Offset-Dialog anzeigen
LL.ShowEmailDialog	true	Dialog zum EMail-Versand anzeigen
LL.Special	""	Zusatzverzeichnis mit benutzerdefinierten Anpassungen einzelner Reports
LL.Zoomfaktor	100	Zoomfaktor beim Öffnen der Vorschau
output	3	Ausgabeziel
scaleFontName	"Arial"	Standardfont für RTF-Text
scaleFontSize	10	Standardschriftgrad für RTF-Text
scaleFontBold	false	Standardauszeichnung für RTF-Text
streamsource1	OBJECT	Hauptdatensatzbereich (Rowset)

Kalkulationsfelder können auch in einem zusätzlichen Array direkt an den Hauptdatensatzbereich angehängt werden.
this.streamsource1.rowset.Calcfields=new ARRAY()
this.streamsource1.rowset.CalcFields.add({<Inhalt>,<Name>,<Typ>,<Länge>,<Dezimalstellen>})
Das LL.Params-Array enthält bereits 3 Elemente:
this.LL.Params[1]={"Lizenznummer",<Lizenznummer>}
this.LL.Params[2]={"Lizenznehmer",<Lizenznehmer>}
this.LL.Params[3]={"ZielFaxnummer",""}

dBASE-Feldtypen werden wie folgt an List&Label vermittelt:

dBASE	**List&Label**
CHARACTER	Character
LOGICAL	Logical
MEMO	RTF[|<nSize>][|cFontname]
NUMERIC,AUTOINC,FLOAT, LONG, DOUBLE	Numeric
DATE	DATE
DATETIME	Character
OLE, BINARY	keine Vermittlung

Sie können diese Basisklasse problemlos in eigene Projekte einbinden. Hierzu ist die Installation der List&Label Software Version 10 erforderlich. Diese befindet sich als Trial-Version auf der Konferenz CD und kann außerdem kostenfrei von www.combit.de heruntergeladen werden. Nach Installation des Vollprodukts müssen keine Anpassungen vorgenommen werden.

Obwohl es nicht erforderlich ist, für die List&Label Reportaufrufe die Dateierweiterung `REP` zu verwenden, wird dies dennoch empfohlen, um im Projektexplorer die Zuordnung der Quelltexte zu den entsprechenden Programmkomponenten zu gewährleisten.

Das folgende Beispiel demonstriert die Verwendung der Klasse

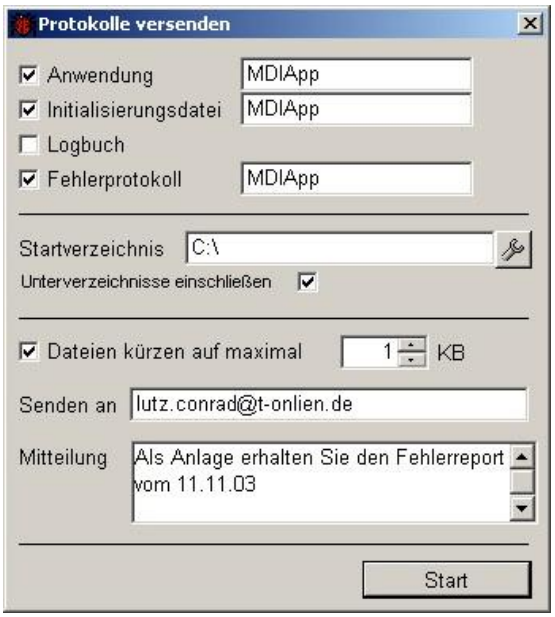
```
class FISHReport of LLReport FROM LLReport.CRO
   this.FISH = new QUERY()
   this.FISH.parent = this
   with (this.FISH)
      sql = "SELECT * FROM FISH"
      requestLive = false
      active = true
   endwith

   this.STREAMSOURCE1.rowset = this.fish.rowset

   FUNCTION Init(lDirekt,lDesign)
   SUPER::Init()
   IF .NOT. EMPTY(lDirekt)
      this.output=1
   ENDIF
   IF .NOT. EMPTY(lDesign)
      this.LL.Design=true
   ENDIF
   this.LL.Filename="Fish"
   RETURN
endclass
```

Das war's.
Mit der dBASE-internen Klasse Report hat dieser Quelltext nur dem Namen nach zu tun. Es handelt sich de facto um ein Datenmodul, welches stets über eine Methode Init() verfügt, in der die Parameter des Moduls gesetzt werden. Bei Erfordernis können weitere Methoden zur Datenauswertung hinzugefügt werden.
Die Methode render() befindet sich in der Basisklasse LLReport.

ErrorScan

Dieses Modul dient zur problemlosen Übermittlung des Fehlerprotokolls einer Anwendung an die Hotline. Oft ist der Anwender allein damit überfordert, eine angezeigte Fehlermeldung richtig und vollständig wiederzugeben. Seit dB2K 0.4 kann eine automatische Protokollierung von Programmfehlern erfolgen. Damit ist dann zumindest nachvollziehbar, wann und wo welcher Fehler aufgetreten ist.
Der Name der Protokolldatei und der Speicherort können mittels `_app.ErrorLogFile` festgelegt werden. Als Dateinamen ist hier der Name der EXE und als Erweiterung `LOG` zu empfehlen. Bei der Festlegung des Pfades ist bei einer Netzwerkinstallation zu berücksichtigen, ob alle Fehler in eine gemeinsame Datei oder arbeitsplatzabhängig gespeichert werden sollen. Zur exakteren Fehleranalyse empfiehlt sich letztere Vorgehensweise.
Jetzt müsste der Benutzer nur noch eine Email an die Hotline schreiben, diese Protokolldatei finden und als Anhang beifügen und dann versenden. Dieses Problem kann sich zur Geduldsprobe von Anwender und Hotline auswachsen.
Aus diesem Grund wurde unter Verwendung von List&Label eine Möglichkeit zum automatischen Versand des Fehlerprotokolls und weiterer Informationen geschaffen.

Das Feld `Anwendung` enthält den Namen der zu suchenden Anwendung. Fehlt die Erweiterung, wird automatisch mit EXE ergänzt.
Im Feld `Initialisierungsdatei` wird entsprechende Name angegeben. Fehlt hier die Erweiterung, wird automatisch mit INI ergänzt.
Das Feld `Logbuch` wird nur dann angezeigt, wenn die Option explizit gewählt wird. Ist dies der Falls, so wird die Datei, in welche die Benutzerzugriffe protokolliert werden ebenfalls gesucht.
Unter `Fehlerprotokoll` ist die o.a. Datei einzutragen. Fehlt die Erweiterung, wird automatisch mit LOG ergänzt.
Das Feld Dateien kürzen auf bezieht sich selbstverständlich nur auf die Fehlerprotokolldatei. Zeitlich zurückliegende Einträge werden nach dem Versand soweit entfernt, dass nur noch die angegebene Dateigröße verbleibt.
Unter Senden an wird die Email-Adresse angegeben, an die das Protokoll bei installiertem MAPI-Client ohne weitere Rückfrage versandt wird, unter Mitteilung kann noch eine ergänzende Information für die Hotline angegeben werden.



Um die letzten Unwägbarkeiten zu beseitigen kann die EXE auch aus einer Batch-Datei mit Parametern aufgerufen werden, der Benutzer muß dann wirklich nur noch das Icon doppelt anklicken und alles läuft von selbst.
Als Parameter stehen zur Verfügung:

/f:<Dateiname der Protokolldatei>	
/m:<EmailAdresse des Empfängers>	
/s:<Größe der Rumpfdatei in KB>	
/v:<Startverzeichnis>	
/u:	Wenn angegeben, werden keine Unterverzeichnisse ausgewählt (außer bei Arbeitsplatz)
/d:<Laufwerksbeschränkungen>	wenn nicht angegeben, werden alle Laufwerke gescannt, sofern nicht ein expliziter Pfad angegeben wird (Diese Option ist nur wirksam wenn 'Arbeitsplatz' gescannt wird)
/i:<Infotext für Empfänger>	
/a:<Name der Anwendung>	
/n:<Name der Initialisierungsdatei>	
/b:<Name des Logbuchs>	Wenn Name des Logbuchs nicht angegeben, wird Nachweis.DBF verwendet

Unabhängig von den in den Dateien enthaltenen Informationen werden folgende weitere Angaben bezüglich der aktuellen Arbeitsstation übermittelt:
Betriebssystem mit Versionsnummer
dBASE-Version
BDE-Version
Globaler Sprachtreiber und Zeichensatz
Computername
Laufwerke und deren freie Kapazitäten
Temporäre Verzeichnisse
Windows-Verzeichnis
Prozessor
Pfad
BDE-Einträge (Datenbanken und deren Pfade)

DBSQL

Dieses Modul kann zur Erstellung von List&Label Reports aus beliebigen Tabellen (nicht nur dBASE) unter Vermittlung der BDE verwendet werden. Es ist quasi die eierlegende Wollmilchsau.

Die EXE erwartet als ersten Aufrufparameter einen Dateinamen, der eine gültige SQL-Anweisung enthält. Alle weiteren Parameter sind optional und können in beliebiger Reihenfolge angeordnet werden.

/f:<Dateiname der Zieldatei>	Wenn nicht angegeben, wird als Dateiname "Export" verwendet
/d	im Designmodus öffnen (Layout des Reports bearbeiten), wenn nicht angegeben, wird versucht, eine vorhandene Liste zu öffnen
/p?	Projektart, (/pc Karteikartenprojekt, /pe Etikettenprojekt, keine Angabe Listenprojekt (Standard))
/m:<EMailadresse des Empfängers>	
/x:<Faxnummer des Empfängers>	
/s	Auswahl einer Zieldatei ermöglichen

Nachteilig bei diesem Verfahren ist, dass lediglich Felder, aber keine Konstanten an List&Label übergeben werden können. Die gesamte Datenauswahl muß hier mittels einer einzigen SQL-Anweisung erfolgen.