Distribution und Aktualisierung von Anwendungen

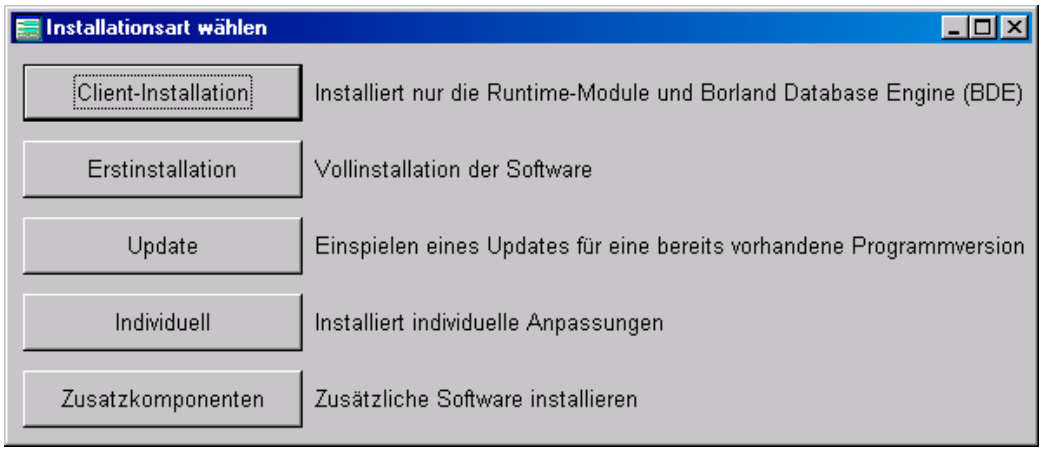
Dieser Beitrag wendet sich vor allem an Entwickler, die eine gleiche Anwendung bei verschiedenen Kunden installieren und warten. Aber auch für Einzellösungen werden Möglichkeiten zur Erleichterung bei der Installation, Pflege und Unterhaltung von Applikationen vorgestellt.
Grundsatz sollte sein, eine hohe Zuverlässigkeit bei der Installation und Aktualisierung der Applikation auch bei minimalen Computerkenntnissen des Anwenders sicher zu gewährleisten.

Im ersten Teil des Vortrags werden einige Möglichkeiten zur Gestaltung einer Installations-CD vorgestellt. Anschließend werde ich auf Methoden zur Aktualisierung vorhandener Datenbestände im Zuge eines Updates eingehen. Es folgen einige Tips zur Konfiguration von Anwendungen und zur Gewährleistung der Datensicherheit. Außerdem möchte ich noch einige selbstentwickelte Tools und Klassen vorstellen, die problemlos in eigene Applikationen eingebunden werden können..

Die verwendete Software steht als Quelltext in benutzerdefinierten Klassen zur Verfügung.

Installations-CD

Im Hauptverzeichnis der Installations-CD befindet `INSTALL.EXE`, von der aus die weitere Installation bzw. das Einspielen von Updates und Ergänzungen gestartet wird. Grundsätzlich sind folgende Komponenten enthalten:



- *Client-Installation*
  In diesem Verzeichnis befindet sich nur die aktuelle Runtime. Bei Ausführung werden demnach nur Runtime und BDE auf der aktuellen Arbeitsstation installiert.
- *Erstinstallation* (bisher mit InstallShield)
  Enthält eine Installationsroutine, die wie üblich mit SETUP.EXE gestartet wird. Bei Ausführung der Datei wird die Anwendung vollständig installiert. Bei größeren Programmen gibt es außerdem die Möglichkeit einer benutzerdefinierten Installation.
  Dieses Verfahren sollte vorwiegend bei Erstinstallation auf dem Server angewandt werden.
- *Update*
  Hier befinden sich die aktuellen Programmversionen als EXE.
- *Individuell*
  Wenn für die Anwendung kundenspezifische Module vorgesehen sind, werden die betreffenden Objektdateien in das Verzeichnis der Anwendungs-EXE kopiert. Mittels DEO (Dynamic External Objects) stehen die erforderlichen Anpassungen zur Verfügung.
- *Zusatzmodule*
  Dies sind z. Zt. folgende kleine ausführbare Dateien, die ins Windows-Systemverzeichnis kopiert werden und somit unmittelbar zur Verfügung stehen.
  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
  Nach Vorliegen einer neuen Programmversion werden nur die erforderlichen Module aktualisiert und es können weitere Datenträger erstellt werden.

Die Installations-CD enthält im Hauptverzeichnis eine entpackte vollständige dB2K-Runtime sowie eine minimal konfigurierte BDE. Damit ist sichergestellt, daß auch ohne vorherige dB2K bzw. BDE-Installation die Anwendung korrekt ausgeführt wird. Eine gleiche Verfahrensweise kann auch für Demo-CDs empfohlen werden, da bei diesem Verfahren zunächst keine Einträge in die Registry erfolgen und auch keine Daten auf die Festplatte kopiert werden müssen.

Hinweis: Auch wenn durch eine dB2K-Anwendung keinerlei Datenzugriffe erfolgen, kann bisher (bis dB2K 0.4) nicht auf das Vorhandensein der BDE verzichtet werden. Beim Start einer dB2K-Anwendung wird durch die Runtime nach der BDE gesucht. Ist letztere nicht im Zugriff, erfolgt eine Fehlermeldung. Dies wäre zwar unter der genannten Voraussetzung für den weiteren Programmablauf unerheblich, wirkt aber auf den potentiellen Kunden unprofessionell.

Beim Aufruf der `INSTALL.EXE` muß der Haupt-Datenbankalias für die Anwendung übergeben werden. Dies erfolgt direkt über die `AUTORUN.INF`. Falls die Autostart Funktion deaktiviert wurde, erfolgt der Aufruf über die Stapeldatei `INSTALL.BAT`.

Client-Installation

Vielleicht werden Sie sich wundern, warum diese Installationsart an erster Stelle steht. Jede Installation auf einer Arbeitsstation ist zunächst wie eine Client-Installation zu betrachten. Beim Austausch einer Arbeitsstation oder beim Einrichten eines neuen Arbeitsplatzes sollte diese Methode gewählt werden.
Dabei ist davon auszugehen, daß die Datenverzeichnisse (auf dem Server) bereits angelegt und ggf. Daten erfaßt sind. Auch die Anwendung selbst ist bereits installiert. Folgende Tätigkeiten müssen noch erledigt werden:

- Installation der dB2K Runtime
- Installation und ggf. Konfiguration der BDE
- Erstellen und Einrichten des BDE-Pfades
- Einrichten des Links auf die Anwendung

Falls individuelle Benutzerrechte auf der Arbeitsstation festgelegt wurden, sollte die Installation von einem Benutzer den Zugriffsrechten eines Administrators vorgenommen werden.
Die Runtime und BDE-Module werden von KSoft Inc. in einer ausführbaren Datei zur Verfügung gestellt. Diese Datei darf frei kopiert und weitergegeben werden. Die Installation der Runtime erfolgt standardmäßig in

```
<Programme>\dBASE\dB2K\BIN
```

Für die BDE ist der Pfad

```
<Programme>\<Gemeinsame Dateien>\Borland\BDE
```

vorgesehen.

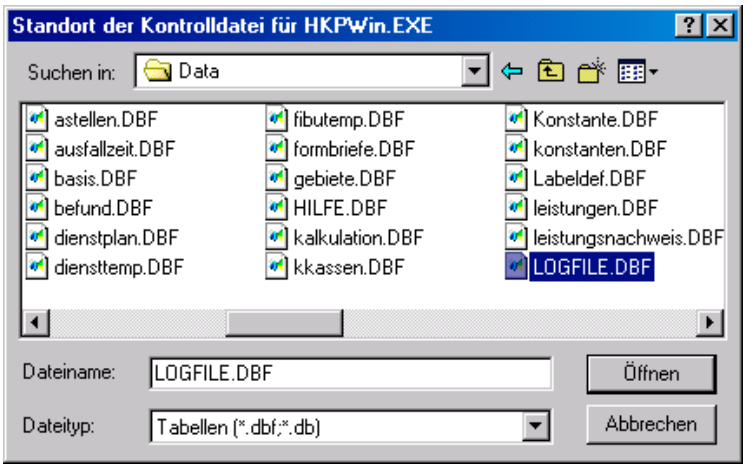
Die Client Installation wird im onClick-Event-Handler der entsprechenden Schaltfläche ausgeführt. Dabei gilt es zu beachten, daß die Ausführung des externen Installations-Moduls beendet sein muß, bevor mit der Einrichtung des BDE-Pfades fortgesetzt werden kann. Dieses Problem ließe sich mit einer MSGBOX nicht lösen.

```
....
RUN(true,(<Runtime-Installation.EXE>))
MSGBOX(Richten Sie nun den BDE-Hauptpfad ein...)
....
```

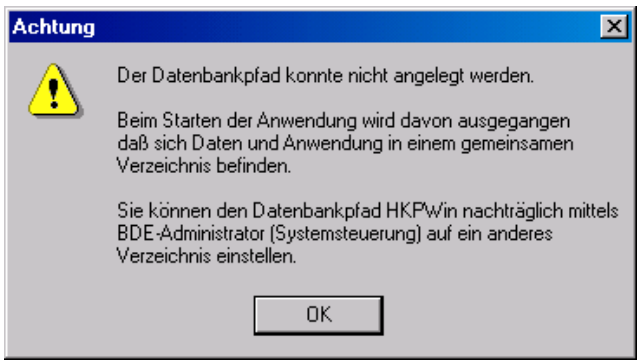
führt nicht zum Erfolg. Bereits während des Ladens der Runtime-Installation würde so die MSGBOX geöffnet und der Anwender klickt natürlich prompt auf OK. Falls die BDE noch nicht installiert ist, führt dieses Verfahren unweigerlich zu einem Fehler.
Es muß daher ein Weg gefunden werden, das Programm so lange einzufrieren, bis die BDE-Installation abgeschlossen ist. Hierfür verwende ich ein *Timer*-Objekt, welches unmittelbar vor dem Start der BDE-Installation aktiviert wird. Vor dem Starten des Timers wird die benutzerdefinierte Eigenschaft *RuntimeIsOpen* des Formulars mit *false* initialisiert.
Im *onTimer*-Event wird mittels der Windows-API-Funktion *FindWindow()* permanent nach einem geöffneten Fenster mit dem Titel der Runtime-Installationsroutine (hier "dB2K 0.4 Setup") gesucht. Wird dies gefunden, so liefert *FindWindow()* das dazugehörige Fensterhandle, andernfalls 0.
Solange das Runtime-Installations-Modul nicht aktiv ist, bleibt das Flag *RuntimeIsOpen* auf dem Wert *false*. Erst wenn der Ladevorgang abgeschlossen ist, wird das aktuelle Formular verborgen und das Flag auf *true* gesetzt. Nun müssen wir nur noch warten, bis das Runtime-Installations-Modul wieder geschlossen wird und *FindWindow()* den Wert 0 liefert. Dann kann es weitergehen mit der Einrichtung des BDE-Hauptalias.

```
FUNCTION TIMER1_onTimer
LOCAL nHandle
LOCAL cFilename
LOCAL cPath
nHandle=FindWindow(0,RUNTIMENAME)
IF nHandle>0
   //Runtime Installation läuft
   this.parent.visible=false
   this.parent.RuntimeIsOpen=true
ELSE
   //Runtime-Installation läuft nicht
   IF this.parent.RuntimeIsOpen
      //ist aber bereits ausgeführt
      this.enabled=false
      ..............//Jetzt noch die BDE-Einstellung vornehmen
      this.parent.close()
   ENDIF
ENDIF
RETURN
```

Obwohl dB2K Anwendungen auch ohne Einrichtung eines Datenbankalias laufen, bringt die Verwendung dieses Features wesentliche Vorteile. Für die meisten Anwendungen reicht es aber aus, einen (einzigen) Datenbankalias zu definieren. Hierfür wird gewöhnlich das Modul BDE-Verwaltung der Systemsteuerung verwendet.
Weil bei der Client-Installation die Datenstrukturen aber bereits angelegt sind, kann der Hauptalias auch ohne Aufruf der BDE-Verwaltung erstellt werden.



Jede meiner Anwendungen enthält im Hauptdatenverzeichnis eine Kontrolldatei `LOGFILE.DBF`. In dieser Datei sind u.a. die Dateinamen aller Tabellen der Anwendung und deren Datenbankpfade gespeichert. Nach Abschluß der Runtime-Installation muß also nur der Pfad zu `LOGFILE.DBF` ermittelt und in die BDE-Verwaltung übernommen werden. Hierzu werden einige BDE-API-Funktionen verwendet.
Falls der BDE-Pfad bei der Client-Installation nicht angegeben wird, erhält der Anwender folgende Information:



Zum Abschluß der Client Installation ist nun noch die Verknüpfung zur Anwendungs-EXE zu erstellen. Dies dürfte in den meisten Fällen über ein Icon auf dem Desktop geschehen.

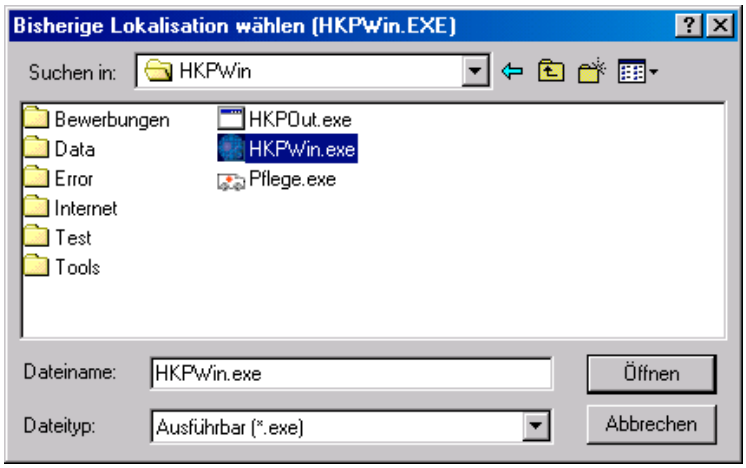
Erstinstallation

Diese Installationsart startet einen Script der Software InstallShield. Nach dem Start des externen Moduls wird das Installationsformular geschlossen und die weitere Steuerung an InstallShield übergeben.
Bei der Vorbereitung der Setup-Routine können bereits einige Einstellungen der BDE und erforderliche Einträge für die Registry vorgegeben werden. Je nach Umfang der Software werden bei der Erstinstallation folgende Komponenten eingerichtet:

Runtime	<Programme>dBASE\DB2K\BIN
BDE	<Programme>\<Gemeinsame Dateien>\Borland\BDE
Ausführbare Datei der Anwendung (EXE)	<Installationsverzeichnis>
Initialisierungsdatei der Anwendung (INI)	<Installationsverzeichnis>\DATA
Tabellen mit Standardwerten	<Installationsverzeichnis>\DATA
Screenshots für Demo und Hilfe	<Installationsverzeichnis>\DEMO
Zusätzliche Multimediaressourcen	<Installationsverzeichnis>\MEDIA

Update

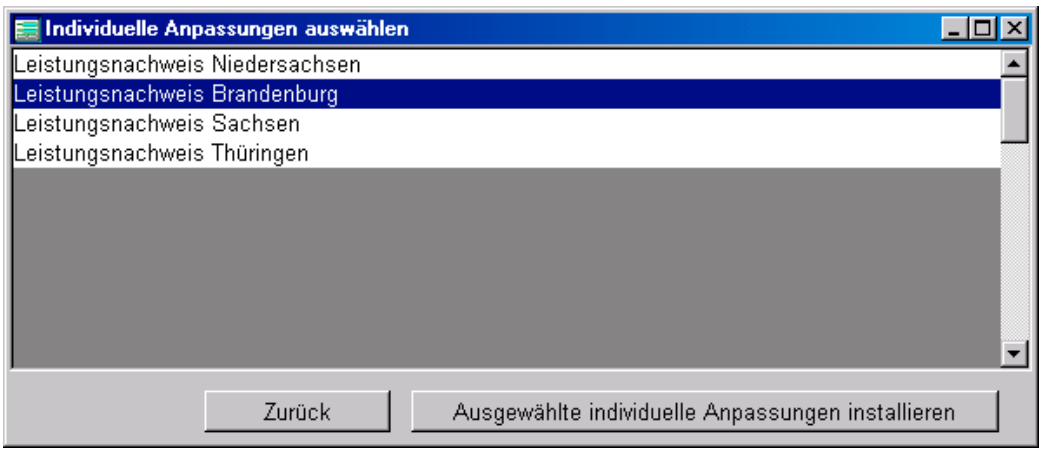
Mit dieser Option werden Programmaktualisierungen vorgenommen. Es ist davon auszugehen, daß sowohl Runtime als auch BDE bereits installiert sind. Außerdem sollten auch die benötigten Datenbankpfade (BDE-Aliase) eingerichtet sein.
Nach Angabe des Pfades zur bereits installierten Anwendung werden alle Dateien des Verzeichnisses Update dorthin kopiert. Weitere Änderungen sind nicht erforderlich, nach dem Einspielen des Updates steht auf allen Arbeitsplätzen sofort die neue Programmversion zur Verfügung.



Alle eventuell erforderlichen Anpassungen der Tabellenstruktur, von Anwendungsparametern u.s.w. erfolgen beim ersten Start der neuen Version.

Individuell

Häufig werden kundenspezifische Anpassungen einzelner Module benötigt. Die EXE enthält in diesem Fall eine für die meisten Kunden zutreffende Basisversion. Für spezielle Wünsche werden einzelne Module ‚maßgerecht' angefertigt und als Objektdateien bereitgestellt. Im Laufe der Zeit können so Bibliotheken von Objekten entstehen, die zwar einen gemeinsamen Verwendungszweck haben, jedoch sowohl im optischen Erscheinungsbild als auch in der Funktionalität völlig anders aufgebaut sind.Klassen- und Dateinamen müssen den in der ausführbaren Datei (EXE) enthaltenen Namen entsprechen. Die Installation dieser Komponenten sollte in das Verzeichnis erfolgen, in welchem sich auch die EXE befindet. Nach Angabe des Pfades zur bereits installierten Anwendung werden alle ausgewählten Dateien dorthin kopiert. Eine Mehrfachauswahl ist möglich.



Eine Inhaltsangabe über die vorhandenen Dateien befindet sich im CD-Verzeichnis INDIV. In der Datei `DEO.TXT` sind die Verweise auf die Module zeilenweise gespeichert. Für jede Spezifikation (jeden Kunden) gibt es ein Unterverzeichnis mit den individuellen Anwendungsobjekten.

Konfiguration

Wie auch viele andere Windows-Anwendungen legt auch dB2K automatisch Initialisierungsdateien (*.INI) an. Diese Dateien befinden sich im gleichen Verzeichnis wie die ausführbare Datei selbst. Wenn also diese Datei ohnehin immer vorhanden ist, dann kann man sie doch auch zur Speicherung eigener Konfigurationsparameter verwenden?
Obwohl dies prinzipiell möglich ist, möchte ich hier ein etwas anderes Verfahren vorstellen.
Grundsätzlich sollte jede Anwendung so aufgebaut sein, daß Daten und ausführbare Datei (sowie alle dazugehörigen Komponenten) in unterschiedlichen Verzeichnissen liegen. Weiterhin sollte auf die Daten stets ein BDE-Pfad verweisen.
Unabhängig von der durch Windows angelegten INI befindet sich im Datenverzeichnis eine weitere Initialisierungsdatei – und nur diese wird tatsächlich von der Anwendung ausgewertet!
Damit sind folgende Vorteile verbunden:

- Es ist möglich, mit einer EXE verschiedene Datenbestände zu verwalten. Hierzu muß nur der BDE-Pfad entsprechend gesetzt werden. Damit läßt sich eine Mandantenverwaltung mit völlig unabhängigen Datenbeständen problemlos aufbauen.
- Es werden keine zusätzlichen Informationen eingetragen, die vorhandene Einstellungen überschreiben können.

Für die Bearbeitung der Initalisierungsdateien steht die Klasse LC_INI (LC_INI.CC) zur Verfügung. Jede INI ist in Sections gegliedert. Beim Lesen der INI wird für jede Section ein Objekt generiert und an das Anwendungsobjekt _app angehängt.
Aus `[Lizenz]` entsteht so z.B. das Objekt `_app.oLizenz`.
Die einzelnen Einträge in der Initialisierungsdatei werden zu Eigenschaften des übergeordneten Objektes und deren Werten umgeformt.

```
//Ausschnitt aus Initialisierungsdatei (Beamer: HKPWIN.INI)
[Lizenz]
Lizenz="Testlizenz0815"     wird verarbeitet zu  _app.oLizenz.Lizenz="Testlizenz0815"
Lizenznummer=9999999999     wird verarbeitet zu  _app.oLizenz.Lizenznummer=9999999999
...
```

Zum Anlegen und ggf. Ergänzen der Initalisierungsdatei wird ein für jede Anwendung angepaßtes Modul MAKEINI.PRG eingesetzt, welches Standardwerte für alle erforderlichen Einträge vorgibt.

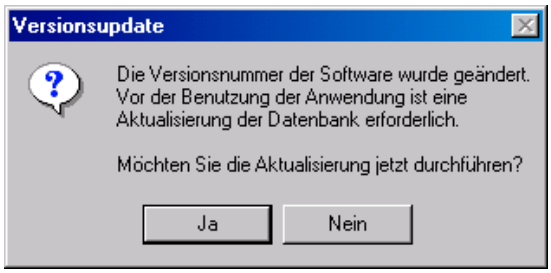
```
//Ausschnitt aus MakeIni.PRG
FPUTS(nHandle,'[Lizenz]')
FPUTS(nHandle,'Lizenz="'+_app.oLizenz.Lizenz+'"')
FPUTS(nHandle,'Firma="'+IIF(TYPE('_app.oLizenz.Firma')=='C',_app.oLizenz.Firma,'')+'"')
FPUTS(nHandle,'Zusatz="'+IIF(TYPE('_app.oLizenz.Zusatz')=='C',_app.oLizenz.Zusatz,'')+'"')
....
```

Alternativen zur Initialisierungsdatei

Die Bereitstellung der Konfigurationsparameter in einer ASCII-Datei bietet gegenüber der Verwendung einer Tabelle den Vorteil, daß die ASCII-Datei problemlos mit jedem Texteditor bearbeitet werden kann, während zur Bearbeitung der Tabelle entsprechende Datenbanktreiber vorhanden sein müssen. Für den Anwender ist es oft einfacher mit einem einfachen Texteditor (z.B. Notepad) Änderungen vorzunehmen, als mit einem Datenbanktool. Daher verwende ich vorwiegend diese Variante zur Einstellung der Ausgangsparameter. Nachteilig beim Laden der Variablen aus der INI ist jedoch, daß unterschiedliche Konfigurationen, z.B. in Abhängigkeit vom aktuellen Benutzer nur schwer zu realisieren sind. In diesem Fall ist der Einsatz einer Konfigurationstabelle oder sogar eine Kombination beider Varianten günstiger.

Tabellenaktualisierung

Bei Anwendungen, die über einen längeren Zeitraum gepflegt und weiterentwickelt werden, steht man eines Tages vor dem Problem, daß die Tabellenstruktur geändert oder ergänzt werden muß. Die Aktualisierung bei jedem Kunden wäre mit einem sehr hohen zeitlichen Aufwand verbunden, wenn es nicht Möglichkeiten gäbe, dies im Zuge des Updates automatisch zu vollziehen.
In meinen Anwendungen hat sich folgende Lösung bewährt. Zu jeder Anwendung gibt es eine Datei VERSION.INF, die im Datenverzeichnis der Anwendung liegt. Diese Datei enthält lediglich eine Textzeile - die Versionsnummer des aktuellen Datenbestandes. In der EXE selbst ist ebenfalls eine Versionsnummer festgeschrieben.
Die Versionsnummer ist in drei Gruppen gegliedert. Die erste Stelle verweist auf die eingesetzte dBASE Version, die zweite auf die aktuelle Version der Datenstruktur und der Initialisierungsdatei, die dritte enthält die Nummer der EXE.
Wird beim Programmstart festgestellt, daß sich die erste oder zweite Gruppe der Versionsnummer unterscheiden, erfolgt eine automatische Tabellenaktualisierung und die Aktualisierung der Initialisierungsdatei.



Hierzu ist es selbstverständlich erforderlich, alle Informationen zur Tabellenstruktur in den Quelltext der Anwendung aufzunehmen. Dies sieht zunächst nach viel Schreibarbeit aus, ist aber letztlich weniger aufwendig als die Definition der Tabellenstruktur mit dem Tabellendesigner.
Alle Struktur- und Indexdefinitionen befinden sich in einer Klasse, die in einer Programmdatei (PRG) der Anwendung enthalten ist. Hier ist für jede Tabelle der Anwendung eine Funktion (Methode), in der die Struktur und der Indexaufbau in Form von Arrays beschrieben werden.
Der Aufruf der einzelnen Funktionen wird über die Kontrolldatei `LOGFILE.DBF` vermittelt. In der Datei sind die Namen aller Tabellen der Anwendung einschließlich ihrer Datenbankaliase gespeichert. Die Tabelle wird bei einem Update sequentiell durchlaufen und alle erforderlichen Anpassungen und Überprüfungen ausgeführt.
Nun zu den Funktionen:
Im Strukturarray wird mit jedem Element ein Tabellenfeld beschrieben. Jedes Element enthält bis zu drei Subarrays.
Das *erste Subarray* beinhaltet die klassische Feldbeschreibung in der Reihenfolge Feldname, Feldtyp, Feldlänge und Dezimalstellen. (Eine Indexdefinition ist hier nicht vorgesehen.)
Die Angabe des Feldnamens ist dabei obligatorisch, alle anderen Angaben sind optional. Sie werden ggf. durch Standardvorgaben ersetzt.

```
{„Name“,“C“, 25, 0}
```

Für die Definition der Feldtypen werden folgende Abkürzungen verwendet:

C Zeichen	N Numerisch	L Logisch	M Memo
D Datum	F Gleitkomma	G OLE	O Double
+ AutoInc,	@ Zeitstempel	B Binär	I Long

Das optionale *zweite Subarray* enthält Definitionen zu den Standard-Feldeigenschaften in der Reihenfolge Required, Minimum, Maximum und Default. Alle Angaben sind optional, falls jedoch lediglich ein Wert für Default definiert werden soll, müssen auch die anderen Werte angegeben werden. Dies kann aber auch der Wert NULL sein.

```
{NULL, NULL, NULL, true} //Logisches Feld wird mit Standardwert true belegt
```

Falls keine Standard-Feldeigenschaften benötigt werden, jedoch benutzerdefinierte Feldeigenschaften vorgesehen sind, ist ein leeres Subarray anzugeben.

Im optionalen *dritten Subarray* werden schließlich die benutzerdefinierten Feldeigenschaften festgelegt. Dabei ist die Reihenfolge beliebig. Es erfolgt keine Prüfung auf Gültigkeit der Werte. Jede Eigenschaft wird mit einem String festgelegt, dabei wird das erste Gleichheitszeichen als Trennung zwischen dem Namen und dem Wert der Eigenschaft interpretiert.

```
{„colornormal=black/yellow“,“statusMessage=Familienname – Obligatorische Eingabe“}
```

Durch diese Definition wird z.B. gewährleistet, daß das angegebene Feld in den Eingabeobjekten (Eingabefelder, Tabellenspalten u.s.w.) aller Formulare der Anwendung in der angegebenen Farbe dargestellt wird. Hat das betreffende Eingabefeld den Focus, so wird in der Statuszeile die entsprechende Meldung angezeigt.

Die Indexdefinitionen für die Tabelle sind in einem weiteren Array enthalten. Mit jedem Element werden die Informationen für einen Index festgelegt. Die Elemente sind auch hier Subarrays, die aus bis zu fünf Elementen bestehen.
Das erste Element bezeichnet den Indexnamen, das zweite beschreibt den Schlüsselaufbau. Diese Elemente müssen angegeben werden, alle weiteren sind optional.
Im dritten Element wird, falls erforderlich, eine Bedingung (FOR) definiert.
Das vierte Element bestimmt die Sortierreihenfolge (DESCENDING), ist es nicht angegeben oder hat den Wert *false*, so wird ein aufsteigender Index (Standard) erzeugt, andernfalls ein absteigender.
Das fünfte Element legt schließlich fest, ob es sich um einen UNIQUE-Index handelt. Dies ist nur dann der Fall, wenn es den Wert *true* enthält.

```
{„Offen“,“DTOS(Rechnungsdatum)+Rechnung“,“Betrag-Gutschrift-Zahlung>0“,true, false})
```

Die Klasse mit den Strukturdefinitionen enthält außerdem alle Routinen zum Anpassen der Tabellenstruktur früherer Programmversionen an den aktuellen Stand.
Bei einem Update ist es daher nicht erforderlich, alle bisherigen Anpassungen nacheinander auszuführen, sondern der Anwender kann z.B. direkt von der Version 1.6.1 auf die Version 1.9.3 aktualisieren.
Weiterhin ist es nicht erforderlich, bei der Distribution einer Anwendung, alle verwendeten Dateien bereits im leeren Zustand mitzuliefern. Die Tabellen werden direkt auf dem Zielsystem in den vorgesehenen Verzeichnissen angelegt. Im Installationsscript kann daher auf eine Vielzahl von Einträgen verzichtet werden, was wiederum die Fehleranfälligkeit erheblich reduziert.
Außerdem bietet dieses Verfahren insbesondere für die unmittelbare Programmierung einen weiteren großen Vorteil. Durch die zentrale Definition aller Strukturen in einer Quelltextdatei stehen die dort enthaltenen Informationen auch beim Bearbeiten der Event-Handler jederzeit zur Verfügung. Neben dem zu bearbeitenden Quelltext wird die Definitionsdatei geöffnet und schon hat man Zugriff auf alle Feldnamen, Feldeigenschaften und Indexdefinitionen.

Zum Anlegen der Tabellenstruktur und für die eventuell erforderliche Übernahme vorhandener Datenbestände werden die Struktur- und Indexdefinitionen im Weiteren an eine andere Klasse übergeben.

LC_Table.CC

Eigentlicher Kern der Tabellenaktualisierung ist die Klasse *LC_Table*, die in der gleichnamigen CC-Datei enthalten ist. Mit Hilfe dieser Klasse ist es möglich, Level 7 Tabellen mit Low-Level-Methoden anzulegen. Der Tabellendesigner wird hierfür nicht benötigt.
Sind Memo-, Binär- oder OLE-Felder in der Tabellenstruktur vorgesehen, so wird die erforderliche *.DBT-Datei entsprechend den aktuellen Einstellungen der BDE (Memo-Blockgröße) automatisch generiert.

Nach dem Instantiieren der Klasse können deren Eigenschaften *Messages, Progress* und *AutoClose* gesetzt werden (Standard *true*).
Das Setzen von Messages auf *false* bewirkt die Unterdrückung jeglicher Meldungen während der Ausführung von Methoden der Klasse.

Wird *Progress* auf *false* gesetzt, erfolgt keine Anzeige des Bearbeitungsfortschritts bei der Datenübernahme aus einer vorhandenen Datei.
Werden mehrere Tabellen unmittelbar nacheinander aktualisiert, würde das ständige Öffnen und Schließen des Progress-Formulars zu einem unerwünschten Flackern führen. In diesem Fall wird *AutoClose* auf *false* gesetzt, Damit bleibt das Formular zur Anzeige des Bearbeitungsfortschritts solange geöffnet, bis es durch einen anderen Prozeß geschlossen wird.

```
...
//Fortschrittsfenster schließen
f=FINDINSTANCE("ProgressForm")
IF .NOT. f==NULL
   f.close()
   f.release()
   f=NULL
ENDIF
...
```

An die Methode *Create* der Klasse wird ein Objekt übergeben, welches wie folgt strukturiert ist:

Objekt/Eigenschaft	Standard	Erf.	Bedeutung
oTable			Das Tabellenobjekt selbst
oTable.Tablename	keine Vorgabe	x	Vollständiger Tabellenname mit oder ohne Erweiterung
oTable.Structure	keine Vorgabe	x	Array mit Strukturdefinitionen
oTable.Index	keine Vorgabe		Array mit Indexinformationen
oTable.Backup	true		Anlegen einer Sicherungskopie der zu erstellenden Tabelle und Zurückladen des Inhalts
oTable.Driver	LDRIVER()		Sprachtreiber
oTable.Autonull	**false**		Leere Werte als NULL belassen Standardmäßig wird wie folgt vorgegeben: Zeichen, Memo: *leerer String* Logisch: *false* Numerisch, Gleitkomma, Long, Double: *0* Datum: *CTOD(„ . . „}* Zeitstempel: *CTODT(„ . . : : „}* OLE, Binär, Zähler: *Keine Vorgabe*
oTable.Renum	false		Autoinc-Felder neu numerieren Mit der Standardvorgabe bleiben alle Werte der Zählerfelder erhalten.
oTable.HoldStructure	true		Ist dieser Wert *true*, werden auch Felder, die in der neuen Struktur nicht mehr vorhanden, übernommen und mit den bisherigen Werten gefüllt. Wird nur ausgewertet, wenn auch *Backup* den Wert *true* hat.
oTable.HoldIndexes	false		Alle bisherigen Indexdefinitionen behalten, also auch zusätzliche, die ggf. vom Benutzer angelegt wurden.

Zunächst werden, falls nicht definiert, alle Standardwerte gesetzt. Ist die zu erstellende Tabelle bereits vorhanden, wird – sofern oTable.Backup den Wert true (Standard) hat - eine Sicherungskopie erstellt.
Mittels der Funktion FCREATE wird eine Datei angelegt und der Dateikopf einer dBASE 7 Tabelle erstellt.
Anschließend wird das Feldbeschreibungs-Arrray mittels Low-Level- Zugriff angelegt.
Hierfür wird das Struktur-Array des übergebenen Objektes benötigt, wobei folgende Regeln gelten:

- Ist ein Feldtyp nicht angegeben, so wird ein Feld vom Typ Zeichen mit der Länge 10 angelegt.
- Wurde der Feldtyp festgelegt, aber nicht die Feldlänge, so wird diese entsprechend des Typs (Logisch 1, Datum, Zeitstempel, Double jeweils 8, Zeichen, Memo, Binär, OLE, Gleitkomma, Numerisch jeweils 10, Long, Zähler jeweils 4 Zeichen) vorgegeben. Fehlerhafte Werte werden anhand des Feldtyps auf die genannten Angaben korrigiert.
- Die Anzahl der Dezimalstellen wird nur bei Feldern der Typen Numerisch und Gleitkomma berücksichtigt, fehlen diese Angaben oder wurden unzulässige Werte angegeben, erfolgt eine automatische Korrektur auf 0.
- Für Memo-, Binär und OLE-Felder wird als temporäre der Feldtyp *Zeichen* mit der Länge 10 in das Feldbeschreibungs-Array aufgenommen.

Nun werden das Array mit den Standard-Feldeigenschaften und das Array mit den benutzerdefinierten Feldeigenschaften generiert. Damit ist der vollständige Dateiheader für die leere Tabelle geschrieben.

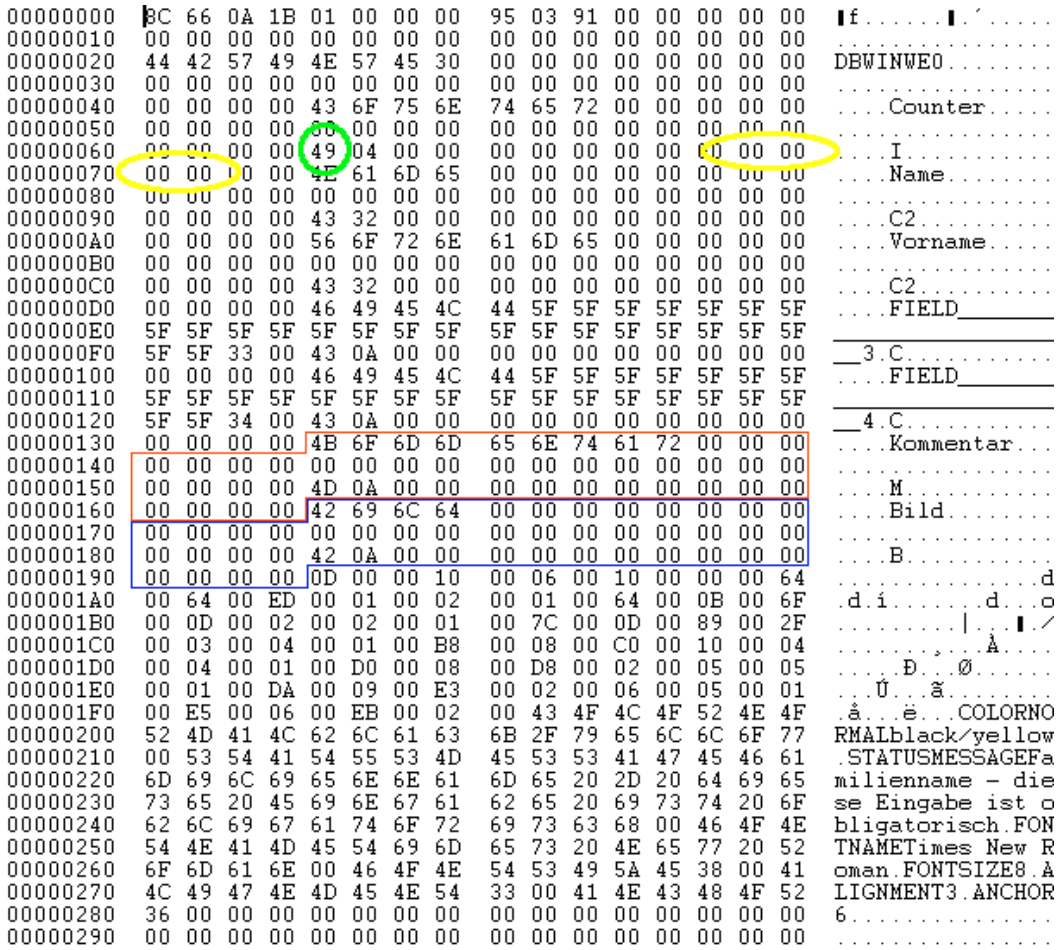
Das Anlegen der Memodatei erfolgt mit folgendem Trick. Die noch leere Tabelle wird geöffnet und mittels

```
ALTER TABLE <Tabellenname> ADD <Feldname> <Feldtyp>
```

der tatsächliche Feldname angefügt. Damit ist die *.DBT Datei – ebenfalls im leeren Zustand – erstellt bzw. modifiziert. Der Sprachtreiber entspricht dem der Haupttabelle (.DBF), die Blockgröße wird durch die BDE korrekt vorgegeben.
Anschließend werden die Angaben zum Feldnamen und Feldtyp im Datei mittels Low-Level-Zugriff getauscht und durch

```
ALTER TABLE <Tabellenname> DROP <Feldname>
```

das nun überflüssige temporäre Feld gelöscht.



Ist eine Sicherung der ursprünglichen Daten vorhanden, wird anschließend Satz für Satz in die neue Struktur importiert.

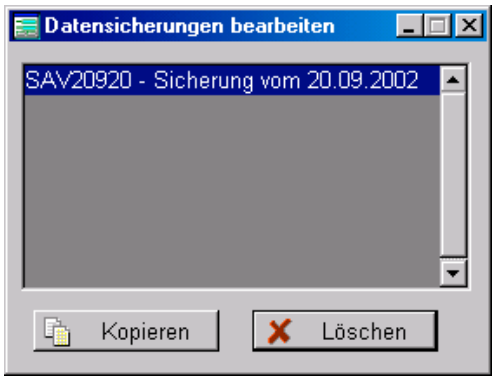
Meist ist es nicht erwünscht, vorhandene Felder vom Typ Zähler (AUTOINC) neu zu numerieren. Werden diese Felder für Schlüssel und Verknüpfungen verwendet, wäre eine Neunumerierung geradezu das Aus für die Datenbank. Beim Anlegen der Sätze würde dies aber zwangsläufig geschehen.
Aus diesem Grund wird – wenn die Neunumerierung nicht ausdrücklich gewünscht wird – der Import zunächst in ein Feld vom Typ Long erfolgen. Erst nach Abschluß der Datenübernahme wird die Angabe zum Feldtyp im Dateiheader durch den Verweis auf ein Zählerfeld (Byte 33l) ersetzt und der Header mit dem Verweis auf den nächsten Zählerwert (Bytes 43-46) ergänzt.

Nach erfolgreicher Datenübernahme werden eventuelle Sicherungen gelöscht, tritt ein Fehler auf, kann der ursprüngliche Zustand wiederhergestellt werden.

Datensicherheit

Vor jedem Versionsupdate sollten die bisherigen Daten gesichert werden. Auch hier kann man dem Anwender die Arbeit etwas erleichtertn.
Vor jeder Tabellenaktualisierung wird ein Verzeichnis `SAVjmmtt` unterhalb des Datenverzeichnisses angelegt und alle Tabellen sowie die Initialisierungsdatei dorthin kopiert. Sollten beim Update unerwartete Probleme auftreten, läßt sich der alte Zustand so unverzüglich wiederherstellen.

Damit diese Datensicherungen nicht nach und nach die Datenträger ‚zumüllen', gibt es in der Anwendung einen Dialog (LC_Sicherung.CC), mit dem vorhandene Datensicherungen kopiert oder gelöscht werden können.



Das Kopieren ist selbstverständlich nicht dafür vorgesehen, den Datenumfang nochmals zu vergrößern. Vielmehr soll damit die Möglichkeit gegeben werden, die ausgewählte Datensicherung auszulagern um sie anschließend zu löschen.

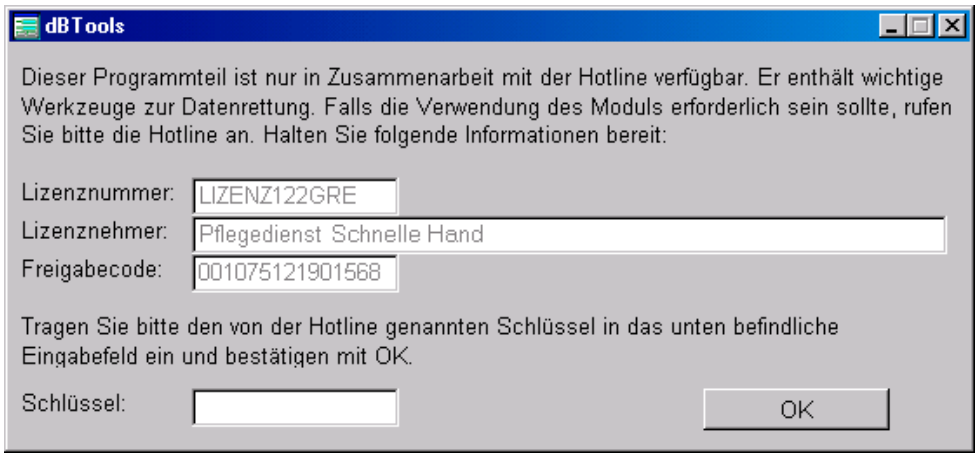
Ein weiteres Problem ist das Anlegen von *periodischen* Datensicherungen. Dies wird zwar nicht unbedingt als lästig empfunden, doch oft genug leichtsinnig vernachlässigt – weil einfach vergessen.
Ich empfehle deshalb, generell eine automatische Datensicherung vorzusehen. Der hierfür verwendete Pfad kann in den Haupteinstellungen gesetzt werden. Er sollte selbstverständlich so gewählt sein, daß alle potentiellen Nutzer über entsprechende Zugriffsrechte verfügen.
Im Datenverzeichnis befindet sich eine Datei `SAVEINF.TXT`. Diese enthält das Datum und die Uhrzeit der letzten Datensicherung. Beim Start der Anwendung wird geprüft, ob die Daten am aktuellen Tag bereits gesichert wurden. Ist das nicht der Fall, wird versucht, alle Tabellen und die Initialisierungsdatei in das angegebene Sicherungsverzeichnis zu kopieren. (Das Sicherungsverzeichnis enthält die Unterverzeichnisse Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag und Sonntag.). Ist das Kopieren erfolgreich, wird der aktuelle Zeitstempel in `SAVEINF.TXT` zurückgeschrieben.
Die Datensicherung erfolgt also nicht abends - wer will schon zum Feierabend noch auf die Datensicherung warten - sondern morgens beim Anmelden des ersten Benutzers. Außerdem gibt es so meist auch weniger Probleme beim eventuell erforderlichen exklusiven Tabellenzugriff.

Wenn es mit den Datensicherungen schon nicht klappt, dann trifft dies noch mehr auf die Reorganisation der Indexdateien zu. Ich kenne Anwender, die umfangreiche Applikationen über Monate nutzen, ohne auch nur einmal eine Reindizierung durchzuführen. Abgesehen davon, daß dies sicher ein Beispiel für die Zuverlässigkeit der Software ist, geht selbstverständlich nach und nach die Performance in den Keller. Um hier allen Problemen aus dem Weg zu gehen, ist eine periodische automatische Datenpflege zu empfehlen. Wie oft sollte im Ermessen des Anwenders liegen, aus meiner Erfahrung reicht ein Intervall von 14 Tagen aus. Ich schreibe hierzu ins Datenverzeichnis derAnwendung die Datei LReindex.TXT, in der das Datum der letzten vollständigen Reindizierung nachgewiesen wird.

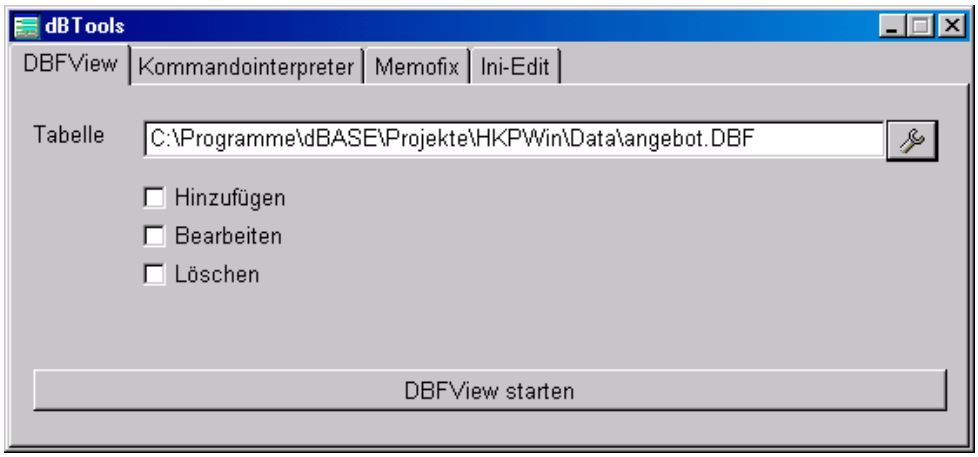
Trotz aller Bemühungen wird es sicher nicht gelingen, ein umfangreiches Programm gänzlich von Bugs zu befreien. Eine wesentliche Erleichterung für die Hotline ist die Verwendung der Eigenschaften zur Fehlerbehandlung, die ab dB2K 0.4 zur Verfügung stehen.
Hier empfehle ich dringend, für jede Anwendung eine Fehlerdatei vorzusehen, in welche alle nicht vom Programm abgefangenen Fehlermeldungen protokolliert werden. Treten Fehler auf, erhält man meist keine sachdienliche Antwort, wird jedoch für die Fehlerprotokollierung eine Datei verwendet, hat man die entsprechenden Informationen parat. Programmdatei, Routine, Zeilennummer und Fehlernummer werden vollständig in die Datei geschrieben, selbst bei echten Abstürzen erfolgt noch ein Eintrag in die Log_Datei.

Ist das Programm erstmal am Laufen, funktionieren die einzelnen Module meist ohne Weiteres. Die meisten Probleme treten beim Starten der Anwendung auf. Aus diesem Grunde unterstützen meine Anwendungen eine Protokollierung des Startverlaufs. Hierzu steht die Klasse `LC_STARTLOG.CC` zur Verfügung. Nach kritischen Programmpassagen wird eine Zeile in die Protokolldatei geschrieben. Ist die Anmeldeprozedur komplett durchgelaufen, können mittels der Methode *GetEnvironment()* auch noch alle Statusangaben in die Datei geschrieben werden.

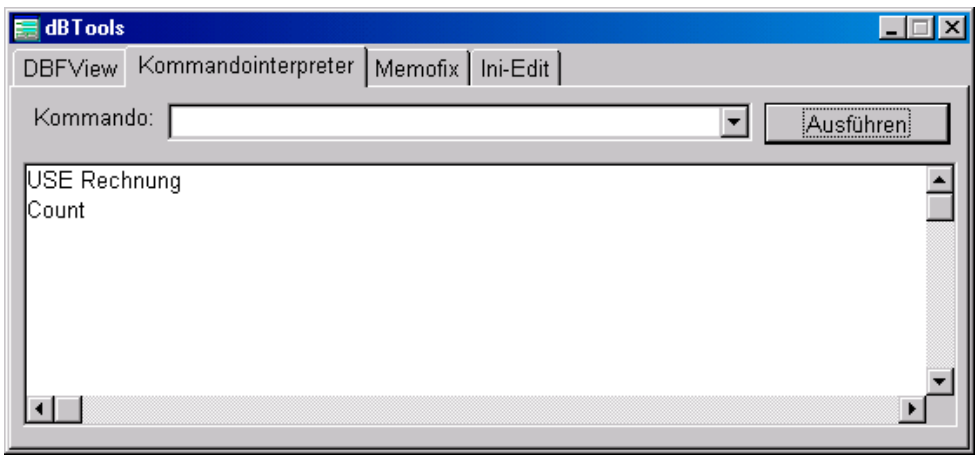
Zur Unterstützung des Anwenders in Zusammenarbeit mit der Hotline gibt es schließlich für alle Programme das Modul LC_DBTool.CC.



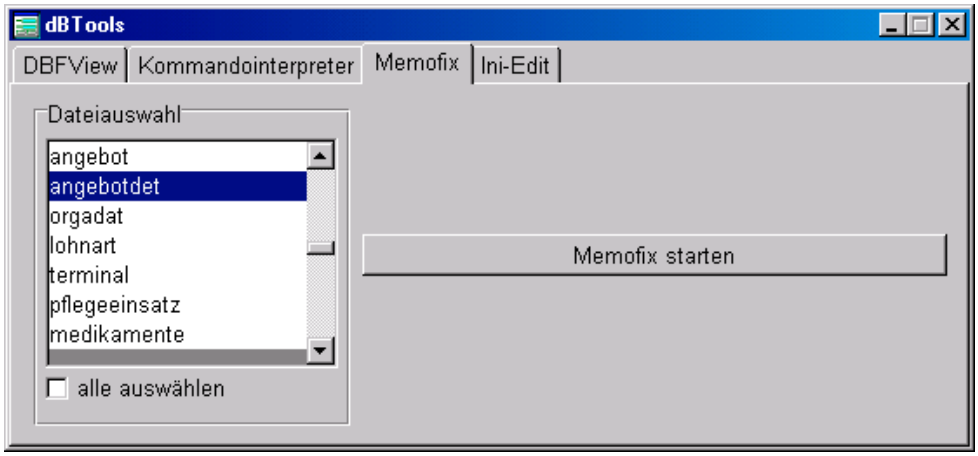
Weil mit dem Dialog u.a. direkte Manipulationen an allen Tabellen und der Initialisierungsdatei vorgenommen werden können, ist der weitere Zugriff erst nach Eingabe eines Freigabecodes möglich.
Wird die Freigabe erteilt, werden folgende Funktionen verfügbar:



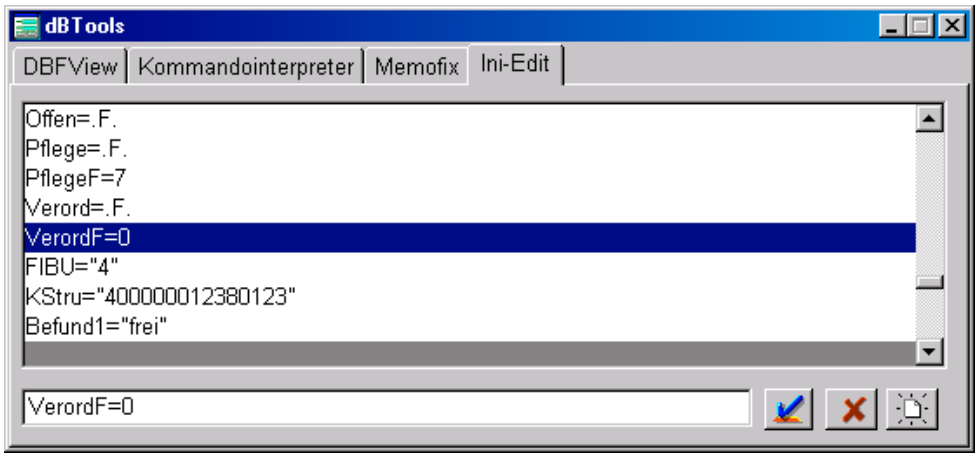
Aus diesem Dialog wird das externe Programm DBFView gestartet. Die Kontrollfelder dienen zum Vermitteln der erforderlichen Aufrufparameter.



Der integrierte Kommandointerpreter hat die gleiche Funktionalität wie das externe Modul dBCmd.EXE. Das Modul dient zur Eingabe und Ausführung von xBASE Kommandos, ähnlich dem Befehlsfenster von dB2K, mit der Einschränkung, daß Speichervariablen nicht verwendet werden können.



Mit diesem Tool, welches auch als externe Anwendung `Memofix.EXE` zur Verfügung steht, werden alle Felder – nicht nur vom Typ *Memo* - in allen Datensätzen der ausgewählten Dateien überprüft. Ungültige Feldinhalte werden, wenn möglich, mit definierten Vorgaben ersetzt. Es wird die Protokolldatei `CORRUPT.HTML` im Datenverzeichnis geschrieben.



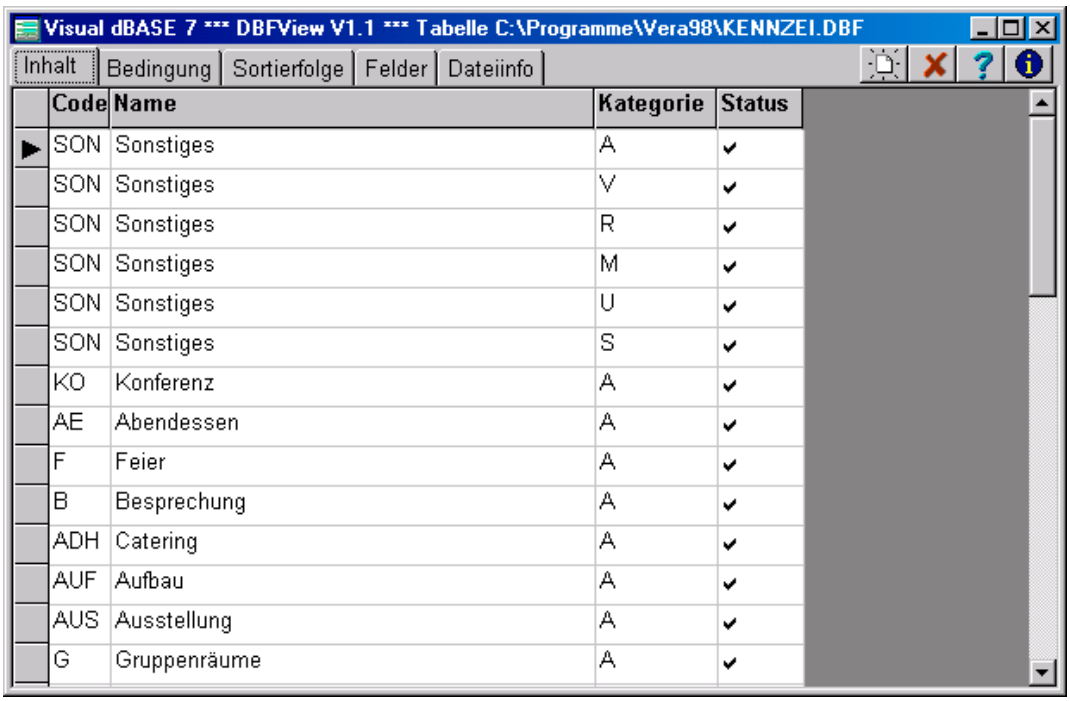
In diesem Dialog wird die Initialisierungsdatei der Anwendung bearbeitet. Dabei ist das Bearbeiten, Hinzufügen und Löschen von Einträgen möglich.

Zusätzliche Tools

Mit allen Anwendungen werden folgende Module geliefert:
- dBCmd.EXE
  Externe Version des Kommandointerpreters
- Memofix.EXE
  Externe Version des Tools zur Tabellenüberprüfung und Berichtigung

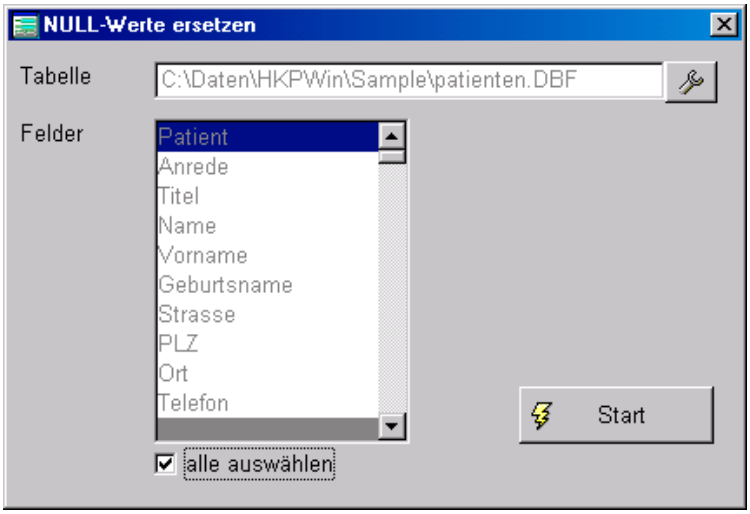
- DBFView.EXE

Der Tabellenbetrachter DBFVIEW.EXE ermöglicht Einsicht in dBASE-Tabellen aller Versionen. Beim Aufruf mit dem Parameter /d können Datensätze gelöscht, mit dem Parameter /a angefügt und mit dem Parameter /e bearbeitet werden. Der Aufruf mit /a schließt das Recht zum Bearbeiten von Datensätzen ein.
Bei Auswahl der Register können Datensätze mit verknüpften Bedingungen ausgewählt werden, die Sortierfolge läßt sich anhand der vorhandenen Indexe schalten, gleichzeitig stehen alle Indexinformationen zur Verfügung. Außerdem sind die Feldinformationen (Name, Typ, Länge, Dezimalstellen) und weitere Parameter der Tabelle verfügbar.



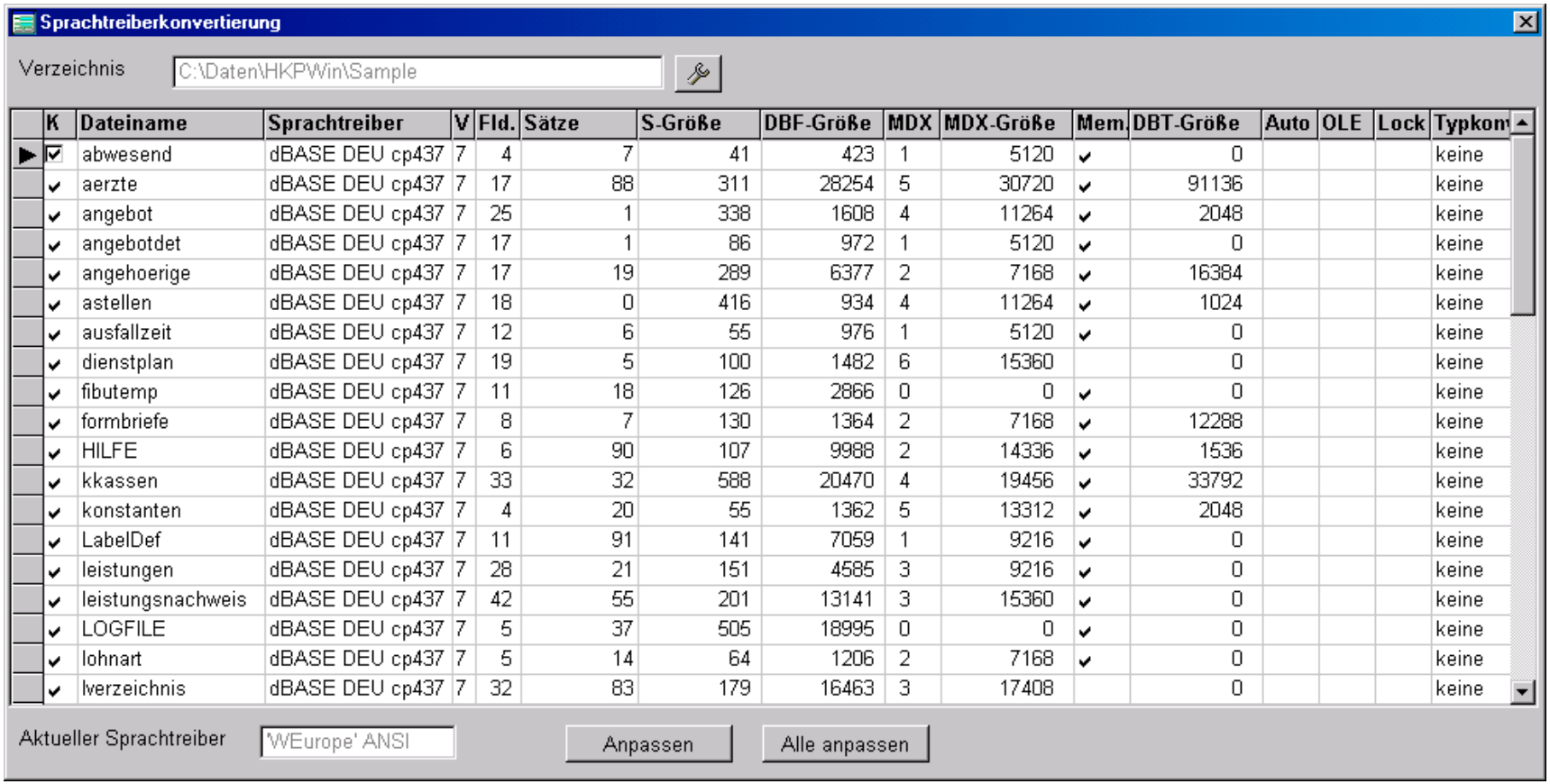
Code	Name	Kategorie	Status
SON	Sonstiges	A	✔
SON	Sonstiges	V	✔
SON	Sonstiges	R	✔
SON	Sonstiges	M	✔
SON	Sonstiges	U	✔
SON	Sonstiges	S	✔
KO	Konferenz	A	✔
AE	Abendessen	A	✔
F	Feier	A	✔
B	Besprechung	A	✔
ADH	Catering	A	✔
AUF	Aufbau	A	✔
AUS	Ausstellung	A	✔
G	Gruppenräume	A	✔

- Killnull.EXE



Mit diesem Dialog werden nicht definierte Feldwerte mit Standardangaben initialisiert, Zeichen und Memofelder mit einem leeren String; logische Felder mit dem Wert *false*; numerische, Gleitkomma, Long- und Double-Felder mit dem Wert *0*.

- LDriver.EXE
  Bei der Übernahme von vorhanden Datenbeständen basieren die Tabellen häufig noch auf einem DOS Sprachtreiber, z.B. dBASE DEU CP 437 bzw. CP 850. Somit muß bei jedem Datenzugriff eine interne Konvertierung der Feldinhalte erfolgen. Außerdem sind einige Zeichen, z.B. € gar nicht darstellbar bzw. werden nicht in den Tabellen oder Memofeldern gespeichert.
  Mit dem Modul `LDRIVER.EXE` können ausgewählte Tabellen an den in der BDE-Verwaltung eingestellten globalen Sprachtreiber angepaßt werden. Die Indexdateien werden neu aufgebaut. Neben der Konvertierung der Tabellendaten werden selbstverständlich auch die Inhalte von Memofeldern konvertiert.



	K	Dateiname	Sprachtreiber	V	Fld.	Sätze	S-Größe	DBF-Größe	MDX	MDX-Größe	Mem	DBT-Größe	Auto	OLE	Lock	Typkon
▶	☑	abwesend	dBASE DEU cp437	7	4	7	41	423	1	5120	✔	0				keine
	✔	aerzte	dBASE DEU cp437	7	17	88	311	28254	5	30720	✔	91136				keine
	✔	angebot	dBASE DEU cp437	7	25	1	338	1608	4	11264	✔	2048				keine
	✔	angebotdet	dBASE DEU cp437	7	17	1	86	972	1	5120	✔	0				keine
	✔	angehoerige	dBASE DEU cp437	7	17	19	289	6377	2	7168	✔	16384				keine
	✔	astellen	dBASE DEU cp437	7	18	0	416	934	4	11264	✔	1024				keine
	✔	ausfallzeit	dBASE DEU cp437	7	12	6	55	976	1	5120	✔	0				keine
	✔	dienstplan	dBASE DEU cp437	7	19	5	100	1482	6	15360		0				keine
	✔	fibutemp	dBASE DEU cp437	7	11	18	126	2866	0	0	✔	0				keine
	✔	formbriefe	dBASE DEU cp437	7	8	7	130	1364	2	7168	✔	12288				keine
	✔	HILFE	dBASE DEU cp437	7	6	90	107	9988	2	14336	✔	1536				keine
	✔	kkassen	dBASE DEU cp437	7	33	32	588	20470	4	19456	✔	33792				keine
	✔	konstanten	dBASE DEU cp437	7	4	20	55	1362	5	13312	✔	2048				keine
	✔	LabelDef	dBASE DEU cp437	7	11	91	141	7059	1	9216	✔	0				keine
	✔	leistungen	dBASE DEU cp437	7	28	21	151	4585	3	9216	✔	0				keine
	✔	leistungsnachweis	dBASE DEU cp437	7	42	55	201	13141	3	15360	✔	0				keine
	✔	LOGFILE	dBASE DEU cp437	7	5	37	505	18995	0	0	✔	0				keine
	✔	lohnart	dBASE DEU cp437	7	5	14	64	1206	2	7168	✔	0				keine
	✔	lverzeichnis	dBASE DEU cp437	7	32	83	179	16463	3	17408		0				keine