5

Das Projekt KLASTAMM: 1. Abschnitt

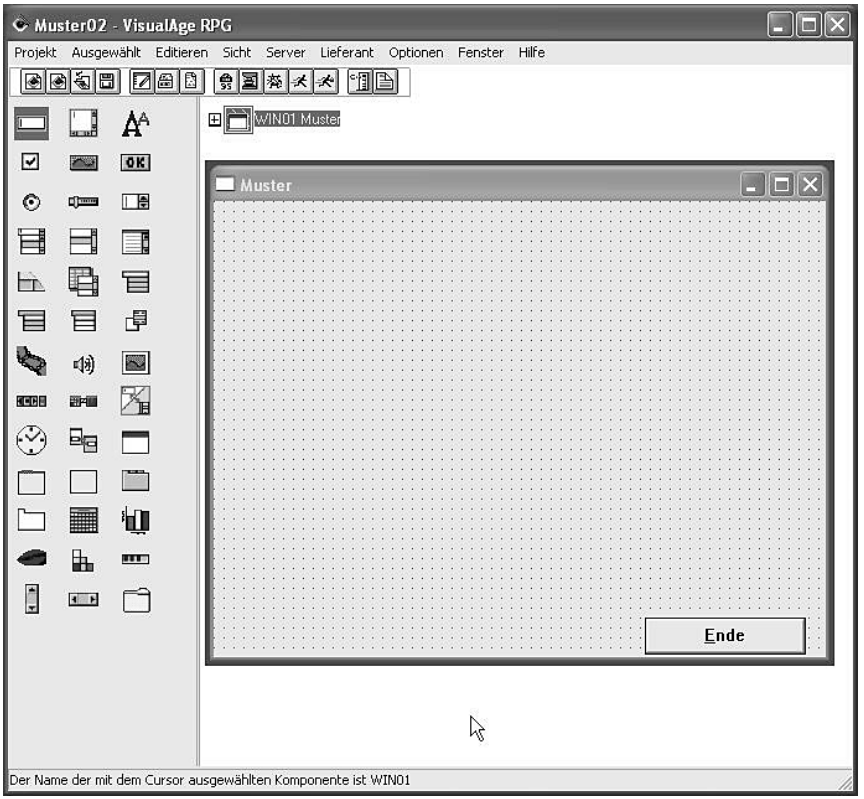
Machen wir uns jetzt also an die Arbeit, um unser erstes richtiges VARPG-Projekt durchzuführen. Unser Projekt wird zuerst aus drei Schritten bestehen:

- Erzeugen des Projekts KLASTAMM aus unserem Musterprojekt.
- Design des Dialogs. Hier werden wir mit folgenden VARPG-Objekten arbeiten: Statusleiste und Subdatei.
- Kodierung: Hier werden wir gemeinsam die notwendige Logik erzeugen, damit unsere Anwendung das ausführt, was wir erwarten.

5. 1 Dialogdesign

Um beim Dialogdesign nicht ganz von vorn beginnen zu müssen, öffnen Sie ein neues GUI-Projekt. Laden Sie dann das Musterprojekt über das Menü „Projekt –> Öffnen“.

Jetzt muss Ihr Desktop etwa folgendermaßen aussehen:



Musterprojekt geladen

Nachdem das Musterprojekt geladen wurde, sichern Sie es sofort wieder unter den Namen KLSTAMM.

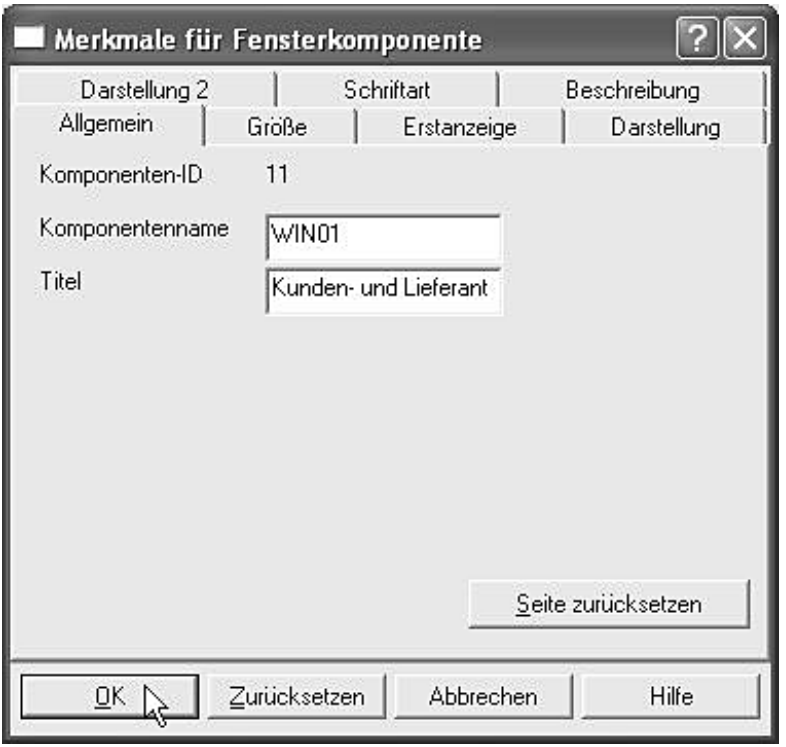
Die Aufgabe unseres ersten Projektabschnitts besteht darin, den grundsätzlichen Dialog für die Verarbeitung von Kunden-, Lieferanten- und Ansprechpartnerdaten zu entwerfen.

Dazu sollen folgende Dialog-Elemente auf dem Basis-Fenster angeordnet werden:

- eine **Statusleiste**, die den Firmennamen wiedergibt und deren Kunden-, Lieferanten und Ansprechpartner derzeit bearbeitet werden,
- ein **Menü**, über dessen Inhalt wir uns derzeit noch keine Gedanken machen werden,
- eine **Subdatei**, in der Kunden- und Lieferantendaten in tabellarischer Form angezeigt werden.

5.1.1 Die Fenster-Komponente

Ich öffne den Dialog „Merkmale für Fensterkomponente“.



Merkmale WIN01

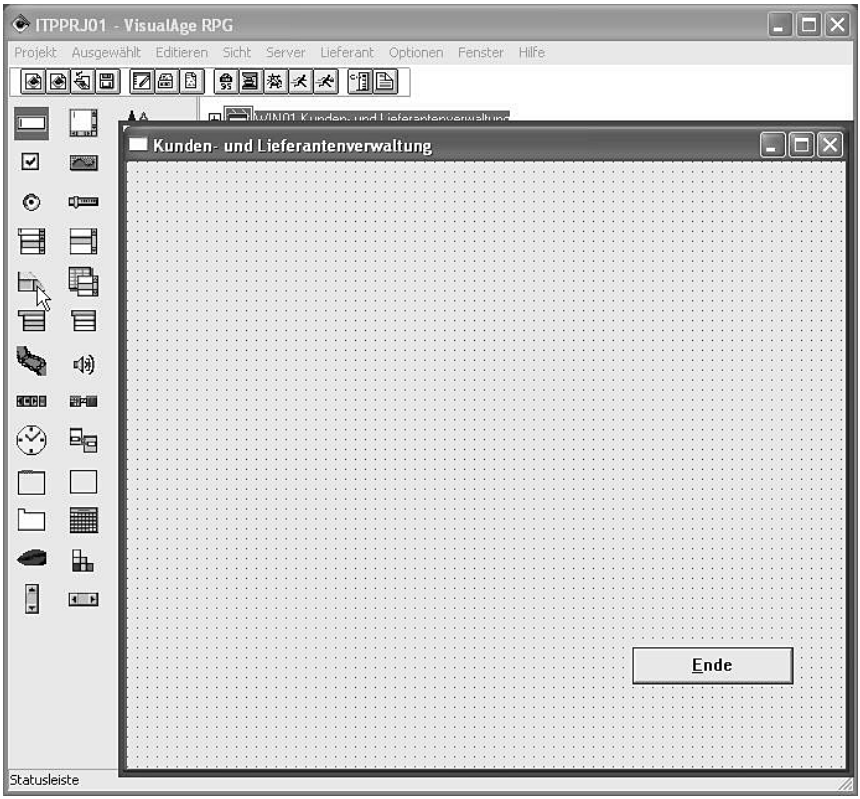
Wie Sie sehen, ist aus dem Musterprojekt der „Komponentenname“ WIN01 übernommen worden.

Den Titel des Fensters – das Label-Merkmal – versorge ich mit dem Wert „Kunden- und Lieferantenverwaltung“.

5.1.2 Die Statusleiste

Als nächstes Element werde ich eine Statusleiste auf dem Fenster erzeugen.

Dazu wähle ich folgendes Element – „Statusleiste“ – aus der Komponentenpalette aus: 

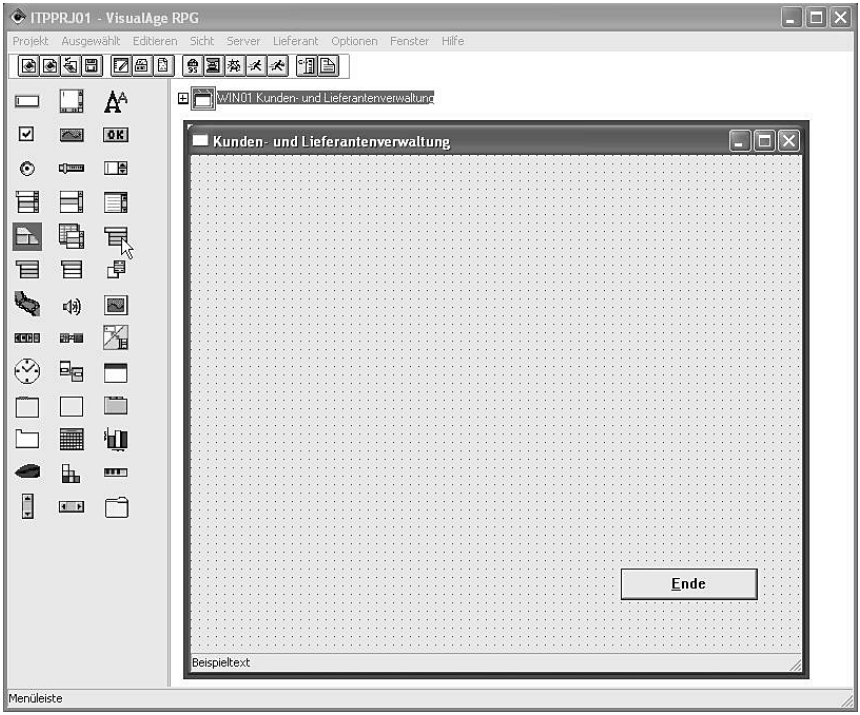


Statusleiste

Sodann wird die Statusleiste mit der Maus in das Fenster gezogen. Sie legt sich automatisch am Fuße des Fensters ab. Diese Position kann nicht beeinflusst werden.

Merkmale Statusleiste

Ein Doppelklick auf die „Statusleiste“ öffnet den Dialog „Merkmale für Statusleistenkomponente“. Hier ändere ich lediglich den Komponentenname in SBRSTS.



Statusleiste im Fenster

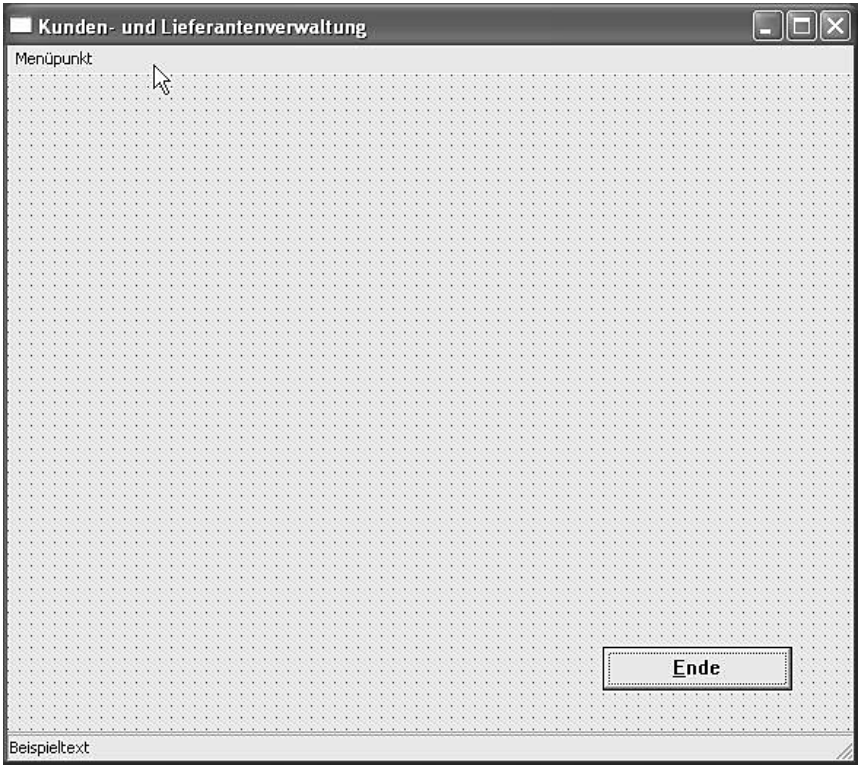
In obiger Abbildung sehen Sie die im Fensterfuß positionierte Statusleiste, in der die Zeichenfolge „Beispieltext“ zu sehen ist.

Zur Laufzeit unseres Programms werden wir in die Statusleiste einen eigenen Text hineinstellen.

5.1.3 Das Menü

Jetzt wähle ich aus der Komponentenpalette folgendes Symbol – die „Menüleiste“ – aus: 

Die Menüleiste ziehe ich über das Fenster und sie platziert sich automatisch am Fensterkopf.



Menüleiste

In der Abbildung oben sehen Sie die Menüleiste. Bestandteil dieser Menüleiste ist ein Menüpunkt, der hier mit dem Platzhaltertext „Menüpunkt“ angezeigt wird.

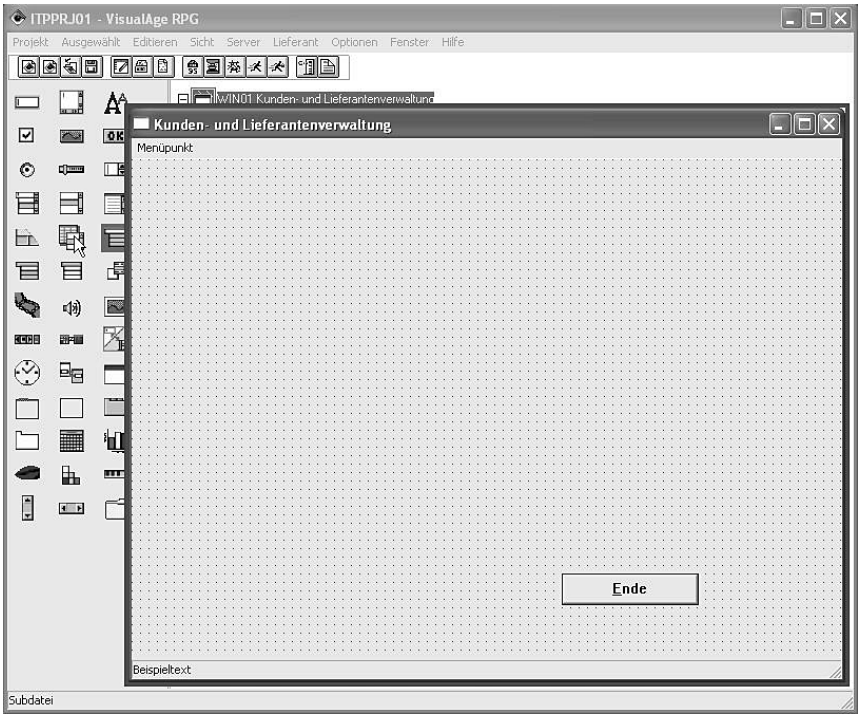
In diesem ersten Projektabschnitt wird das Menü lediglich aus designtechnischen Gründen auf dem Fenster abgelegt. Würden wir z. B. zuerst weitere Komponenten auf dem Fenster platzieren und danach die Menüleiste einfügen, so wird sich der Fensteraufbau insgesamt nach unten verschieben, so dass Komponenten, die nach dem Rand positioniert wurden, zumindest teilweise in den nicht sichtbaren Bereich verlagert werden.

Ich werde jetzt der Menüleiste einen neuen Komponentennamen zuordnen. Diesmal wird jedoch nicht nach einem Doppelklick auf die Menüleiste der Merkmale-Dialog angezeigt.

Die Bearbeitung von Menüs erfolgt aus der Projektverwaltung heraus. Ich suche also im Komponentenbaum der Projektverwaltung die „Menüleiste“. Durch einen Doppelklick öffnet sich jetzt der Dialog „Merkmale für Menüleistenkomponente“.

Menüleisten-Merkmale

Ich ändere den Komponentennamen in MNBAR.



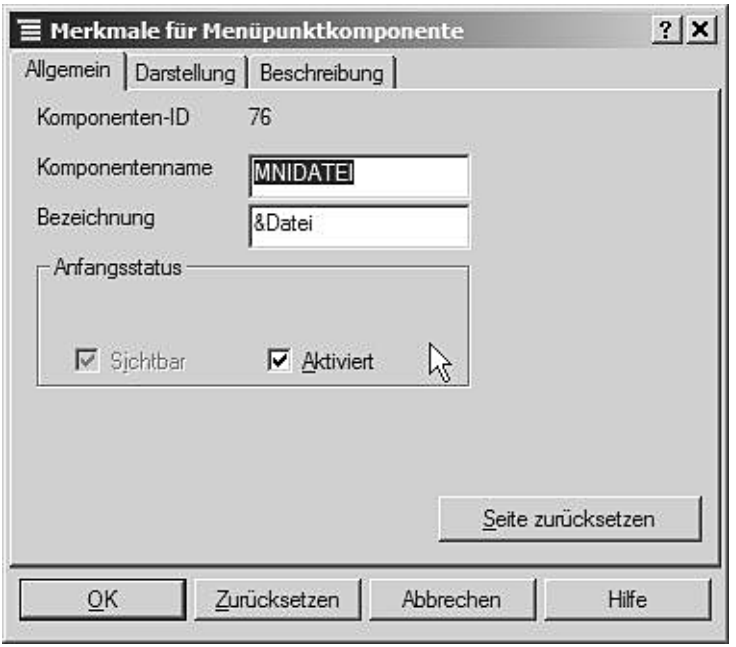
Subdatei-Komponente

Ich ziehe danach noch zwei weitere Menüpunkte  auf die Komponente MNBBAR in der Projektverwaltung.

Durch Doppelklick auf die Menüpunkte öffne ich die entsprechenden Merkmale-Dialoge und benenne die Menüpunktkomponenten mit folgenden Namen:

- MNIDATEI
- MNIBEARB
- MNISRV

Hier sehen Sie den Merkmale-Dialog für den Menüpunkt MNIDATEI:



Menüpunkt MNIDATEI

Im Parameter „Bezeichnung“ – Label-Merkmal – wird die Zeichenfolge beschrieben, die zur Laufzeit als Menüauswahl zu sehen ist. Beachten Sie das &-Zeichen vor dem „D“ von „Datei“. Zur Laufzeit des Programms hat der Anwender die Möglichkeit, mit „Alt-D“ diesen Menüpunkt direkt aufzurufen.

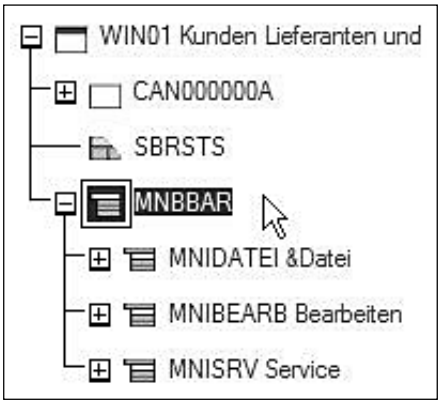
Nachfolgend sehen Sie den Merkmale-Dialog für den Menüpunkt MNIBEARB:

Menüpunkt MNIBEARB

Und anschließend der Merkmale-Dialog für den Menüpunkt MNISRV:

Menüpunkt MNISRV

Zur Übersicht sehen wir uns den Komponentenbaum aus der Projektverwaltung an:

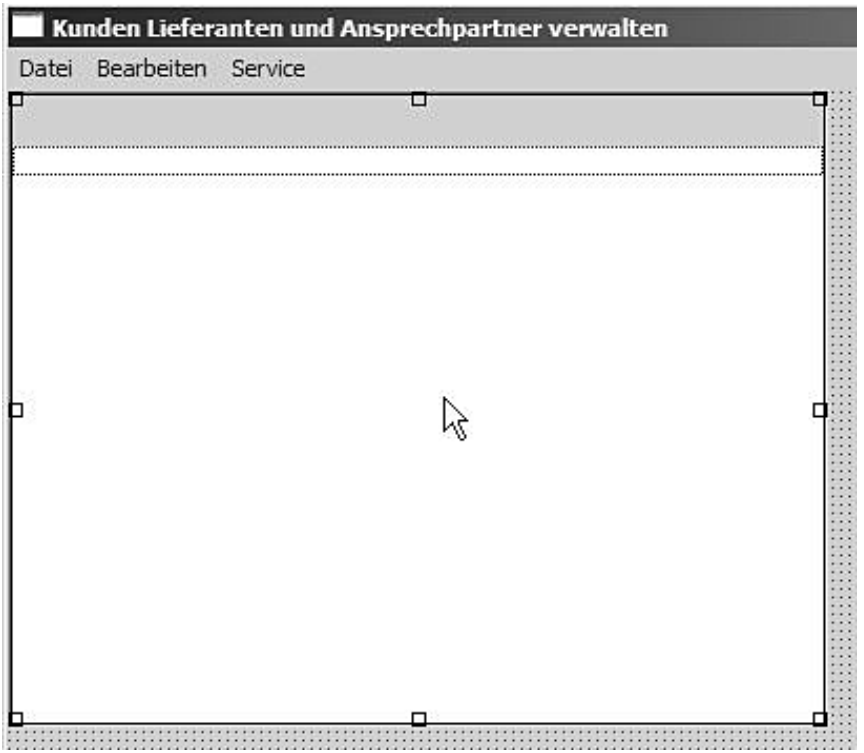


Das Menü in der Übersicht

5.1.4 Die Subdatei

Jetzt werde ich eine Subdatei im Fenster erzeugen. Dazu wähle ich in der Komponentenpalette das Symbol für eine Subdatei aus:

Dieses Symbol ziehe ich über das Fenster.



Eine leere Subdatei

Ich positioniere und dimensioniere die Subdatei-Komponente so, dass sie etwa das obere linke Viertel des gesamten Fensters belegt. Das Fenster insgesamt ziehe ich soweit auf, dass es fast den gesamten Desktop überlagert, denn wir werden in den nächsten Projektabschnitten noch einigen Platz benötigen.

Ein Doppelklick auf die „Subdatei-Komponente“ öffnet den Merkmale-Dialog für die Subdatei.

Register „Allgemein“ – Komponentenname: SUBF01

Im Parameter „Komponentenname“ des Registers „Allgemein“ trage ich folgenden Wert ein: SUBF01.

Danach öffne ich das Register „Darstellung“:

Register „Darstellung“

Hier ist die Parametergruppe „Auswahlart“ interessant:

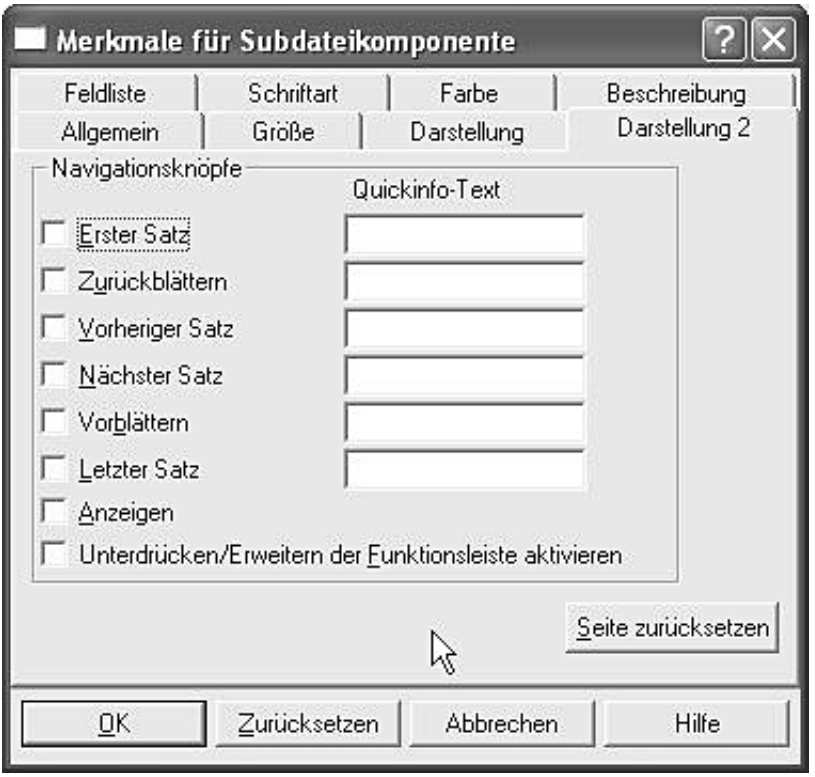
- **Einzelne**: Es können nur einzelne Sätze in der Subdatei ausgewählt werden.

- **Mehrere**: Es können mehrere Sätze in der Subdatei ausgewählt werden. Diese Sätze müssen jedoch zusammenhängend ausgewählt werden. Mit „Shift+Mausklick“ markieren Sie den ersten Satz einer Gruppe, dann wählen Sie mit „Shift+Mausklick“ den letzten Satz einer Gruppe aus. Anschließend werden alle Sätze, die zwischen diesen beiden liegen, ebenfalls markiert.

- **Erweitert**: Es können mehrere Sätze in der Subdatei markiert werden, sie müssen jedoch nicht benachbart liegen. Mit „Strg+Mausklick“ markieren Sie jeden in Frage kommenden Satz.

Ich belasse es bei dem Wert „Einzelne“.

Im Merkmal-Register „Darstellung2“ können dann Navigationsknöpfe zur Datensatznavigation erzeugt werden.



Navigationsknöpfe

Im Register „Feldliste“ werden alle in der Subdatei befindlichen Felder angezeigt.

Register „Feldliste“

Sie sehen, dass derzeit die Feldliste leer ist. In einem der folgenden Arbeitschritte werden wir Felder in diese Liste einfügen.

Im Register „Schriftart“ bestimmen Sie den Font und die Punktgröße sowie die Farbe der in der Subdatei benutzten Schriftarten.

Register Schriftart

Sinn und Zweck dieser ersten Subdatei bestehen darin, Kunden- und Lieferantenkenndaten anzuzeigen. Ganz im Sinne eines iSeries-Subdatei-Bildschirms sieht der Anwender beim Start des Programms eine Reihe von Kunden- und Lieferantensätzen, die dann, je nachdem welcher Satz ausgewählt wurde, weiter bearbeitet werden können.

Wir werden jetzt Felder interaktiv in die Subdatei aufnehmen. Da wir Kunden- und Lieferantenfelder in der Subdatei anzeigen, benutzen wir die Datei SVKLST00 in der Bibliothek EPSSVFIL20 sozusagen als Feldreferenzdatei.

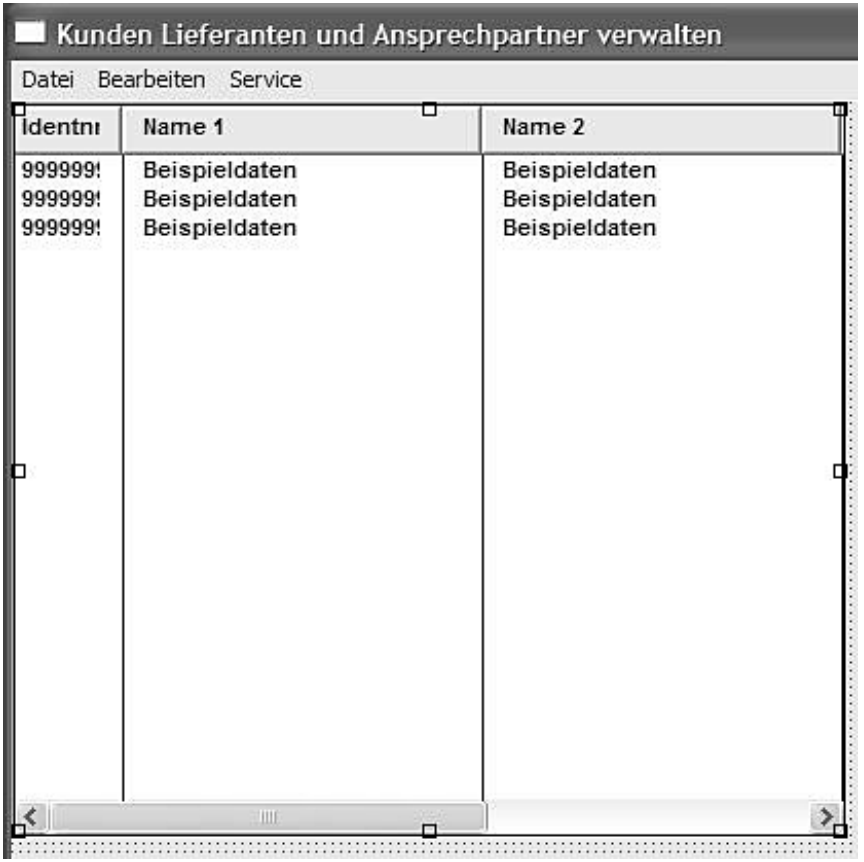
Ich expandiere im Dialog „Dateihierarchie“ den Eintrag „Server“:

Server –> EPSSVFI20 –> SVKLST00 –> SVKLST00

Im Dialog „Felder“ markiere ich folgende Einträge:

- KLFINR
- KLIDNR
- KLNAM1
- KLNAM2
- KLNAM3
- KLTEL

Diese markierten Felder ziehe ich jetzt mit der Maus in die Subdatei. So werden die Felder in die Subdatei aufgenommen.



Subdatei mit Feldern

Oben sehen Sie die beispielhafte Darstellung der in die Subdatei aufgenommenen Dateifelder.

Ich öffne den Merkmale-Dialog für die Subdatei SUBF01 und öffne das Register „Feldliste“.

Ich werde jetzt das Feld KLFINR bearbeiten.

Merkmal – Register „Feldliste“ der Subdatei SUBF01

Dazu markiere ich das Feld KLFINR in der Feldliste und klicke auf den Knopf „Ändern…“.

Jetzt erscheint der Merkmale-Dialog für das Subdateifeld KLFINR.

Merkmal – Register „Allgemein“ des Subdateifeldes KLFINR

Sie sehen im Parameter „Spaltenname“, dass der Feldname der iSeries-Datei SVKLST00 übernommen wurde. In den Parameter „Feldüberschrift“ ist die Spaltenüberschrift der iSeries-Datei übernommen worden.

Das Subdateifeld KLFINR soll später nicht in der Subdatei angezeigt werden. Wir können jedoch nicht einfach dieses Feld aus der Feldliste der Subdatei SUBF01 löschen, da wir den Inhalt dieses Feldes für eine weitere Verarbeitung des Datensatzes als Schlüsselbegriff benötigen.

Ich öffne das Register „Darstellung“.

Merkmal – Register „Darstellung“ des Subdateifeldes KLFINR

Im Rahmen „Optionen“ markiere ich „Verdeckt“. Damit wird KLFINR aus der Anzeige der Subdatei herausgenommen.

Beachten Sie den Parameter „Spaltenbreite“. Sollte der dargestellte Beispielwert in der Subdatei abgeschnitten bzw. zu groß erscheinen, so können Sie die Breite der Spalte an die Länge des Inhalts anpassen.

Die Reihenfolge der Felder in der Subdatei sollte wie folgt sein:

1. KLFINR – verdeckt
2. KLIDNR
3. KLNAM1
4. KLNAM2
5. KLNAM3
6. KLTEL

5. 2 Verarbeitungslogik

Jetzt kümmern wir uns um die Verarbeitungslogik des Programms. Zuerst einmal die positive Nachricht: Um die Beendigung des Programms KLASTAMM brauchen wir uns nicht mehr zu kümmern. Da KLASTAMM durch eine Kopie des Projekts „Muster02“ entstanden ist, wurde dessen Beendigungslogik komplett übernommen.

Die negative Nachricht: Um den Rest müssen wir uns selbst kümmern.

5.2.1 Die Statusleiste

Beginnen wir mit der Statusleiste. In der Datei SVFIST00 sind alle Firmen eingetragen, die mittels unseres Programms verwaltet werden. Einer der Firmensätze bezeichnet die so genannte Anfangsfirma. Das ist die Firma, deren Sätze anfänglich verarbeitet werden sollen. Im Datensatz des Firmenstammes SVFIST00 befindet sich das Feld FISTAT. Steht in diesem Feld der Wert 'A', so handelt es sich bei diesem Satz um die Anfangsfirma.

In unserem ersten Verarbeitungsschritt muss diese Anfangsfirma ermittelt werden. Danach wird der komplette Name der Firma, der sich in den Feldern FINAM1, FINAM2 und FINAM3 befindet, verkettet und in der Statusleiste des Windows angezeigt.

5.2.1.1 CREATE-Ereignis

Da das Ermitteln der Anfangsfirma im Programm eine einmalige Angelegenheit ist, benötigen wir eine Art Vorlaufroutine, in der die oben beschriebene Verarbeitung durchgeführt wird.

Im VARPG gibt es eine besondere Routine, die mit dem Namen *INZSR beschrieben ist. Diese Initialisierungs-Subroutine könnten wir an dieser Stelle nutzen. Ich bin jedoch kein Freund dieser Standard RPG-Logik und versuche mich, soweit es dem VARPG möglich ist, der objektorientierten Logik anzupassen.

Jede grafische Komponente kann man als ein Objekt verstehen. Wenn ein Objekt mit dem Start des Programms in den Hauptspeicher geladen wird, so wird ein CREATE-Ereignis ausgelöst. Abhängig von diesem Ereignis kann eine Aktion ausgelöst werden: die oben beschriebene Verarbeitung.

Und genau diesen Umstand werden wir ausnutzen.

☞ **Wichtig!** Jede Komponente löst ein CREATE-Ereignis aus. Also kann die Initialisierung einer Komponente innerhalb des eigenen CREATE-Ereignisses ausgeführt werden.

Damit kennen Sie schon zwei Ereignisse, die allen Komponenten eigen sind: Sie lösen alle ein CREATE- und alle ein DESTROY-Ereignis aus.

Weiterhin müssen wir eine F-Bestimmung für unser Programm anlegen, in der die Datei SVFIST00 definiert wird.

Die F-Bestimmung sieht wie folgt aus:

```
DName++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++
fsvfist00  if   e           k disk    remote
f                                     rename(svfist00:fistfmt00)
```

F-Bestimmung für SVFIST00

Wie Sie sehen, ist diese F-Bestimmung grundsätzlich nicht anders aufgebaut als eine F-Bestimmung im iSeries RPG – mit einer Ausnahme: Beachten Sie das Schlüsselwort REMOTE. Dieses Schlüsselwort besagt, dass die Datei sich auf dem mit dem Projekt verbundenen Server befindet.

Damit das Programm später kompiliert und ausgeführt werden kann, muss sich diese Datei in einer Bibliothek befinden, die sich wiederum zum Zeitpunkt der Kompilation sowie zur Laufzeit in der Bibliotheksliste des Benutzers befindet, der die Verbindung zur iSeries hergestellt hat.

Da die Datei SVFIST00 und auch alle anderen Dateien, die wir noch verarbeiten werden, NULL-fähige Felder erlauben, sollte noch folgende H-Bestimmung in die Programmquelle aufgenommen werden.

☞ **Achtung!** Für die Datei SVFIST00 ist ein RENAME auf das Satzformat notwendig, da in dieser Datei Dateiname und Satzformatname identisch sind.

```
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 ***************************************************************
 *
H alwnull(*usrctl)
```

H-Bestimmung für KLASTAMM

Nach diesen Präliminarien wenden wir uns der Programmierung der CREATE-Ereignisroutine für die Statusleiste SBRSTS zu.

Mit der rechten Maustaste fordere ich das „Kontextmenü“ für SBRSTS an und wähle „Ereignisse –> CREATE“.

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++
 *
C     SBRSTS        BEGACT    CREATE        WIN01
c                   exsr      HoleFirma
C                   ENDACT
```

CREATE-Ereignisroutine für SBRSTS

Sie sehen, dass ich in der Ereignisroutine lediglich ein EXSR kodiert habe und nicht die ganze Verarbeitungslogik zum Auffinden der Anfangsfirma.

Sie sollten aus Übersichtlichkeitsgründen – so weit wie möglich – die Verarbeitungslogik des Programms, die ja nicht VARPG-spezifisch ist, von der Aktionslogik, die VARPG-spezifisch ist, trennen.

Aus diesem Grund habe ich die komplette Verarbeitungslogik zum Auffinden der Anfangsfirma in die Subroutine „HoleFirma“ gelegt.

Die Subroutine „HoleFirma“ habe ich ganz an den Anfang des Programms vor die erste Aktionssubroutine gelegt. So gliedert sich mein Programm in der Folge in klar abgegrenzte Blöcke:

1. F-Bestimmungen für Dateidefinitionen
2. D-Bestimmungen für alle sonstigen Deklarationen

3. C-Bestimmungen, die alle Benutzersubroutinen beinhalten
4. C-Bestimmungen, die alle Aktionssubroutinen beinhalten

Die Verarbeitungslogik als solche ist schnell erklärt. Ich lese einfach sequentiell alle Sätze der Datei SVFIST00 solange, bis der Satz der Anfangsfirma (Feld FISTAT = 'A') gefunden wurde.

Hier ist die kodierte Logik:

```
CSRN01Factor1++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq
C     HoleFirma     BEGSR

C                   read      svfist00

C                   dow       not %eof(svfist00)

C                   if        FiStat = 'A'
c                   eval      %setatr('win01':'sbrsts':'sblabel') =
c                             %trim(FiNam1) + ' ' +
c                             %trim(Finam2) + ' ' +
c                             %trim(finam3)
c                   endif

C                   read      svfist00

c                   enddo

C                   ENDSR
```

Benutzersubroutine „HoleFirma“

5.2.1.2 %SETATR-Funktion

Beachten Sie die EVAL-Anweisung hinter der IF-Abfrage:

```
CSRN01Factor1+++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
c                   eval      %setatr('win01':'sbrsts':'sblabel') =
c                             %trim(FiNam1) + ' ' +
c                             %trim(Finam2) + ' ' +
c                             %trim(finam3)
```

Qualifizierte Wertzuweisung auf ein Komponentenmerkmal

Diese Anweisung bewirkt, dass in der Statusleiste des Fensters der verkettete Name der Anfangsfirma ausgegeben wird.

Um ein Merkmal einer Komponente zu setzen oder zu verändern, verwenden wir die VARPG-Funktion %SETATR. Diese Funktion hat folgenden schematischen Aufbau:

%SETATR('Fenster':'Komponente':'Merkmal') = Wert

- **Fenster**: Der Name des Fensters, auf dem sich die angesprochene Komponente befindet. Hier das Fenster WIN01.

- **Komponente**: Die Komponente, deren Merkmal wir setzen oder ändern wollen. Hier die Statusleiste SBRSTS.

- **Merkmal**: Das zu setzende oder zu ändernde Merkmal. Hier das Merkmal SBLABEL

- **Wert**: Der dem Merkmal zugewiesene Wert. Hier die verketteten Felder FINAM1 bis FINAM3.

Der Vollständigkeit halber sei hier erwähnt, dass es auch einen SETATR Op-Code gibt. Ich werde jedoch diesen Op-Code nicht verwenden, da ich immer dann, wenn es alternativ zu einem Op-Code eine Built-in-Function gibt, diese nutzen werde.

Werden also in den weiteren Arbeitsschritten Wertzuweisungen auf Komponenten-Merkmale durchgeführt, so wird dafür die Funktion &SETATR verwendet.

Ich kompiliere das Programm. Danach rufe ich es zum Testen auf.

Wie Sie sehen, wird in der Statusleiste der Firmenname der aktiven Firma angezeigt.

5.2.2 Die Subdatei

Jetzt muss die Subdatei mit Daten aus dem Kunden- und Lieferantenstamm – SVKLST00 – befüllt werden.

Mit dem Start des Programms sollen die ersten 50 Sätze aus der Datei SVKLST00 in die Subdatei SUBF01 geladen werden.

5.2.2.1 Datei deklarieren

Dazu muss die Datei SVKLST00 deklariert werden.

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++
fsvklst00  if   e           k disk    remote
f                                     rename(svklst00:klstfmt00)
```

F-Bestimmung für SVKLST00

5.2.2.2 Subdatei laden

Das Laden der Subdatei mit den ersten 50 Sätzen aus der Datei SVKLST00 erfolgt aus dem CREATE-Ereignis der Subdatei SUBF01 heraus.

Die CREATE-Aktionsroutine:

```
CSRN01Factor1++++++++Opcode(E)+Factor2+++++++Result++++++++
C     SUBF01        BEGACT    CREATE        WIN01
c                   exsr      LadeSUBF01
C                   ENDACT
```

Aktionssubroutine für CREATE von SUBF01

Laderoutine für SUBF01:

```
CSRN01Factor1+++++++Opcode(E)+Factor2+++++++Result
C     LadeSUBF01    BEGSR

c                   for       zaehler = 1 to 50

c                   read      svklst00
c
c                   if        %eof(svklst00)
c                   leave
c                   endif

c                   write     SUBF01

c                   endfor

C                   ENDSR
```

Laderoutine für SUBF01

Für einen iSeries-Programmierer ist diese Lade-Subroutine bemerkenswert. Es wird keine relative Satznummer geführt! Diese ist in VARPG nicht notwendig.

Beachten Sie die Anweisung „WRITE SUBF01“. Diese Anweisung sorgt dafür, dass ein Satz aus dem Dateipuffer in den Dialogpuffer übertragen wird.

☞ **Folgende Regel gilt**: Ist ein Feld im Programm- oder Dateipuffer deklariert, so kann es per „WRITE“-Anweisung in eine Subdatei übertragen werden, wenn die Subdateifelder den gleichen Namen besitzen – wie die Programm- oder Dateifelder.

Dieser Umstand wurde hier ausgenutzt.

Ich kompiliere das Programm und führe einen Test durch.

5.2.2.3 Subdatei nachladen

Jetzt haben wir die ersten 50 Sätze in die Subdatei geladen. Wir wollen jedoch erreichen, dass, wenn der Anwender über das Ende der Subdatei hinausblättert, die nächsten 50 Sätze nachgeladen werden.

☞ **Wichtig!** Die VARP-Subdatei kennt keine Größenbeschränkungen. Es ist also durchaus möglich, mehr als 9.999 Sätze in eine Subdatei zu laden. Rein theoretisch ist es auch möglich, Millionen von Sätzen in eine VARPG-Subdatei zu laden. Ob das sinnvoll ist, sei dahingestellt.

Wie können wir jetzt unsere Subdatei SUBF01 nachladen? Ganz einfach! Blättert ein Anwender über das Ende einer Subdatei hinaus, so wird das Ereignis PAGEEND ausgelöst.

Ich kodiere also für das PAGEEND-Ereignis folgende Aktionsroutine:

```
CSRN01Factor1++++++Opcode(E)+Factor2+++++++Result
C      SUBF01        BEGACT     PAGEEND         WIN01
c                    exsr       LadeSUBF01
C                    ENDACT
```

Nachladen für SUBF01

Das ist alles!

Ich kompiliere das Programm und führe einen Test durch. Benutze ich den rechten Scrollbar zum Blättern, so wird PAGEEND ausgelöst. Gleiches gilt, wenn ich mit der Bild-ab- und der Pfeil-unten-Taste durch die Subdatei blättere.