1 Einleitung

Dieses Kapitel beschreibt kurz das Produkt DRIVE/WINDOWS, die Zielgruppe des Handbuchs sowie das Konzept der Handbuchreihe. Außerdem enthält es eine Liste mit Änderungen gegenüber der Vorgängerversion sowie eine Aufstellung der in den DRIVE/WINDOWS-Handbüchern verwendeten Darstellungsmittel.

1.1 Kurzbeschreibung des Produkts

DRIVE/WINDOWS ist eine Programmiersprache der 4. Generation (4GL) zur Entwicklung kommerzieller Client-Server-Anwendungen. Sie ist die 4GL für Zugriff auf Dateien oder die BS2000 Datenbanksysteme SESAM/SQL V1, SESAM/SQL V2 und UDS . DRIVE-Anwendungen können nach verschiedenen Client-Server-Architekturen auf heterogene Rechner verteilt werden, denn DRIVE/WINDOWS ist auf den Plattformen BS2000, SINIX und MS-Windows verfügbar und unterstützt die Verbindung von Client und Server optimal.

Die einheitliche Sprache mit mächtigen und leicht erlernbaren Anweisungen ermöglicht die Erstellung komplexer Anwendungen für Datenbankzugriff, Report, Oberfläche, Kommunikation und Verarbeitung. DRIVE/WINDOWS versorgt automatisch systemnahe Schnittstellen zu den jeweiligen Komponenten und nimmt dem Programmierer diese Arbeit ab.

Zum Test seiner DRIVE-Anwendung stellt DRIVE/WINDOWS dem Programmierer einen integrierten Debugger zur Verfügung.

DRIVE-Anwendungen können unabhängig vom Einsatz eines Transaktionsmonitors erstellt und getestet werden. Sie sind ohne Änderungen mit oder ohne Transaktionsmonitor ablauffähig.

Zur Steigerung der Performance können die erstellten DRIVE-Anwendungen mit dem Compiler DRIVE/WINDOWS-COMP übersetzt werden.

DRIVE-Server-Anwendungen unter SINIX ermöglichen Ihnen den Dateizugriff und Datenbankzugriff auf INFORMIX und können problemlos als Komponenten einer verteilten Anwendung mit DRIVE-Anwendungen unter BS2000 oder MS-Windows eingebunden werden. Auch hier gibt es, wie im BS2000, eine integrierte Reportfunktion und für performanten Ablauf der erstellten Server-Applikation einen Compiler.

Über die Erstellung von Anwendungen hinaus stellt DRIVE/WINDOWS unter MS-Windows komfortable Werkzeuge zur Verfügung, die voll in den Entwicklungsprozeß integriert sind. case/4/0 unterstützt den Entwurf einer Anwendung. Darauf aufsetzend ermöglicht DRIVE/DESIGNER den nahtlosen Übergang in die Codierphase und die konsistente Generierung und Montage von DRIVE-Quellprogrammen aus den Entwurfsergebnissen. Die Software-Produktionsumgebung von DRIVE/WINDOWS bietet dann für die Programmerstellung, den Test und und die Anwendungssteuerung eine komfortable grafische und menügestützte Oberfläche.

1.2 Zielgruppe

Das Handbuch wendet sich an Programmierer, die DRIVE-Anwendungen oder Teile von Client-Server-Anwendungen mit DRIVE/WINDOWS auf BS2000-Rechnern entwickeln. Dafür benötigt der Programmierer allgemeine Kenntnisse über das Betriebssystem BS2000.

Abhängig vom konkreten Einsatzfall sind weitere Kenntnisse erforderlich über:

– das Datenbanksystem UDS

– das Datenbanksystem SESAM

– den Transaktionsmonitor UTM

– das Format Handling System FHS zur Erstellung von Bildschirmen

– das Betriebssystem des Clients (MS-Windows oder SINIX)

1.3 Konzept der Handbuchreihe

Die Beschreibung von DRIVE/WINDOWS umfaßt hauptsächlich drei Handbücher:

– Das Handbuch „Programmiersystem“ [1] gibt einen allgemeinen Überblick über das System DRIVE/WINDOWS und erläutert die Funktionen, die der Vorbereitung und Sicherung, dem Testen und Ausführen von DRIVE-Programmen dienen. Es enthält auch die für den Systemverwalter notwendigen Informationen zur Einsatzvorbereitung von DRIVE/WINDOWS und über die Konfiguration von Client-Server-Anwendungen.

– Das Handbuch „Programmiersprache“ [2] beschreibt die Regeln für ein DRIVE-Programm. Es wird auf die Programmierlogik, die Erstellung von Bildschirm- und Listenformaten, die Gestaltung von Berichten mit dem Report-Generator und Client-Server-Anwendungen eingegangen.

– Das Handbuch „Lexikon der DRIVE-Anweisungen“ [3] enthält, alphabetisch sortiert, alle DRIVE-Anweisungen mit Syntax und der Beschreibung des gesamten Funktionsumfangs. Die SQL-Anweisungen sind in separaten Handüchern beschrieben (s.u.).

  Die Beschreibung der Anweisungen ist aufgeteilt in drei Teile: in die DRIVE-Anweisungen, in die Report-Anweisungen für die Erstellung von Listen, Formularen und Berichten sowie in komplexe Unterstrukturen von Anweisungen, die als „Metavariablen“ beschrieben sind.
  Außerdem enthält das Lexikon eine Einführung in die Syntax der DRIVE-Anweisungen sowie eine Aufstellung aller Meldungen und Schlüsselwörter von DRIVE/WINDOWS.

Darüberhinaus gibt es noch Lexika mit DRIVE-SQL-Anweisungen für die verschiedenen Datenbanksysteme:

– „Lexikon der DRIVE-SQL-Anweisungen für SESAM V1“ [4]

– „Lexikon der DRIVE-SQL-Anweisungen für SESAM V2“ [5]

– „Lexikon der DRIVE-SQL-Anweisungen für UDS“ [6]

DRIVE/WINDOWS bietet zusätzlich die volle Funktionalität von DRIVE V5.1 im sogenannten Old-Style- und im Mischbetrieb.
Die Old-Style-Funktionen finden Sie in den Handbüchern DRIVE V5.1 Benutzerhandbuch [14] und Lexikon [15]. Wie Sie Old-Style-Programme in neue DRIVE-Anwendungen integrieren, ist im Handbuch „Programmiersprache“ [2] beschrieben, wie Sie DRIVE/WINDOWS für den Old-Style- und Mischbetrieb generieren im Handbuch „Programmiersystem“ [1].

1.4 Readme-Datei

Funktionelle Änderungen und Nachträge der aktuellen Produktversion zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei. Sie finden die Readme-Datei auf Ihrem BS2000-Rechner unter dem Dateinamen SYSRME.*produkt*.*version*.*sprache*. Die Benutzerkennung, unter der sich die Readme-Datei befindet, erfragen Sie bitte bei Ihrer zuständigen Systembetreuung. Die Readme-Datei können Sie mit dem Kommando /SHOW-FILE oder mit einem Editor ansehen oder auf einem Standarddrucker mit folgendem Kommando ausdrucken:

```
/PRINT-DOCUMENT dateiname, LINE-SPACING=*BY-EBCDIC-CONTROL
```

bei SPOOL -Versionen kleiner 3.0A:

```
/PRINT-FILE FILE-NAME=dateiname,LAYOUT-CONTROL=
    PARAMETERS(CONTROL-CHARACTERS=EBCDIC)
```

1.5 Änderungen gegenüber der Ausgabe vom Dezember 1993 (DRIVE/WINDOWS V1.1)

Komponenten

– DRIVE/WINDOWS unterstützt SESAM V2. Damit wird der Sprachstandard SQL2 erfüllt.

– SAM- und ISAM-Dateien lassen sich mit DRIVE-Programmen bearbeiten.

– Verteilte DRIVE-Anwendungen ermöglichen den Anschluß zu grafischen Bedienoberflächen auf den Clients (MS-Windows und SINIX).

– Old-Style-Programme lassen sich in neue DRIVE-Anwendungen integrieren durch den Programmaufruf mit CALL. Parameter können an das rufende Programm zurückgegeben werden.

– Im Old-Style ist neben dem Zugriff auf SESAM V1 nun auch der Zugriff auf SESAM V2, LEASY und DMS möglich.

– DRIVE-Programme mit SQL-Anweisungen für SESAM V2 können nur übersetzt werden, wenn keine Transaktion offen ist.

– Die Anschlüsse zu den Produkten TOM-REF und QUERY werden nicht mehr unterstützt.

Datentypen

– DRIVE/WINDOWS unterstützt die Datentypen TIME(3) und TIMESTAMP(3).

– Der Datentyp VARCHAR belegt ein Byte mehr als bisher. Redefinitionen auf VARCHAR-Variablen und Manipulationen des Längenfelds sind nicht mehr möglich.

Funktionen

– Die Anweisung HELP entfällt. Dafür wird das DRIVE-Lexikon als Datei (Softbook) ausgeliefert.

– Für Reports können Hintergrundmuster sowohl für Seiten als auch für einzelne Zeilen festgelegt werden.

– Die Aufrufreihenfolge bei CALL und DO hat sich geändert: Während DRIVE/WINDOWS bisher zunächst nach einem Zwischencode und dann nach einer Source suchte, wird nun immer das Element mit dem neuesten Datum aufgerufen.

– Mit der Anweisung COMPILE werden keine Fehlerlisten erzeugt. Fehlermeldungen werden in den Source-Teil der Übersetzungsliste eingestreut.

- Mit der Anweisung PARAMETER DYNAMIC LIBRARY wird nicht mehr automatisch eine PLAM-Bibliothek angelegt, sondern kann nur eine bereits vorhandene PLAM-Bibliothek zugewiesen werden.
- Benutzereigene Datentypen können als Übergabeparameter an gerufene Programme übergeben werden. Die Anweisung DECLARE TYPE darf vor der PROCEDURE-Anweisung stehen.
- Die Anweisungen COMMIT WORK und ROLLBACK WORK sind ohne die Anweisung OPTION DBSYSTEM= und mit der Anweisung OPTION DBSYSTEM=OFF erlaubt. Ist zum Zeitpunkt ihrer Ausführung keine Datenbanktransaktion offen, so beziehen sie sich nur auf UTM-Transaktionen.
- Mit der Übersetzungsoption OPTION SCREENCHECK legen Sie fest, ob DRIVE/WINDOWS CHECK-Klauseln in den Adressierungshilfen auswerten soll.
- Die DRIVE-Systemvariable &SQL_STATE enthält den SQLSTATE von SESAM V2.
- Die Anweisung PERMIT hat bei SESAM V2 keine Wirkung. Sie setzt nur den SQLSTATE.
- Mit CURRENT TIMESTAMP wird der aktuelle Zeitstempel ausgegeben.
- Sekundenbruchteile (FRACTION) können sowohl als Einheit für Zeitspannen als auch in Zeitangaben angegeben werden.
- Datumzeitausdrücke lassen sich zu einem Ergebnis mit dem Datentyp TIMESTAMP(3) verknüpfen.
- Die Funktion LENGTH liefert die letzte Position in einer Zeichenkette, die kein Leerzeichen ist.
- Mit den Funktionen UPPERSTRING und LOWERSTRING werden Zeichen gemäß der länderspezifischen Einstellung umgesetzt, mit TRSTRING gemäß der benutzereigenen Definition.
- Bei der Berechnung von Zeitspannen mit Uhrzeiten (TIME) rechnet DRIVE/WINDOWS nicht über die Tagesgrenzen, sondern es werden negative Sunden ausgewiesen.
- NULL-Werte werden am Bildschirm, wenn nichts anderes vereinbart ist, durch das Sonderzeichen @ dargestellt.
- Der Aufruf von PASCAL-Programmen wird nicht unterstützt.

Schlüsselwörter

– DRIVE/WINDOWS verwendet neue Schlüsselwörter. Eine Aufstellung aller Schlüsselwörter befindet sich im Anhang des DRIVE-Lexikons [3].

Kompatibilität

– DRIVE-Programme, deren Code-Elemente mit Vorgängerversionen von DRIVE/WINDOWS erstellt wurden, müssen erneut übersetzt werden, damit sie ablauffähig sind.

– Für DRIVE-Programme, die auf SESAM-Datenbanken zugreifen, sind die Migrationshinweise im Handbuch „Lexikon der DRIVE-SQL-Anweisungen für SESAM V2“ [5] zu beachten.

1.6 Darstellungsmittel

Die in den DRIVE/WINDOWS-Handbüchern verwendeten Zeichen und Schriftarten haben folgende Bedeutung:

`Schreibmaschinenschrift`

wird für feststehende Namen (z. B. Kommandos auf Betriebssytemebene, Dateinamen) und Fehlermeldungen im Fließtext verwendet. Außerdem wird sie in den Beispielen benutzt.

Kursive Schrift

kennzeichnet als Zwischenüberschrift Beispiele und im Fließtext frei wählbare Namen sowie Metavariablen.

Dieses Zeichen weist Sie auf eine sehr wichtige Information hin, die Sie unbedingt beachten müssen.

Die verwendete Metasprache wird im „DRIVE-Lexikon“ [3] beschrieben.

Bei Literaturverweisen, z.B. auf die oben erwähnten Handbücher, wird der Kurztitel zusammen mit einer Zahl in eckigen Klammern angegeben. In jedem Handbuch befindet sich im Anhang eine Literaturliste, die nach diesen Zahlen aufsteigend angelegt ist.

2 Aufbau von DRIVE-Programmen

Dieses Kapitel beschreibt die Struktur eines DRIVE-Programms (ab Seite 7) und den unterschiedlichen Einsatz eines DRIVE-Programms als Dialog-Programm oder UTM-Asynchronvorgang (ab Seite 15).

2.1 Programmstruktur

DRIVE-Programme bestehen aus vier Komponenten, deren Reihenfolge Sie unbedingt einhalten müssen:

– Startanweisung
– Deklarationsteil
– Verarbeitungsteil
– Endanweisung

Alle Anweisungen innerhalb eines Programms müssen mit „;“ (Strichpunkt) abgeschlossen werden.

2.1.1 Startanweisung

Die erste Anweisung eines Programms muß immer PROCEDURE sein, gefolgt vom Programmnamen.

Der Programmname *progname* muß nicht unbedingt mit dem Namen der Datei oder des Bibliothekelements übereinstimmen, unter dem das Quellprogramm abgespeichert ist.

```
PROCEDURE progname;
.
.
.
```

Handelt es sich bei dem Programm um ein Unterprogramm, an das beim späteren Aufruf mit DO, ENTER oder CALL Parameter übergeben werden, müssen diese Parameter in der USING-Klausel der PROCEDURE-Anweisung nach dem Programmnamen angegeben werden:

```
PROCEDURE progname USING parameter...;
```

Vor der Zeile mit dem Schlüsselwort PROCEDURE dürfen im Programm nur OPTION-Anweisungen angegeben werden, mit denen Sie die Übersetzung eines DRIVE-Programms steuern, und die Anweisung DECLARE TYPE.
OPTION-Anweisungen müssen vor PROCEDURE und eventuellen DECLARE TYPE-Anweisungen stehen. DECLARE TYPE-Anweisungen können direkt vor der Anweisung PROCEDURE stehen, aber auch nach PROCEDURE im Deklarationsteil (siehe „DRIVE-Lexikon“ [3], Anweisungen DECLARE TYPE und OPTION).

```
OPTION option-operand;
OPTION option-operand1;
...
OPTION option-operand5;
DECLARE TYPE usertyp;
...
DECLARE TYPE usertyp4;
PROCEDURE progname;
...
```

2.1.2 Deklarationsteil

Im Deklarationsteil stehen alle Definitionen von DRIVE-Objekten und die Definitionen aller internen Unterprogramme, die innerhalb des Programms aufgerufen werden. Am Ende des Deklarationsteils können Sie mit WHENEVER Fehlerausgänge definieren.

Stehen im Deklarationsteil SQL-Anweisungen, sollte aus Gründen der Datensicherheit nach dem Deklarationsteil die Anweisung COMMIT WORK stehen, also der Verarbeitungsteil mit COMMIT WORK begonnen werden.

Die folgende Tabelle zeigt alle Anweisungen, die im Deklarationsteil vorkommen können. Sie sind in vier Blöcken angeordnet, die die Reihenfolge festlegen, in der sie im Programm stehen müssen. Innerhalb des DECLARE- und CREATE-Blocks wird die Reihenfolge durch Bezüge bestimmt. Es kann z.B. sinnvoll sein, Reports am Ende des Blocks zu deklarieren, da evtl. auf vorher definierte Variablen zugegriffen wird.

Es folgen SUBPROCEDUREs in beliebiger Reihenfolge. WHENEVER muß am Ende des Deklarationsteils stehen.

Die einzelnen Blöcke sind durch verdoppelte Striche voneinander getrennt.

Anweisung	**Bedeutung**
CREATE TEMPORARY VIEW	Deklarieren eines temporären Views
DECLARE FORM	Deklarieren eines DRIVE-Bildschirmformats
DECLARE SCREEN	Deklarieren eines FHS-Formats
DECLARE LIST	Deklarieren einer DRIVE-Liste
DECLARE TYPE	Deklarieren eines Benutzerdatentyps
DECLARE CONSTANT	Deklarieren einer Konstanten
DECLARE VARIABLE	Deklarieren einer Variablen
DECLARE CURSOR	Deklarieren eines Cursors
DECLARE REPORT ... END REPORT	Deklarieren und definieren eines Reports
DECLARE FILE	Deklarieren einer Datei
SUBPROCEDURE ... END SUBPROCEDURE	Definition eines internen Unterprogramms
WHENEVER	Definieren von Fehlerausgängen

Die Reihenfolge der DECLARE- und CREATE-Anweisungen ist durch Bezüge bedingt. Bezieht sich eine Deklaration auf einen View, einen Cursor oder eine Variable, muß dieses Objekt vorher deklariert worden sein.

Beispiel

Ein Cursor, der sich auf eine Variable bezieht, muß nach der entsprechenden Variablen deklariert werden.
Eine Variable, die sich auf einen Cursor bezieht (LIKE...), muß nach diesem Cursor deklariert werden.

Die Anweisung DECLARE TYPE darf auch direkt vor PROCEDURE stehen.

Wird im Verarbeitungsteil mit CALL ein internes Unterprogramm aufgerufen, muß dieses Unterprogramm vorher im Deklarationsteil mit der Anweisung SUBPROCEDURE deklariert worden sein.

2.1.3 Verarbeitungsteil

Im Verarbeitungsteil stehen alle Anweisungen, die für die Ausführung des Programms benötigt werden. Es gibt drei Arten von Verarbeitungsblöcken:

– einfache Strukturblöcke

  Folgende Anweisungen können verwendet werden:

Anweisungen	**Bedeutung**
ALTER TABLE	Tabellenstruktur ändern
BREAK	Abbrechen einer Schleife, eines Programms oder eines Unterprogramms
CALL	Aufrufen von Unterprogrammen
CLEAR	Rücksetzen von Variablen und DRIVE-Formaten
CLOSE CURSOR	Schließen eines Cursors
CLOSE FILE	Schließen einer Datei
COMMIT WORK	Beenden einer Transaktion
COPY	Einfügen von COPY-Elementen
CREATE TABLE	Tabelle erzeugen
DELETE FILE RECORD	Löschen eines Satzes in einer ISAM-Datei
DELETE POSITIONED	Löschen eines Satzes
DELETE SEARCHED	Löschen von Sätzen
DISPLAY	Ausgeben von Formaten
DO	Starten eines Dialog-Programms
DROP VIEW	Freigeben eines Views
DROP TABLE	Tabelle löschen
ENTER	Starten eines UTM-Asynchronvorggangs
EXECUTE	Generieren und Ausführen von Anweisungen
FETCH	Positionieren des Cursors, Versorgen von Variablen
FILL	Aufbauen von Formaten
GET FILE POSITION	Lesen der Dateiposition
INSERT	Einfügen eines Satzes
LIST	Ausgeben von Druckaufträgen
LOCATE FILE	Positionieren in einer ISAM-Datei
OPEN	Öffnen eines Cursors
OPEN FILE	Öffnen einer Datei
PARAMETER	Festlegen von Programmablaufbedingungen

Anweisungen	**Bedeutung**
PERMIT	Angeben einer Benutzeridentifikation
READ FILE	Lesen einer Datei
REVOKE	Zugriffsrechte entziehen
ROLLBACK WORK	Rücksetzen einer Transaktion
SELECT	Auswählen eines Satzes
SEND MESSAGE	Ausgeben von Meldungen
SET	Zuweisen von Werten
SET FILE POSITION	Positionieren in einer Datei
SET TRANSACTION	Festlegen von Transaktionsbedingungen
STOP	Beenden des DRIVE-Laufs
STORE	Speichern einer Cursorposition
SYSTEM	Eingeben von BS2000-Kommandos
UNSAVE	Löschen von Programmen und COPY-Elementen
UPDATE POSITIONED	Ändern eines Satzes
UPDATE SEARCHED	Ändern von Sätzen
UPDATE STATISTICS	Statistik aktualisieren
WRITE FILE	Schreiben in eine Datei

– Schleifenstrukturblöcke

  Schleifenstrukturblöcke werden mit der Anweisung CYCLE eingeleitet und mit END CYCLE beendet.

– Auswahlstrukturblöcke

  Auswahlstrukturblöcke sind IF- oder CASE-Blöcke. Ein IF-Block wird mit der Anweisung IF eingeleitet und mit END IF beendet, ein CASE-Block wird mit CASE ... OF eingeleitet und mit END CASE beendet.

2.1.4 Endanweisung und Kommentare

Endanweisung Die letzte Anweisung eines Programms muß immer END PROCEDURE sein.

Kommentare Sie können Kommentare hinter Anweisungen stellen oder eigene Kommentarzeilen an beliebiger Stelle im Programm eingeben. Kommentare müssen mit `/*` eingeleitet und mit `*/` beendet werden.

2.1.5 COPY-Elemente

Es ist sinnvoll, für häufig wiederkehrende Anweisungsfolgen COPY-Elemente anzulegen.

COPY-Elemente enthalten eine Folge oder Teile von Anweisungen. Sie müssen als Elemente vom Typ S in einer DRIVE-Bibliothek gespeichert sein. Innerhalb eines Programms können sie sowohl im Deklarations- als auch im Verarbeitungsteil stehen. COPY-Elemente dürfen nicht geschachtelt werden.

Im Programm-Modus wird das COPY-Element bei der Übersetzung in das Quellprogramm an die Stelle kopiert, wo die COPY-Anweisung steht. In der Übersetzungsliste sind somit alle Anweisungen eines COPY-Elements angegeben.

Im Dialog-Modus wird nur die erste Anweisung des COPY-Elements am Bildschirm ausgegeben (siehe „DRIVE-Programmiersystem“ [1], Kapitel Expertenmodus).

2.1.6 Programmstruktur im Überblick

Die Programmstruktur im Überblick

[OPTION = ...]	**Compileranweisung**
[DECLARE TYPE]	**TYPE-Deklarationen**
PROCEDURE progname [USING]	**Startanweisung**
CREATE TEMPORARY VIEW ... DECLARE TYPE DECLARE ... CURSOR DECLARE SCREEN ... DECLARE CONSTANT ... DECLARE VARIABLE ... DECLARE FORM ... DECLARE REPORT ... DECLARE LIST ... DECLARE FILE ... SUBPROCEDURE subprogname WHENEVER ...	**Deklarationsteil**

COMMIT WORK OPEN FILE SET TRANSACTION COPY CASE ...OF READ FILE IF ...THEN DISPLAY ... INSERT INTO ... WRITE FILE CALL subprogname CLOSE FILE ...	**Verarbeitungsteil**
END PROCEDURE	**Endanweisung**

Beispiel

```
/*************************************************************************/
/* DRIVE/WINDOWS V2.1                                                    */
/* PROGRAMM MITARBEITER.LIST                                             */
/* ERSTELLUNGSDATUM 01.10.95                                             */
/* SNI STM QM 233                                                        */
/* SIEMENS NIXDORF INFORMATIONSSYSTEME AG                                */
/*************************************************************************/
/*                                                                       */
/* STARTANWEISUNG                                                        */
/*                                                                       */
/*************************************************************************/
OPTION DBSYSTEM=SESAM;
PROCEDURE MITLIST;
/*************************************************************************/
/*                                                                       */
/* DEKLARATIONSTEIL                                                      */
/*                                                                       */
/*************************************************************************/
DECLARE DRUCKCURSOR CURSOR FOR S ABT_MIT_NR,LFD_NR,NACHNAME,VORNAME,
                    GEHALT,ABT_LEITER,PROJ_MIT FROM SESAMDBMIT;
DECLARE VARIABLE &DRUCKSATZ LIKE CURSOR DRUCKCURSOR,
                 &FRAGE CHAR(1),
                 &DATUM DATE,
                 &ZEIT TIME;
```

```
DECLARE LIST MITARBEITERAUSGABE            /***** LISTENDEFINITION *****/
    TTITLE &DATUM,TAB 37,&ZEIT,TAB 110,'SEITE:',&PAGES,NL 3,
           TAB 60,'MITARBEITERLISTE',NL 1,
           TAB 59,'-'(18),NL 2
    BTITLE NL 2,'='(123);
/*************************************************************************/
/*                                                                       */
/* VERARBEITUNGSTEIL                                                     */
/*                                                                       */
/*************************************************************************/
  FILL MITARBEITERAUSGABE TABLE NAMES &DRUCKSATZ;
  CYCLE DRUCKCURSOR INTO &DRUCKSATZ.*;
    FILL MITARBEITERAUSGABE TABLE VALUES &DRUCKSATZ;
  END CYCLE;


   DISPLAY MITARBEITERAUSGABE; /***** AUSGABE DER LISTE nach SYSLST *****/
  DISPLAY FORM                /***** KOMPAKT-FORMAT                *****/
    NL 15,TAB 5,'SOLL JETZT GEDRUCKT WERDEN ? (J/N) : ',RETURN &FRAGE;
  IF &FRAGE = 'J' THEN
    LIST *;                   /***** DRUCKANSTOSS SOFORT           *****/

     SEND MESSAGE 'DIE MITARBEITERLISTE WIRD AUSGEDRUCKT. DUE-TASTE';
    SET &FRAGE = 'N';
  ELSE
    SEND MESSAGE
   'DIE MITARBEITERLISTE WIRD AM ENDE DER BS2000-SITZUNG GEDRUCKT.';
  END IF;
COMMIT WORK;
/*************************************************************************/
/*                                                                       */
/* ENDANWEISUNG                                                          */
/*                                                                       */
/*************************************************************************/
END PROCEDURE;
```

2.2 Einsatzarten von DRIVE-Programmen

Ein DRIVE-Programm kann als Dialog-Programm oder als UTM-Asynchronvorgang eingesetzt werden. Für den jeweiligen Einsatz müssen Besonderheiten beachtet werden, die in den folgenden Absätzen beschrieben sind.

2.2.1 Dialog-Programme

Ein DRIVE-Programm starten Sie mit der Anweisung DO als Dialog-Programm. Dies ist sowohl im TIAM- als auch im UTM-Betrieb erlaubt.

Während des Programmablaufs wird ein Dialog mit dem Programmierer oder Anwender geführt oder die Anweisungen werden aus einer Datei gelesen.

Ein Dialog-Programm kann im TIAM-Betrieb auch innerhalb einer Batch-Prozedur gestartet werden. In der Prozedur muß dann die Anweisung DO stehen (siehe“ DRIVE-Programmiersystem“ [1], Kapitel Expertenmodus).

Tritt während des Ablaufs ein Fehler auf, wird das Dialog-Programm abgebrochen, eine Fehlerliste erzeugt und nach SYSLST geschrieben. Zusätzlich wird eine Fehlermeldung am Bildschirm ausgegeben. Ausnahme: Tritt ein Ablauffehler auf, für den über WHENEVER ein Fehlerausgang definiert wurde, wird das Programm mit der bei WHENEVER vereinbarten Fehlerbehandlung fortgesetzt. Steht das Dialog-Programm während des Ablaufs in der EDT-Arbeitsdatei 0, wird keine Fehlerliste erzeugt, sondern die Fehlermeldungen werden bei der nächsten EDT-Anweisung in das in der EDT-Arbeitsdatei 0 stehende Programm eingefügt.

2.2.2 UTM-Asynchronvorgänge

Ein DRIVE-Programm starten Sie mit der Anweisung ENTER als UTM-Asynchronvorgang in einer lokalen oder entfernten UTM-Anwendung. Der UTM-Asynchronvorgang ist ein eigenständiger Vorgang, d.h. er läuft im Hintergrund ab. Während des Ablaufs können Sie daher weitere Anweisungen an ihrem Bildschirm eingeben.

Die Anzahl der UTM-Asynchronvorgänge eines Benutzers wird durch die UTM-Generierung begrenzt, d.h. es muß mindestens eine Asynchron-Task für die UTM-Anwendung definiert worden sein.

Für DRIVE-Programme, die als UTM-Asynchronvorgänge gestartet werden sollen, gelten folgende Regeln:

– Nicht erlaubt sind Anweisungen, die eine Bildschirmeingabe/-ausgabe erzeugen. Sie führen zu einem Programmabbruch.

– Es ist nur eine Transaktion erlaubt. Bei transaktionsgesicherten Programmen bricht DRIVE/WINDOWS das Programm nach dem ersten COMMIT WORK ab.

– DO-Anweisungen sind innerhalb einer offenen Transaktion nicht erlaubt. Als Ersatz sind in diesem Fall ENTER-Anweisungen gestattet.

– Die Anweisungen END PROCEDURE, BREAK PROCEDURE oder STOP beenden den UTM-Asynchronvorgang. Sollte zu diesem Zeitpunkt noch eine Transaktion offen sein, führt das zum Fehlerabbruch der Asynchron-Verarbeitung.

– BREAK führt zum Fehlerabbruch des UTM-Asynchronvorgangs.

Tritt während des Programmablaufs ein Fehler auf, wird der UTM-Asynchronvorgang abgebrochen und eine Fehlerliste in der zentralen Druckdatei erzeugt. Zusätzlich wird am Bildschirm eine Fehlermeldung ausgegeben, wenn der UTM-Asynchronvorgang in der lokalen Anwendung läuft.

3 Variablen und Konstanten einsetzen

Dieses Kapitel beschreibt:

- alle Arten von Variablen, die Sie in DRIVE/WINDOWS verwenden können, inklusive der DRIVE-Systemvariablen (ab Seite 18)
- die Datentypen der Variablen (ab Seite 29)
- wie Datentypen aus Datenbanksystemen in DRIVE/WINDOWS abgebildet werden (ab Seite 34)
- wie Sie Variablen definieren und qualifizieren (ab Seite 40)
- wie Sie Konstanten definieren und qualifizieren (ab Seite 64)
- wie Sie Variablen eindeutig zuweisen mit SET (ab Seite 65)

  hier finden Sie auch Informationen zu:

  Datums- und Zeitangaben
  Abkürzung „.*“ für eine Liste von Komponenten
- welche Datentypen Sie umwandeln können (ab Seite 75)

3.1 Variablenarten

Die Programmiersprache DRIVE/WINDOWS unterscheidet zwischen Variablen, die Sie als Programmierer selbst definieren müssen, und Systemvariablen, die bei jeder DRIVE-Sitzung dem Programm automatisch zur Verfügung stehen.

Weiterhin gibt es in DRIVE/WINDOWS die „Indikatorvariablen". Diese Variablen werden bei Parameterübergaben in andere Programmiersprachen benötigt. Ihr Wert gibt an, ob der Parameter den NULL-Wert oder einen definierten Wert enthält (siehe Abschnitt „Externe fremdsprachige Unterprogramme" auf Seite 111)

3.1.1 Variablen

Variablen definiert der Programmierer im Deklarationsteil eines Programms. Es gibt folgende Arten von Variablen:

– Einfache Variable

– Eindimensionale Variable oder Vektor

– Zweidimensionale Variable oder Matrix

– Datengruppe

– Wiederholungsgruppe

Beispiele

Einfache Variable	`&PREIS NUM(7,2)` 7 Stellen lang, davon 2 Nachkommastellen
Vektor	`&FREMDSPRACHE (3) CHAR(10)` mit Wiederholungsfaktor
Matrix	`&FILIALE_MONATSUMSATZ (3,12) NUM(7,2)` Die Wiederholungsfaktoren 3 und 12 ergeben 3 * 12 Ausprägungen

Datengruppe	`1 &ADRESSE`
	`  2 STRASSE,`
	`  3 STRASSENNAME CHAR(25)`
	`  3 HAUSNR NUM(3),`
	`  2 PLZ NUM(5),`
	`  2 ORT CHAR(30),`
	`  2 TELEFONNR (5) NUM(12);`

Wiederholungs-gruppe	`1 &ADRESSE (3),`
	`  2 STRASSE,`
	`  3 STRASSENNAME CHAR(25)`
	`  3 HAUSNR NUM(3),`
	`  2 PLZ NUM(5),`
	`  2 ORT CHAR(30),`
	`  2 TELEFONNR (5) NUM(12);`

Folgende Begriffe sind wichtig:

Komponenten sind die Teile, aus denen eine Variable aufgebaut ist. Eine Variable kann aus einer einzigen Komponente bestehen, wie die einfache Variable, oder aus mehreren Komponenten, wie die Datengruppe. Eine Komponente kann wiederum aus Komponenten aufgebaut sein, wie die Komponente STRASSE, die aus STRASSENNAME und HAUSNR besteht.

Einfache Komponenten sind z.B. STRASSENNAME oder ORT. Einfache Komponenten sind Komponenten, die mit einem Datentyp definiert und nicht weiter in untergeordnete Komponenten strukturiert sind.

Ein **Vektor** oder eine **Matrix** besteht aus Komponenten des gleichen Datentyps (ausführliche Beschreibung hierzu siehe Abschnitt „Strukturierte Datentypen“ auf Seite 37).

Eine **Datengruppe** besteht aus Komponenten beliebiger Datentypen. Sie kann mehrere gleichartige Unterstrukturen enthalten. Diese Form der Datengruppe wird durch eine Wiederholungsgruppe dargestellt, wobei die Ausprägung der Anzahl der (gleichen) Unterstrukturen entspricht. Eine **Wiederholungsgruppe** ist also eine Datengruppe mit mehrfacher Ausprägung (ausführliche Beschreibung hierzu siehe Abschnitt „Strukturierte Datentypen“ auf Seite 37).

Vektor, Matrix, Datengruppe und Wiederholungsgruppe sind **strukturierte Variablen**. Datengruppe und Wiederholungsgruppe werden in den DRIVE-Handbüchern auch unter dem Begriff **Gruppe** zusammengefaßt.

In den DRIVE-Handbüchern wird auch von **Feldern** gesprochen. Ein Feld kann eine einfache Variable sein oder eine Komponente einer Variablen.

Der Begriff **Ausprägung** ist hier mit Hilfe der Wiederholungsgruppe &ADRESSE (3) aus obigem Beispiel erläutert. Die Wiederholungsgruppe &ADRESSE hat drei Ausprägungen: &ADRESSE (1), &ADRESSE (2) und &ADRESSE (3). Die Komponente STRASSENNAME kommt in jeder der drei Ausprägungen von &ADRESSE einmal vor, hat also drei Ausprägungen; die Komponente TELEFONNR kommt jeweils fünfmal vor, hat also insgesamt 3 x 5 = 15 Ausprägungen.

3.1.2 Systemvariablen

Für jedes Programm stehen DRIVE-Systemvariablen zur Verfügung. Sie können diese Variablen benutzen, ohne sie selbst zu definieren. Der Gültigkeitsbereich der Systemvariablen ist auf das jeweilige Programm begrenzt. Sie können über Systemvariablen keine Informationen an andere externe DRIVE-Programme geben, ausgenommen mit Hilfe der USING-Klausel.

Folgende Systemvariablen werden für DRIVE-Programme automatisch angelegt:

Bedeutung	**Bezeichnung**	**Definition der Systemvariablen**
User-Identification	&USER	PERMANENT CHAR (8) INIT 'user' [1)]
logische Seitenanzahl	&PAGES	PERMANENT INTEGER INIT 0
K/F-Taste	&KFKEY	PERMANENT CHAR (3) INIT '␣'

Bedeutung	Bezeichnung	Definition der Systemvariablen
DRIVE-Fehlerhinweise	&ERROR_STATE Die Komponenten von &ERROR_STATE sind: &ERROR &DML_STATE &SQL_CODE &SQL_STATE mit Komponenten: &SQL_CLASS &SUB_CLASS &FORMAT_NAME &VAR_NAME &LINE_NR &DISPLACEMENT &STATEMENT &SUB_CODE	PERMANENT CHAR (16) INIT 'OK' CHAR (16) INIT 'OK' INTEGER INIT 0 CHAR (2) INIT '00' CHAR (3) INIT '000' CHAR (31) INIT '␣' CHAR (31) INIT '␣' INTEGER INIT 0 INTEGER INIT 0 CHAR (16) INIT '␣' CHAR (8) INIT '␣'
Return-Information bei Meldungen	&CONTROL_STATE	PERMANENT CHAR (8) INIT '␣'
Zählfeld für die &ERROR-Fehler	&ACCESS_ERROR &CALC_OVERFLOW &CHECK_ERROR &CONVERSION_ERROR &DIVISION_ERROR &FORMAT_ERROR &INDEX_ERROR &SYNTAX_ERROR &SYSTEM_ERROR &WINDOW_ERROR	PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0
Zählfeld für die &DML_STATE-Fehler	&ACC_SYS_ERROR &ADMIN_SYS_ERROR &CURSOR_SQL_ERROR &DIRTY_READ &LIMIT_REACHED &SQL_ERROR &TABLE_END &TEMP_SYS_ERROR	PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0 PERMANENT INTEGER INIT 0

[1] Bei der Initialisierung wird unterschieden zwischen Nicht-UTM- und UTM-Betrieb.
Im Nicht-UTM-Betrieb wird &USER initialisiert mit dem Wert, der bei PARAMETER STATIC USER angegeben wurde oder mit TSN=nnnn, wenn diese Angabe fehlt.
Im UTM-Betrieb wird &USER mit dem Namen initialisiert, der bei KDCSIGN angegeben wurde.

Systemvariablen qualifizieren

Wenn Sie einer Systemvariablen einen Wert zuweisen wollen oder sie nach einem Wert abfragen, müssen Sie sie ansprechen, d.h. qualifizieren, können. Die Qualifizierung der Systemvariablen verläuft nach den gleichen Regeln wie die Qualifizierung der anderen Variablen (siehe Abschnitt „Strukturierte Variablen qualifizieren“ auf Seite 44). Komponenten von strukturierten Systemvariablen können also direkt angesprochen werden, wenn Eindeutigkeit vorliegt. Zum Beispiel ist die Angabe &ERROR_STATE.ERROR identisch mit der Angabe &ERROR. Die kürzere Bezeichnung &ERROR ist eindeutig, ebenso wie z.B. &DML_STATE.

Ausgeben von Systemvariablen

Sie können die Inhalte der Systemvariablen mit der DISPLAY FORM-Anweisung ausgeben.

Beispiel

```
DISPLAY FORM LINE VALUES &ERROR_STATE;
```

Bedeutung der Systemvariablen

&USER

In der Systemvariable &USER steht die *user*-Angabe aus der PARAMETER-Anweisung oder die TSN-Nummer.

&PAGES

In der Systemvariable &PAGES steht die Anzahl der Seiten, die seit Beginn des Programms bzw. dem letzten DISPLAY LIST, d.h. seit dem letzten Löschen von &PAGES ausgegeben wurden. Jeder DISPLAY LIST erhöht den Wert um 1. Dies gilt auch bei implizitem DISPLAY LIST, d.h. bei Überlauf durch FILL LIST.

&KFKEY

K- bzw. F-Taste, die zuletzt betätigt wurde.

&KFKEY enthält jedoch nur dann eine Tastenbezeichnung, wenn für diese Taste in der PARAMETER-Anweisung KFKEY = 'taste' angegeben war. Andernfalls wird diese Systemvariable, auch wenn die K- bzw. F-Taste gedrückt wurde, mit Leerzeichen versorgt.

&ERROR_STATE

Die Systemvariable &ERROR_STATE ist eine strukturierte Variable mit Komponenten. Abhängig von Fehlerklassen versorgt DRIVE/WINDOWS diese Komponenten mit Einträgen. Die Komponenten können als Systemvariablen mit folgenden Namen angesprochen werden:

- &ERROR
- &DML_STATE
- &SQL_CODE
- &SQL_STATE
- &SQL_CLASS
- &SUB_CLASS
- &FORMAT_NAME
- &VAR_NAME
- &LINE_NR
- &DISPLACEMENT
- &STATEMENT
- &SUB_CODE

&ERROR

In der Systemvariablen &ERROR stehen DRIVE-Fehlerhinweise.

Läuft ein Programm ohne Fehler erhält &ERROR den Eintrag 'OK'. Den Eintrag 'OK' erhält &ERROR auch, wenn bei SQL-Anweisungen mit INTO-Klausel die Systemvariable &DML_STATE einen Eintrag erhält. Bei allen anderen SQL-Anweisungen bleibt &ERROR unverändert.

Läuft ein Programm nicht fehlerfrei, erhält &ERROR einen der folgenden Einträge:

Eintrag	**Bedeutung**
DIVISION ERROR	Divisionsfehler
CONVERSION ERROR	ein Feld wurde nicht typgerecht versorgt
INDEX ERROR	Index nicht im zulässigen Bereich
CHECK ERROR	CHECK-Klausel verletzt
CALC OVERFLOW	Wertüberlauf. Diese Fehlermeldung wird um mehrere Zusätze erweitert: neben 'machine error' wird z.B. auch ausgegeben 'Fehler im x.ten Argument'.
FORMAT ERROR	Felder eines FHS-Formats unkorrekt versorgt: : mindestens ein Feldattribut mit EDIT_STATE = 'I' oder 'M' oder unzulässige K/F-Taste betätigt
SYNTAX ERROR	Fehler während der Übersetzung bei EXECUTE
DIS ERROR	Fehler bei Verteilter Verarbeitung
SYSTEM ERROR	Fehler bei einer SYSTEM-Anweisung: Der vom Betriebssystem gelieferte Returncode wird abdruckbar in &SUB_CODE abgelegt
ACCESS ERROR	Zugriffsfehler bei UNSAVE
FILE ERROR	Fehler bei der Dateibearbeitung

Zusätzlich wird &ERROR noch in folgenden Fällen versorgt, die nicht als Fehler gewertet werden und nicht zum Programmabbruch führen:

OK END	Eintrag nach der Anweisung READ FILE, wenn das Dateiende erreicht ist.
TOO SHORT	Fehler nach der Anweisung READ FILE, wenn der gelesene Datensatz kürzer ist als die DRIVE-Variable(n) der INTO-Klausel.
TOO LONG	Fehler nach der Anweisung READ FILE, wenn der gelesene Datensatz länger ist als die DRIVE-Variable(n) der INTO-Klausel. Fehler nach der Anweisung WRITE FILE, wenn der zu schreibende Datensatz länger ist als die erlaubte Satzlänge im BS000.

Wie Sie Ablauffehler in Programmen mit der Anweisung WHENEVER abfangen, finden Sie im Abschnitt „Fehler abfangen, Endekriterien“ auf Seite 88.

Bei jedem Auftreten eines &ERROR-Fehlers wird die entsprechende Systemvariable hochgezählt. Liegt z.B. ein Fehler bei einer Division vor, so erhält die Systemvariable &ERROR den Eintrag 'DIVISION ERROR'. Die gleichnamige Systemvariable &DIVISION_ERROR, die als Zählfeld für derartige Fehler dient, wird um 1 hochgezählt.

```
DIVISION ERROR          →  Inhalt von &ERROR: Fehlerhinweis
wert + 1                →  Inhalt von &DIVISION_ERROR: Anzahl der Fehler
```

&DML_STATE

In der Systemvariablen &DML_STATE stehen DRIVE-Fehlerhinweise der letzten DML-Operation.

Läuft ein Programm ohne Fehler erhält &DML_STATE bei SQL-Anweisungen den Eintrag 'OK'. Den Eintrag 'OK' erhält &DML_STATE auch, wenn die Systemvariable &ERROR einen Eintrag erhält.

Läuft ein Programm nicht fehlerfrei, erhält &DML_STATE in Abhängigkeitvom SQL-Code einen der folgenden Einträge:

Eintrag	**Bedeutung**
SQL ERROR	fehlerhafte SQL-Anweisung
CURSOR SQL ERROR	Positionsfehler beim Cursor
SYS ERROR	systembedingte Fehler
TEMP SYS ERROR	kurzfristiges Hindernis
ADMIN SYS ERROR	Administratoreingriff notwendig
ACC SYS ERROR	Unverträglichkeiten im SQL-Schema
LIMIT REACHED	Betriebsmittelengpaß beim DB-System
TOO MANY CURSORS	Mehr als 20 dynamische und variable Cursor definiert

Die Einträge 'SQL ERROR' und 'CURSOR SQL ERROR' fassen SQL-Codes zusammen, die Fehler identifizieren, die sich aus der Sprachdefinition von SQL ergeben (-700 < &SQL_CODE < 0).
Die Einträge 'TEMP SYS ERROR', 'ADMIN SYS ERROR' und 'ACC SYS ERROR' fassen SQL_CODEs zusammen, die auf systembedingte Fehler hinweisen (-990 ≤ &SQL_CODE ≤ -700).
Für den Eintrag 'LIMIT REACHED' gibt es keinen SQL-Code.

Zusätzlich wird &DML_STATE noch in zwei Fällen versorgt, die nicht als Fehler gewertet werden und nicht zum Programmabbruch führen:

Eintrag	**Bedeutung**
TABLE END	kein weiterer Treffer
DIRTY READ	von fremder Transaktion gehaltener Satz gelesen (nur bei Konsistenzlevel 0 und 1)

So können Sie beim Einlesen von Datenbanksätzen abfragen, ob alle Sätze eingelesen worden sind oder keiner vorhanden ist:

Beispiel

```
...
FETCH C_PROJEKT INTO &STAMMSATZ_PROJEKT.*;
IF &DML_STATE = 'TABLE END'
   THEN CALL ende;
END IF;
...
```

Wie Sie Ablauffehler in Programmen mit der Anweisung WHENEVER abfangen, finden Sie im Abschnitt „Fehler abfangen, Endekriterien“ auf Seite 88.

&SQL_CODE

Das Feld enthält standardmäßig den Wert 0 . Tritt ein Fehler der Fehlerklasse DML_STATE auf, wird der Fehlercode 'SQLCODE' des jeweiligen Datenbanksystems (SESAM/SQL, UDS/SQL) eingetragen.

&SQL_STATE

In der Systemvariablen &SQL_STATE steht der Fehlercode 'SQLSTATE' von SESAM/SQL V2.

&SQL_CLASS

In der Systemvariablen &SQL_CLASS steht der Klassenwert für den Fehlercode 'SQL-STATE' von SESAM V2.

&SUB_CLASS

In der Systemvariablen &SUB_CLASS steht der Unterklassenwert für den Fehlercode 'SQLSTATE' von SESAM V2.

&FORMAT_NAME

Das Feld enthält standardmäßig Leerzeichen. Tritt bei der Abarbeitung eines FHS-Formats ein Fehler auf, d.h.enthält nach DISPLAY *screenformat* das Feld EDIT_STATE für ein Teilformat einen Eintrag ≠ 'V' (VALID), wird der Name des ersten fehlerhaften Teilformats eingetragen.

&VAR_NAME

Das Feld enthält standardmäßig Leerzeichen. Erhält bei einer Variablenbearbeitung eines der Felder &ERROR oder &DML_STATE einen Eintrag ≠ 'OK', wird der Name dieser Variablen eingetragen.
Hat nach einer Anweisung DISPLAY *screenformat* ein Teilformat einen EDITZUSTAND ≠ 'V' (VALID), wird der Name des ersten fehlerhaften Datenfeldes eingetragen.

&LINE_NR

Das Feld enthält standardmäßig den Wert '0'. Erhält eines der Felder &ERROR oder &DML_STATE einen Eintrag ≠ 'OK', wird die Zeilennummer (bezogen auf die Übersetzungsliste) der fehlerhaften Anweisung eingetragen.

&DISPLACEMENT

Das Feld enthält standardmäßig den Wert '0'. Erhält eines der Felder &ERROR oder &DML_STATE einen Eintrag ≠ 'OK', wird die Distanz vom Anfang der fehlerhaften Anweisung bis zur Fehlerstelle eingetragen (bezogen auf die Übersetzungsliste).

&STATEMENT

Das Feld enthält standardmäßig Leerzeichen. Erhält eines der Felder &ERROR oder &DML_STATE einen Eintrag ≠ 'OK', wird das erste Schlüsselwort der fehlerhaften Anweisung eingetragen. Handelt es sich um eine zusammengesetzte Anweisung (z.B. COMMIT WORK WITH DISPLAY), wird die Teilanweisung eingetragen, die den Fehler verursacht hat.

&SUB_CODE

Das Feld enthält standardmäßig Leerzeichen. Erhält eines der Felder &ERROR oder &DML_STATE einen Eintrag ≠ 'OK', wird ein Systemfehlercode eingetragen.

Nach der Ausführung der Anweisung SYSTEM wird der Returncode des Betriebssystems eingetragen.

Nach Kommunikation mit dem Server wird im Fehlerfall die entsprechende Fehlerklasse eingetragen, wenn &ERROR = 'DIS ERROR' oder &ERROR = 'REMOTE ERROR'.

Bei jedem Auftreten eines &DML_STATE-Fehlers wird die entsprechende Zählvariable hochgezählt. Liegt z.B. ein Fehler bei einem Datenbank-Zugriff vor, so erhält die Systemvariable &DML_STATE einen Wert, z.B. 'SQL ERROR'. Die gleichnamige Systemvariable &SQL_ERROR, die als Zählfeld für derartige Fehler dient, wird um 1 hochgezählt.

`SQL ERROR`	→	Inhalt von &DML_STATE: Fehlerhinweis
`wert + 1`	→	Inhalt von &SQL_ERROR: Anzahl der Fehler

&CONTROL_STATE

In Grafik-Anwendungen wird nach der Anweisung SEND MESSAGE der im MESSAGE-Fenster gedrücke Knopf (OK, CANCEL oder CLOSE) in der Systemvariable &CONTROL_STATE hinterlegt.

&ACCESS_ERROR bis &SYSTEM_ERROR

Die Systemvariablen &ACCESS_ERROR bis &WINDOW_ERROR sind Zählfelder. In ihnen wird das Auftreten entsprechender Fehlerausgänge oder-hinweise von &ERROR gezählt.

&ACC_SYS_ERROR bis &TEMP_SYS_ERROR

In den Systemvariablen &ACC_SYS_ERROR bis &TEMP_SYS_ERROR wird das Auftreten entsprechender Fehlerausgänge von &DML_STATE gezählt. Erreicht eine dieser Variablen den Maximalwert, so wird sie nicht weiter hochgezählt. Der Maximalwert ist ein Integerwert von 2^{31}-1 (siehe Abschnitt „Grunddatentypen“ auf Seite 29).

Beispiel für die Belegung der Systemvariablen &ERROR und &DML_STATE

Anweisung	**Ereignis**	**&ERROR=**	**&DML_STATE=**
SET &v=&a(&i)	INDEX ERROR	INDEX ERROR	unverändert
OPEN *cursorname*	SQL ERROR	unverändert	SQL ERROR
FETCH *cursorname* INTO...	DIRTY READ	OK	DIRTY READ

Zum Abfangen von Ablauffehlern siehe Abschnitt „Fehler abfangen, Endekriterien“ auf Seite 88 und „DRIVE-Lexikon“ [3], Anweisung WHENEVER.

3.2 Datentypen

Dieser Abschnitt beschreibt die Datentypen von DRIVE-Variablen. Den Datentyp müssen Sie bei der Definition einer Variablen festlegen.

Datentypen lassen sich einteilen:

– die Grunddatentypen

  Einfache Variablen und Komponenten einer Variable besitzen einen der alphanumerischen, numerischen oder Zeit-Datentypen oder den Datentyp INTERVAL (siehe DRIVE-Lexikon [3], Metavariable *grunddatentyp*).

– die benutzerdefinierten Datentypen

  Mit der Anweisung DECLARE TYPE können Datentypen definiert werden, die dann einer einfachen Variable oder der Komponente einer Variable zugeordnet werden (siehe DRIVE-Lexikon [3], Metavariable *grunddatentyp*).

– die strukturierten Datentypen

  Strukturierte Variablen, die aus Komponenten mit beliebigen Grunddatentypen bestehen wie Vektoren, Matrizen, Datengruppen und Wiederholungsgruppen, sind Varianten des strukturierten Datentyps. Sie lernen im folgenden den Aufbau der Varianten des strukturierten Datentyps kennen (siehe Abschnitt „Strukturierte Datentypen“ auf Seite 37). Diese Information benötigen Sie, wenn Sie z.B. die gesamte Struktur in Bedingungen angeben.

3.2.1 Grunddatentypen

DRIVE/WINDOWS unterstützt alphanumerische, numerische und Zeit-Datentypen sowie den Datentyp INTERVAL.

3.2.1.1 Alphanumerische Datentypen

Datentyp	**Definition**
CHARACTER [(*l*)]	*l*: Anzahl der Zeichen $0 < l \leq 32000$ Vorbelegung: $l = 1$
VARCHAR (*l*) CHARACTER VARYING (*l*)	*l*: Anzahl der Zeichen $0 < l \leq 32000$

Eine Variable vom Datentyp CHARACTER ist mit „␣“ (Leerzeichen) vorbelegt. Eine Variable vom Datentyp VARCHAR ist mit einer leeren Zeichenkette vorbelegt.

Zum Ausgeben überlanger CHARACTER-Variablen (max. ein Bildschirm, Fortsetzungszeichen „>>“) siehe „DRIVE-Lexikon“ [3], Anweisung DISPLAY FORM.

3.2.1.2 Numerische Datentypen

Datentyp	Definition
NUMERIC [(*m*[,*n*])]	*m*: Anzahl der Gesamtstellen *n*: Anzahl der Nachkommastellen $0 < m \leq 15$ $0 \leq n \leq m$ Ist $m = 15$, muß $n < m$ sein. Vorbelegung: $m = 8$, $n = 0$
DECIMAL [(*m*[,*n*])]	*m*: Anzahl der Gesamtstellen *n*: Anzahl der Nachkommastellen $0 < m \leq 15$ $0 \leq n \leq m$ Ist $m = 15$, muß $n < m$ sein. Vorbelegung: $m = 15$, $n = 0$
EXTENDED DECIMAL [(*m*[,*n*])] XDEC [(*m*[,*n*])]	*m*: Anzahl der Gesamtstellen *n*: Anzahl der Nachkommastellen $0 < m \leq 32$ $0 \leq n \leq m$ Ist $m = 32$, muß $n < m$ sein. Vorbelegung: $m = 32$, $n = 0$
SMALLINT	2-Byte-Ganzzahl Wertebereich: -32768 bis 32767
INTEGER	4-Byte-Ganzzahl Wertebereich: -2147483648 bis 2147483647
REAL	Gleitpunktzahl mit der Genauigkeit von 6 Stellen
DOUBLE PRECISION FLOAT	Gleitpunktzahl mit der Genauigkeit von 15 Stellen

Eine Variable mit numerischem Datentyp ist mit dem Wert '0' vorbelegt.

Wenn die Rechengenauigkeit einer Operation im Vordergrund steht, wählen Sie nicht den Datentyp REAL. Beim Zuweisen und Ausgeben eines Datenwerts kann es zu Ungenauigkeiten kommen.

Der Datentyp XDEC als Ergebnis eines numerischen Ausdrucks (siehe „DRIVE-Lexikon“ [3], Metavariable numausdruck) wird bei Parameterübergabe an Old-Style-Programme als Datentyp NUMERIC abgebildet.

3.2.1.3 Zeit-Datentypen

Datentyp	Definition
DATE	Datum: *jahr-monat-tag* (Darstellung: YYYY-MO-DD)
TIME	Uhrzeit: *stunde*:*minute*:*sekunde* (Darstellung: HH:MI:SS)
TIME(3)	Uhrzeit: *stunde*:*minute*:*sekunde.bruchteil* (Darstellung: HH:MI:SS.FFF)
TIMESTAMP(3)	Zeitstempel (*jahr-monat-tag stunde*:*minute*:*sekunde.bruchteil*) (Darstellung: YYYY-MO-DD␣HH:MI:SS.FFF)

Eine Variable vom Zeit-Datentyp ist, abhängig von der Variablendefinition (Anweisung DECLARE VARIABLE) folgendermaßen vorbelegt:

Angabe bei DECLARE VARIABLE	**DATE**	**TIME**	**TIME(3)**	**TIMESTAMP(3)**
PERMANENT	Datum des Rechners zum Übersetzungszeitpunkt	Uhrzeit des Rechners zum Übersetzungszeitpunkt	Uhrzeit des Rechners zum Übersetzungszeitpunkt	Zeitstempel des Rechners zum Übersetzungszeitpunkt
TEMPORARY	aktuelles Datum des Rechners	aktuelle Uhrzeit des Rechners	aktuelle Uhrzeit des Rechners	aktueller Zeitstempel des Rechners

Rechnen mit Zeitpunkten

Die Subtraktion von Zeitpunkten ergibt eine Zeitspanne.

Das Ergebnis ist eine Ganzzahl. Nachkommastellen werden abgeschnitten.

Abhängig vom Datentyp der Zeitpunkte (DATE, TIME, TIME(3) oder TIMESTAMP(3)) hat das Ergebnis folgenden Datentyp:

INTERVAL DAYS — wenn ein Zeitpunkt mit dem Datentyp DATE von einem Zeitpunkt mit dem Datentyp DATE oder TIMESTAMP(3) subtrahiert wird oder wenn ein Zeitpunkt mit dem Datentyp TIMESTAMP(3) von einem Zeitpunkt mit dem Datentyp DATE subtrahiert wird
(DATE - DATE, TIMESTAMP(3) - DATE oder DATE - TIMESTAMP(3)).

INTERVAL SECONDS	wenn ein Zeitpunkt mit dem Datentyp TIME von einem Zeitpunkt mit dem Datentyp TIME subtrahiert wird (TIME - TIME).
INTERVAL FRACTIONS	wenn ein Zeitpunkt mit dem Datentyp TIME von einem Zeitpunkt mit dem Datentyp TIME(3) oder TIMESTAMP(3) subtrahiert wird (TIME(3) - TIME oder TIMESTAMP(3) - TIME), wenn ein Zeitpunkt mit dem Datentyp TIME(3) von einem Zeitpunkt mit dem Datentyp TIME, TIME(3) oder TIMESTAMP(3) subtrahiert wird (TIME - TIME(3), TIME(3) - TIME(3) oder TIMESTAMP(3) - TIME(3)) oder wenn ein Zeitpunkt mit dem Datentyp TIMESTAMP(3) von einem Zeitpunkt mit dem Datentyp TIME, TIME(3) oder TIMESTAMP(3) subtrahiert wird (TIME - TIMESTAMP(3), TIME(3) - TIMESTAMP(3) oder TIMESTAMP(3) - TIMESTAMP(3)).

Zum Rechnen mit Zeitspannen siehe „Rechnen mit Zeitpunkten und -spannen“ auf Seite 33.

Für das Rechnen mit Zeitpunkten gelten folgende Regeln:

– Das Rechnen mit negativen Datumsangaben (vor Christus) ist nicht erlaubt.

– Das Rechnen mit Zeitpunkten ist nicht assoziativ.

Beispiel

(&datum1 - &datum2) - &datum3 ≠ &datum1 - (&datum2 + &datum3)
Der Rechenausdruck *&datum1 - &datum2 - &datum3* ist mehrdeutig.

3.2.1.4 Datentyp INTERVAL

Datentyp	Definition
INTERVAL *q*	Ganzzahl (max. 32 Stellen) *q* gibt die Intervalleinheit für eine Zeitspanne an. Es dürfen nur Intervalleinheiten mit einer Komponente angegeben werden: YEARS, MONTHS, DAYS, HOURS, MINUTES, SECONDS, FRACTIONS oder YEAR TO YEAR, MONTH TO MONTH, usw.

Eine Variable vom Datentyp INTERVAL ist mit dem Wert '0' vorbelegt.

Rechnen mit Zeitpunkten und -spannen

Die Addition oder Subtraktion einer Zeitspanne zu einem Zeitpunkt ergibt einen Zeitpunkt.

Wenn der Zeitpunkt ein Datum (DATE) ist, darf die Zeitspanne nur die Intervalleinheit YEARS, MONTHS oder DAYS haben.

Wenn der Zeitpunkt eine Uhrzeit (TIME oder TIME(3)) ist, darf die Zeitspanne nur die Intervalleinheit HOURS, MINUTES oder SECONDS haben. Bei TIME(3) ist auch die Intervalleinheit FRACTIONS erlaubt.

Abhängig vom Datentyp der Zeitpunkte (DATE, TIME, TIME(3) oder TIMESTAMP(3)) hat das Ergebnis folgenden Datentyp:

Datentyp	**Begründung**
DATE	wenn eine Zeitspanne mit dem Datentyp INTERVAL zu einem Zeitpunkt mit dem Datentyp DATE addiert oder von ihm subtrahiert wird (DATE + INTERVAL oder DATE - INTERVAL).
TIME	wenn eine Zeitspanne mit dem Datentyp INTERVAL zu einem Zeitpunkt mit dem Datentyp TIME addiert oder von ihm subtrahiert wird (TIME + INTERVAL oder TIME - INTERVAL).
TIME(3)	wenn eine Zeitspanne mit dem Datentyp INTERVAL zu einem Zeitpunkt mit dem Datentyp TIME(3) addiert oder von ihm subtrahiert wird (TIME(3) + INTERVAL oder TIME(3) - INTERVAL).
TIMESTAMP(3)	wenn eine Zeitspanne mit dem Datentyp INTERVAL zu einem Zeitpunkt mit dem Datentyp TIMESTAMP(3) addiert oder von ihm subtrahiert wird (TIMESTAMP(3) + INTERVAL oder TIMESTAMP(3) - INTERVAL).

Für das Rechnen mit Zeitpunkten und -spannen gelten folgende Regeln:

– Das Rechnen mit Zeitpunkten und Zeitspannen ist nicht kommutativ.

– Die Rechenausdrücke *&intervall + &datum* oder *&intervall - &datum* sind Fehler.

Rechnen mit Zeitspannen

Die Addition oder Subtraktion zweier Zeitspannen ergibt eine Zeitspanne.

Das Produkt aus einer Zeitspanne und einem numerischen Wert und die Division einer Zeitspanne durch einen numerischen Wert ergeben ebenfalls eine Zeitspanne. Das Ergebnis ist eine Ganzzahl, Nachkommastellen werden abgeschnitten.

Das Ergebnis hat folgenden Datentyp:

INTERVAL MONTHS	wenn Zeitspannen mit den Intervalleinheiten YEARS und/ oder MONTHS addiert oder subtrahiert werden oder wenn Zeitspannen mit der Intervalleinheit YEARS oder MONTHS mit einem numerischen Wert multipliziert oder durch einen numerischen Wert dividiert werden.
INTERVAL SECONDS	wenn Zeitspannen mit den Intervalleinheiten DAYS, HOURS, MINUTES und/oder SECONDS addiert oder subtrahiert werden oder wenn Zeitspannen mit der Intervalleinheit DAYS, HOURS, MINUTES oder SECONDS mit einem numerischen Wert multipliziert oder durch einen numerischen Wert dividiert werden.
INTERVAL FRACTIONS	wenn Zeitspannen mit den Intervalleinheiten DAYS, HOURS, MINUTES, SECONDS und/oder FRACTIONS addiert oder subtrahiert werden oder wenn Zeitspannen mit der Intervalleinheit FRACTIONS mit einem numerischen Wert multipliziert oder durch einen numerischen Wert dividiert werden.

Für das Rechnen mit Zeitspannen gilt: Sie dürfen nur mit Zeitspannen rechnen, wenn die Intervalleinheiten miteinander vergleichbar sind (siehe Abschnitt „Intervalleinheiten umrechnen“ auf Seite 80). 6 YEARS + 7 HOURS ist z.B. nicht erlaubt

Setzen Sie im Zweifelsfall Rechenausdrücke in Klammern und fügen Sie die Einheit hinzu (siehe „DRIVE-Lexikon“ [3], Metavariable *datumzeiteinheit*). Dies ist insbesonders bei Prädikaten sinnvoll.

3.2.1.5 Grunddatentypen aus Datenbanksystemen

Nicht alle Datentypen von DRIVE/WINDOWS und alle Datentypen der Datenbanksysteme SESAM V1, SESAM V2und UDS stimmen völlig überein. Dieser Abschnitt enthält die Unterschiede zwischen den Datentypen und wie Datentypen der Datenbanksysteme in DRIVE/WINDOWS abgebildet werden.

SESAM V1- und UDS-Datentypen in DRIVE/WINDOWS

Es bestehen folgende Unterschiede zwischen gleichnamigen Datentypen in SESAM V1 und UDS und DRIVE/WINDOWS:

Datentyp	**SESAM V1 / UDS**	**DRIVE/WINDOWS**
CHARACTER [(*l*)]	$l \leq 256$	$l \leq 32000$

SESAM V2-Datentypen in DRIVE/WINDOWS

Es bestehen folgende Unterschiede zwischen gleichnamigen Datentypen in SESAM V2 und DRIVE/WINDOWS:

Datentyp	**SESAM V2**	**DRIVE/WINDOWS**
CHARACTER [(*l*)]	$l \leq 256$	$l \leq 32000$
NUMERIC [(*m*[,*n*])]	$m \leq 31$ Vorbelegung: m=1	$m \leq 15$ Vorbelegung: m=8
DECIMAL [(*m*[,*n*])]	$m \leq 31$ Vorbelegung: m=1	$m \leq 15$ Vorbelegung: m=15
FLOAT	FLOAT [(*m*)] *m*: Anzahl der Binärstellen Vorbelegung: m=1	FLOAT

SESAM V2-Datentypen werden folgendermaßen in DRIVE/WINDOWS abgebildet:

SESAM V2-Datentyp	**Abbildung in DRIVE/WINDOWS**
NUMERIC [(*m*[,*n*])]	NUMERIC [(*m*[,*n*])] XDEC [(*m*[,*n*])], wenn $m > 15$ [1)]
DECIMAL [(*m*[,*n*])]	DECIMAL [(*m*[,*n*])] XDEC [(*m*[,*n*])], wenn $m > 15$ [1)]
FLOAT [(*m*)] *m*: Anzahl der Binärstellen	REAL, wenn $m \leq 21$ DOUBLE PRECISION oder FLOAT, wenn $22 \leq m \leq 53$

1) In dynamischen SQL-Anweisungen werden NUMERIC- und DECIMAL-Datentypen mit $m > 18$ zwischenkonvertiert in den Datentyp DOUBLE PRECISION. Dadurch können Rundungsfehler auftreten.

3.2.2 Benutzerdefinierte Datentypen

Der benutzerdefinierte Datentyp wird im DRIVE-Programm mit der Anweisung DECLARE TYPE definiert worden sein. Die Anweisung DECLARE TYPE steht entweder im Deklarationsteil eines Programms oder unmittelbar vor der Anweisung PROCEDURE. Die DECLARE TYPE-Anweisung vor der PROCEDURE-Anweisung eines Programms ermöglicht es, an ein Programm benutzerdefinierte Datentypen zu übergeben.

Datentyp	**erlaubter Feldinhalt**	**Feldlänge in Byte**
usertyp	alle DRIVE-Datentypen	entsprechend dem DRIVE-Datentyp

Beispiele

Die Variable &pi hat den benutzereigenen Datentyp „pi_float“.

```
DECLARE TYPE pi_float  FLOAT INIT $PI;
...
DECLARE VARIABLE &pi   pi_float;
```

Die Variable &uhrzeit hat den benutzereigenen Datentyp „jetzt“.

```
DECLARE TYPE jetzt     TIME INIT NULL
                            MASK '''Es ist jetzt: ''ZH'' Uhr ''ZI'' Minuten''';
...
DECLARE VARIABLE &uhrzeit  jetzt;
```

Die Variable &wohnung hat den benutzereigenen Datentyp „adresse“.

```
DECLARE TYPE     1 adresse,
                 2 strasse  CHARACTER (30),
                 2 plz      NUM       (5) INIT '80798',
                 2 ort      CHARACTER (30);
...
DECLARE VARIABLE &wohnung   adresse;
```

Der benutzereigene Datentyp „adresse“ kann in strukturierten Variablen verwendet werden.

```
DECLARE VARIABLE &mitarbeiter,
                 2 nachname  CHARACTER  (25),
                 2 vorname   CHARACTER  (25),
                 2 wohnung   adresse,
                 2 gehalt    NUMERIC (7,2);
```

Der benutzereigene Datentyp „db_typ“ wird in einem Programm deklariert und in der USING-Leiste angegeben. DECLARE TYPE steht nach der Anweisung OPTION und vor der Anweisung PROCEDURE.

```
OPTION LISTING=LIST;
DECLARE TYPE 1 db_typ,
               2 db_system   CHARACTER (3),
               2 funktion,   CHARACTER (10),
               2 daten,
                 3 spalte    CHARACTER  (20),
                 3 art_name  CHARACTER  (15);
PROCEDURE h_prog USING &db_parameter db_typ;
...
```

3.2.3 Strukturierte Datentypen

Ein strukturierter Datentyp faßt einzelne Komponenten zusammen. Varianten des strukturierten Datentyps sind

– Vektor
– Matrix
– Datengruppe
– Wiederholungsgruppe

Ist eine Variable strukturiert, bedeutet das, daß diese Variable aus Komponenten verschiedener oder gleicher Datentypen besteht. Eine strukturierte Variable kann als gesamte Einheit angesprochen werden oder Sie können einzelne Komponenten daraus in DRIVE-Anweisungen ansprechen. Die Hierarchie in Daten- und Wiederholungsgruppen wird über Stufennummern festgelegt. Die Regeln für die Vergabe von Stufennummern finden Sie im Abschnitt „Alle Variablenarten definieren“ auf Seite 40.

Vektor

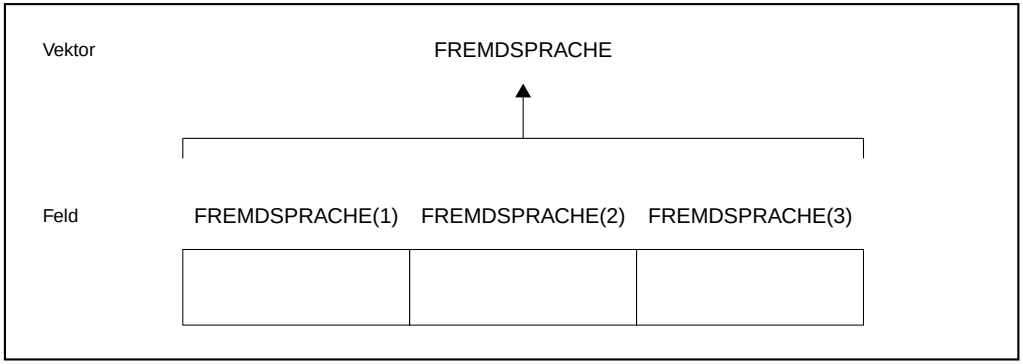
Ein Vektor ist ein Feld mit Wiederholungsfaktor. Der Wiederholungsfaktor gibt an, wie viele Ausprägungen des Feldes zu dem Vektor zusammengefaßt sind, wobei
$1 \leq$ *wiederholungsfaktor* ≤ 255.

Beispiel

```
&FREMDSPRACHE (3) CHARACTER (10)
```

Wiederholungsfaktor

Das alphanumerische Feld FREMDSPRACHE mit der Länge 10 kommt 3 mal vor.



Matrix

Eine Matrix besitzt eine zweidimensionale Ausprägung. In der Horizontalen befindet sich ein Vektor, dessen Komponenten in der Vertikalen mehrfach ausgeprägt sind.
$1 \leq spalten \leq 255$; $1 \leq zeilen \leq 255$.

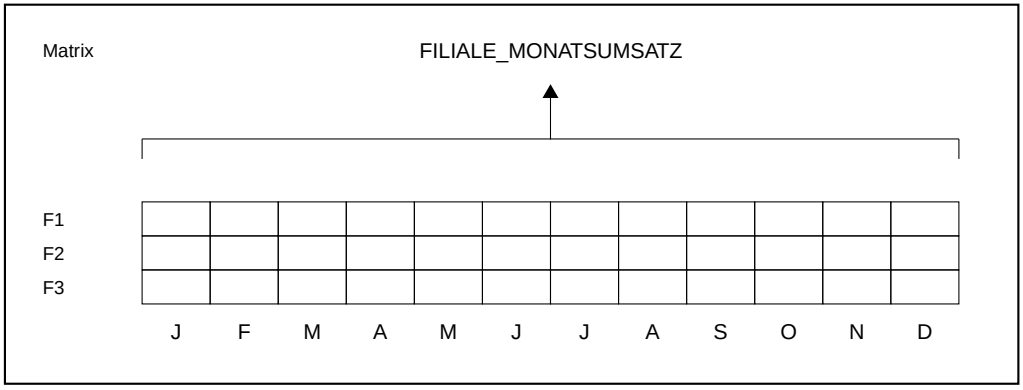
Beispiel

```
&FILIALE_MONATSUMSATZ (3,12) NUMERIC (7,2)
```

→ Wiederholungsfaktor in der Horizontalen (Spalten)

→ Wiederholungsfaktor in der Vertikalen (Zeilen)

Das numerische Feld FILIALE_MONATSUMSATZ mit der Länge 7, davon 2 Nachkommastellen, kommt 36 mal vor. Damit können die Monatsumsätze von 3 Filialen (F1, F2, F3) für ein Jahr dargestellt werden.



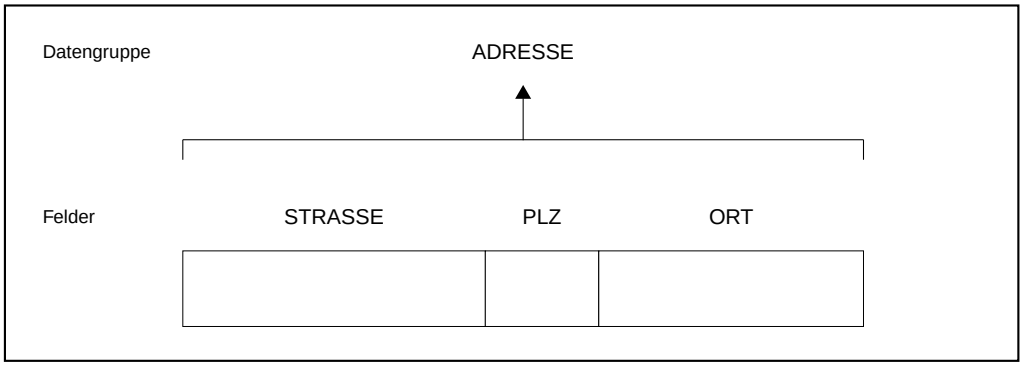
Datengruppe

Eine Datengruppe ist eine Zusammenfassung von Komponenten verschiedener oder gleicher Datentypen. Die Komponenten der Datengruppe können ein Feld, ein Vektor, eine Datengruppe oder eine Wiederholungsgruppe sein.

Beispiel

```
1 &ADRESSE,                    → Datengruppe
  2 STRASSE  CHARACTER(20),    → Feld
  2 PLZ      NUMERIC(5),       → Feld
  2 ORT      CHARACTER(20);    → Feld
```

Die Datengruppe ADRESSE besteht aus Komponenten, die alphanumerische oder numerische Felder sind. „1“ ist die Stufennummer der höchsten Hierarchiestufe; „2“ ist die Stufennummer, die die nächstniedrigere Hierarchiestufe qualifiziert.



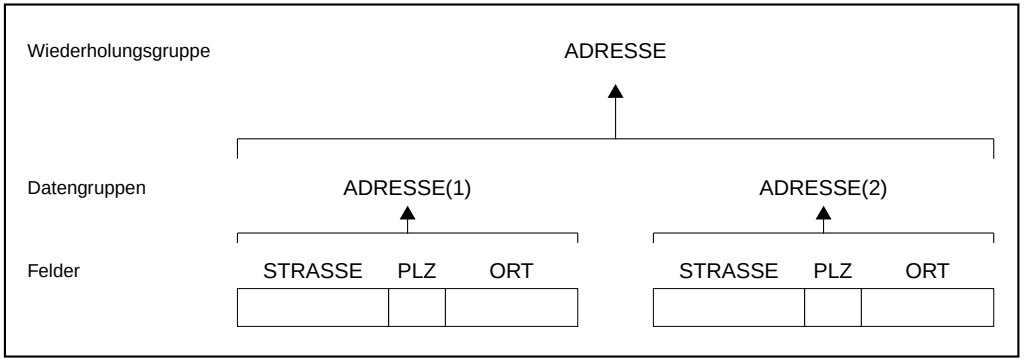
Wiederholungsgruppe

Eine Wiederholungsgruppe besteht aus *x* Wiederholungen einer Datengruppe. Der Wiederholungsfaktor *x* gibt an, wie viele Ausprägungen der gleichen Datengruppe zusammengefaßt werden. 0 < *wiederholungsfaktor* < 256.

Beispiel

```
1 &ADRESSE  (2)                 → Wiederholungsgruppe
  2 STRASSE   CHARACTER(20),    → Feld   }
  2 PLZ       NUMERIC(5),       → Feld   } Datengruppe
  2 ORT       CHARACTER(20);    → Feld   }
```

Die Datengruppe ADRESSE kommt zweimal vor und ist deshalb eine Wiederholungsgruppe. Jede Datengruppe ADRESSE enthält die Felder STRASSE, PLZ und ORT.



Strukturierte Datentypen in Datenbanksystemen

Strukturierte Datentypen von DRIVE/WINDOWS werden in folgenden Datenbanksystemen unterstützt:

Strukturierter Datentyp	SESAM	UDS
Vektor	ja [1)]	ja
Matrix	nein	nein
Datengruppe	nein	ja
Wiederholungsgruppe	nein	ja

[1)] Vektoren dürfen in SESAM ab V2.x nicht den Datentyp VARCHAR haben.

Bei Verteilten UTM-Anwendungen (VTV) und Verteilten Anwendungen gelten weitere Einschränkungen (siehe Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339 Seite 299).

3.3 Variablen definieren

Im Deklarationsteil des Programms definieren Sie die Variablen mit der Anweisung DECLARE VARIABLE (siehe DRIVE-Lexikon [3], Anweisung DECLARE VARIABLE).

Wenn Sie nicht explizit einen Anfangswert zuweisen, werden die Variablen bei der Definition mit Standard-Werten initialisiert.

3.3.1 Alle Variablenarten definieren

Für das Definieren von Variablen können Sie beliebige Namen aus dem DRIVE-Zeichenvorrat bilden. Der Variablenname muß mit dem &-Zeichen beginnen und darf exklusive „&“ maximal 31 Zeichen lang sein (siehe „DRIVE- Lexikon“ [3], Metavariablen *variable*, *vektor*, *matrix*, *datengruppe* und *wiederholungsgruppe*).

Enthält der Name einer Variablen außer dem ersten &-Zeichen Sonderzeichen, so muß er in Anführungszeichen stehen, z.B. "&monatsumsatz / filiale".

Einfache Variablen jeden Datentyps definieren

Definition		Bedeutung
DECLARE VARIABLE	&ENDE CHAR,	alphanumerische Variable der Länge 1
	&ARTNR CHAR (6),	alphanumerische Variable der Länge 6
	&ZAHL INT,	Variable vom Datentyp INTEGER
	&NR SMALLINT,	Variable vom Datentyp SMALLINTEGER
	&PREIS NUM(7,2),	numerische Variable der Länge 7 davon 2 Nachkommastellen
	&ERG DEC(6,2),	Variable vom Datentyp DECIMAL der Länge 6, davon 2 Nachkommastellen
	&ZEIT TIME,	Variable vom Datentyp TIME
	&DATUM DATE;	Variable vomDatentyp DATE

Vektor definieren

```
DCL VAR &FREMDSPRACHE (3) CHAR(10);
```

Es wird ein Vektor mit 3 Feldern definiert. In jedem Feld kann ein alphanumerischer Wert mit 10 Byte Länge abgelegt werden.

Matrix definieren

```
DCL VAR &FILIALE_UMSATZ (3,12) NUM(7,2);
```

Es wird eine Matrix mit 3 x 12 Feldern definiert. In jedem Feld kann eine numerische Variable mit 7 Stellen, davon 2 Nachkommastellen, abgelegt werden.

Datengruppe definieren

```
DCL VAR 1 &NAME,
              2 VORNAME  CHAR(15),
              2 NACHNAME CHAR(20);
```

Es wird eine Datengruppe mit 2 einfachen Komponenten (VORNAME und NACHNAME) definiert. In jeder Komponente kann ein alphanumerischer Wert abgelegt werden.

Wiederholungsgruppe definieren

```
DCL VAR 1 &NAME (3),
            2 VORNAME  CHAR(15),
            2 NACHNAME CHAR(20);
```

Es wird eine Datengruppe mit dem Wiederholungsfaktor 3 und somit eine Wiederholungsgruppe definiert.

Vergabe von Stufennummern

Bei der Definition von Datengruppen und Wiederholungsgruppen wird eine Struktur mit Stufennummern aufgebaut. Die Vergabe von Stufennummern unterliegt folgenden Regeln:

– Die höchste Hierarchiestufe muß die Stufennummer 1 erhalten.

– Untergeordnete Stufen erhalten höhere Nummern.

– Eine Vereinbarung mit Stufennummer x (x > 1) ist eine Komponente der nächsten ihr vorausgehenden Vereinbarung, deren Stufennummer kleiner als x ist.

– Komponenten, die auf gleicher Hierarchiestufe stehen, erhalten gleiche oder niedrigere Stufennummern, jedoch nicht niedrigere als die der darüberliegenden Hierarchiestufe. Dabei stellt 1 immer die höchste, 49 immer die niedrigste mögliche Stufe dar.

– Die Stufennummern können aufeinanderfolgend (1, 2, 3 ...) oder mit Lücken (1, 3, 6 ...) vergeben werden. Dadurch können später noch Korrekturen oder Erweiterungen eingefügt werden.

– Eine niedrige Hierarchiestufe darf keine Stufennummer haben, die vom Wert her kleiner ist als die Stufennummer einer höheren Stufe.

– Jede Komponente kann selbst zu einer Struktur werden, wenn ihrer Vereinbarung eine Vereinbarung mit höherer Stufennummer folgt (siehe z.B. Variable 3 B in Beispiel 1).

– Die Schachtelungstiefe von Datengruppen ist maximal 49, wobei wiederum maximal 3 Wiederholungsgruppen ineinander geschachtelt sein dürfen.

Beispiel 1

richtig:

```
1 &A,
    3 B,
        8 C INT,
  2 D,
        7 E INT;
```

falsch:

```
2 &A,            → Fehler, die höchste Hierarchiestufe muß Stufennummer 1 haben
    3 B,
      8 C INT,
  2 D,
      7 E INT;
```

falsch:

```
1 &A,
    3 B,
      2 C INT,   → Fehler, untergeordnete Stufen müssen höhere Nummern haben
  4 D,
      7 E INT;
```

Beispiel 2

Die linke Definition entspricht der rechten:

```
DCL VAR 1 &RECORD,                    DCL VAR 1 &RECORD,
            5 ART     CHAR (02),                2 ART       CHAR (02),
          4 NAME,                               2 NAME,
              10 VOR    CHAR (10),                3 VOR     CHAR (10),
            5  NACH   CHAR (15),                  3 NACH    CHAR (15),
        3 ADRESSE,                              2 ADRESSE,
            8 ORT     CHAR (12),                3 ORT     CHAR (12),
            8 STRASSE CHAR (12),                3 STRASSE CHAR (12),
            8 FILLER  CHAR (12);                3 FILLER  CHAR (12);
```

Beispiel 3

Definieren einer Datengruppe mit Wiederholungsgruppen

```
DCL VAR 1 &VAR_A,
          2 A2 (2),
            3 A3,
              4 A4 (5),
                5 A5           INT INIT 1,
                5 A6 (2),
                  6 A7         INT,
                  6 A8         INT,
              4 A9,
                5 A10,
                  6 A11 (10)   INT INIT 2,
                  6 A5         INT,
           2 A12 (3)           INT;
```

Einsatz von FILLER

Anstelle einer Bezeichnung für eine einfache Komponente kann auf einer Stufe > 1 auch FILLER angegeben werden. Ein mit FILLER benanntes Feld kann einzeln nicht angesprochen werden, sondern nur innerhalb eines Aggregats. Mit Hilfe eines Aggregats kann jedem Feld ein bestimmter Wert zugewiesen werden: Feld 1 erhält den Wert 'ENG', Feld 2 'FRA', Feld 3 'SPA'. Ein Aggregat stellt eine Liste von Werten dar, die von spitzen Klammern eingeschlossen und jeweils durch ein Komma voneinander getrennt sind. Auch der NULL-Wert kann angegeben werden. Wert kann sein: eine nicht strukturierte Variable, ein Literal, ein Aggregat oder die Zahl $PI (siehe DRIVE-Lexikon [3], Metavariable *wert*).

Beispiel

```
SET &RECORD.ADRESSE = <'NEW YORK', 'FIFTH AVENUE','USA'>;   → richtig
SET &RECORD.ADRESSE.FILLER = 'USA';                         → falsch
```

Variable als Index definieren

Ausprägungen von Variablen werden über Indizes angesprochen. Eine Variable kann aber auch selbst als Index verwendet werden. Index-Variablen dürfen keine Nachkommastellen haben.

Beispiel

```
DCL VAR &IND NUM (2) INIT 1;
DCL VAR &FREMDSPRACHE (3) CHAR (10);
```

&FREMDSPRACHE ist ein Vektor. Für die Indizierung der einzelnen Ausprägungen wird die Variable &IND definiert. Die Ausprägungen des Vektors &FREMDSPRACHE können nun in den Verarbeitungsanweisungen des DRIVE-Programms mit &FREMDSPRACHE(&IND) angesprochen werden.

3.3.2 Strukturierte Variablen qualifizieren

Wenn Sie einer Variablen einen Wert zuweisen wollen oder sie nach einem Wert abfragen, müssen Sie sie ansprechen, d.h. qualifizieren, können. Entscheidend ist die korrekte Qualifizierung bei Variablen strukturierten Datentyps: Vektor, Matrix und Gruppe (Datengruppe und Wiederholungsgruppe).

Vektor qualifizieren

```
DCL VAR &FREMDSPRACHE (3) CHAR(10);
```

– Jedes Feld (Komponente) ansprechen:

```
SET &FREMDSPRACHE = 'ENG';
```

Jedes der 3 Felder erhält den Wert 'ENG'.

– Ein Feld (Komponente) ansprechen:

```
SET &FREMDSPRACHE (2) = 'ENG';
```

Das 2. Feld erhält den Wert 'ENG'.

– Jedes Feld (Komponente) mit einem anderen Wert belegen:

```
SET &FREMDSPRACHE = <'ENG', 'FRA', 'SPA'>;
```

Mit Hilfe eines Aggregats kann jedem Feld ein bestimmter Wert zugewiesen werden: Feld 1 erhält den Wert 'ENG', Feld 2 'FRA', Feld 3 'SPA'. Ein Aggregat stellt eine Liste von Werten dar, die von spitzen Klammern eingeschlossen und jeweils durch ein Komma voneinander getrennt sind. Auch der NULL-Wert kann angegeben werden. Wert kann sein: eine nicht-strukturierte Variable, ein Literal, ein Aggregat oder die Zahl π (pi) (siehe DRIVE-Lexikon [3], Metavariable *wert*).

Matrix qualifizieren

```
DCL VAR &FILIALE_MONATSUMSATZ (3,12) NUM(7,2);
```

– Jedes Feld (Komponente) ansprechen:

```
SET &FILIALE_MONATSUMSATZ = 3500;
```

Jedes der 36 Felder erhält den Wert 3500.

– Ein Feld (Komponente) ansprechen:

```
SET &FILIALE_MONATSUMSATZ (2,7) = 3500;
```

Dem Feld in Zeile 2, Spalte 7, wird der Wert 3500 zugeordnet, d.h. die Filiale 2 hat im Monat Juli einen Umsatz von 3500.

Datengruppe qualifizieren

```
DCL VAR 1 &ADRESSE,
          2 STRASSE  CHAR(20),
          2 PLZ      NUM(5),
          2 ORT      CHAR(20);
```

– Eine Komponente ansprechen:

  Die Variable &ADRESSE ist eine strukturierte Variable, die aus drei Komponenten besteht. Jede dieser Komponenten ist direkt ansprechbar. Somit kann eine Komponente eine „eigenständige" Variable sein. Voraussetzung ist, daß die strukturierte Variable eindeutig qualifiziert bzw. teilqualifiziert wird.

  ```
  SET &ADRESSE.PLZ = 80000;
  ```

  Die Variable &ADRESSE.PLZ erhält den Wert 80000.

  ```
  SET &PLZ = 80000;
  ```

  Hier wird nicht voll qualifiziert, sondern eine Variable über Teilqualifizierung angesprochen. Solange die Eindeutigkeit gewahrt ist, ist dies erlaubt.

– Alle Komponenten einer Datengruppe ansprechen:

  Mit Hilfe eines Aggregats können alle Komponenten einer Variablen angesprochen werden.

  ```
  SET &ADRESSE = <&STRASSEVAR,  '80000', 'MUENCHEN'>;
  ```

Wiederholungsgruppe qualifizieren

```
DCL VAR 1 &ADRESSE (2),
          2 STRASSE  CHAR(20),
          2 PLZ      NUM(5),
          2 ORT      CHAR(20);
```

– Eine Komponente ansprechen:

  Die Variable &ADRESSE ist eine strukturierte Variable, die aus 2 mal 3 Komponenten besteht. Jede dieser Komponenten ist direkt ansprechbar. Somit kann eine Komponente eine „eigenständige" Variable sein. Voraussetzung ist, daß die strukturierte Variable eindeutig qualifiziert bzw. teilqualifiziert wird.

  ```
  SET &ADRESSE(1).PLZ = '80000';
  ```

  Die Variable &PLZ der ersten Ausprägung von &ADRESSE erhält den Wert 80000 .

– Von Wiederholungsgruppen dürfen nur einzelne Ausprägungen, d.h. Datengruppen, qualifiziert werden. Ausprägungsbereiche sind nur als Basiskomponente erlaubt.

Die folgenden Beispiele sollen nochmals verdeutlichen, was Eindeutigkeit bei der Qualifizierung von Variablen bedeutet.

Beispiel 1

Die folgenden SET-Anweisungen beziehen sich auf diese Variablendefinition:

```
DCL VAR 1 &B1    CHAR(2);
DCL VAR 1 &V,
          2 A,
            3 A1 INT,
            3 A2 CHAR(2),
          2 B,
            3 B1 INT,
            3 A2 INT;
```

Eindeutige Zuweisungen:

```
Auf 3 A1 INT (4 Möglichkeiten):
SET &V.A.A1 = 1,
    &A.A1   = 1,
    &A1     = 1,
    &V.A1   = 1;
```

```
Auf 3 A2 INT (2 Möglichkeiten):
SET &B.A2   = 1,
    &V.B.A2 = 1;
```

Falsche Zuweisung:

Auf die Variable 3 B1 INT soll zugegriffen werden.

```
SET &B1 = 1
```

falsch, da `1 &B1 CHAR(2)` angesprochen

Nicht eindeutige Zuweisung:

```
SET &A2 = 1;
```

falsch, da 3 A2 CHAR(2) oder 3 A2 INT nicht eindeutig

Beispiel 2

Das Folgende bezieht sich auf diese Variablendefinition:

```
DCL VAR &L CHAR,
        1 &K,
          2 L,
            3 L INT;
DCL VAR 1 &N,
          2 N,
            3 N INT,
            3 M INT;
```

Sie können auf Variablen jeder Stufennummer zugreifen. Voraussetzung ist, daß die betreffende Variable eindeutig angegeben wird.

Zugriff auf die Datengruppe &K: Die Angabe &K ist eindeutig.

Zugriff auf die Variable 3 N, die eine einfache Komponente der strukturierten Variablen &N ist: Die Angabe &N.N.N ist eindeutig.

Zugriff auf das CHAR-Feld &L: Die Angabe &L ist eindeutig.

Zugriff auf 3 L vom Datentyp INT: Die Angaben &L.L und &K.L.L sind eindeutig. Die Angabe &K.L ist zwar eindeutig, greift jedoch auf 2 L zu.

Beispiel 3

Das folgende Beispiel zeigt, daß von Wiederholungsgruppen nur einzelne Ausprägungen qualifiziert werden dürfen.

```
DCL VAR 1 &v(3),
          2 w(2),
            3 x(4) INTEGER;
```

Erlaubt ist die Angabe &v(2).w(1).x(3) oder &v(1).w(1).x(2-4). Nicht erlaubt ist die Angabe &v(3).w(1-2).x(4), weil Ausprägungsbereiche nur als letzte angegebene Komponente erlaubt sind: &v(1).w(1-2).

Ausgeben von Variablen am Bildschirm

Sie können die Inhalte der Variablen mit der DISPLAY FORM-Anweisung ausgeben:

```
DISPLAY FORM LINE VALUES &VAR;
```

3.3.3 LIKE-Klausel (Strukturen kopieren)

Die LIKE-Klausel ermöglicht Funktionen, die einem Kopieren ähnlich sind:

Die Datenstruktur einer Gruppe (≡ Datengruppe oder Wiederholungsgruppe) kann in eine Variable übernommen werden. Daneben ist auch LIKE auf Tabellen, Cursor und Views möglich (siehe „DRIVE-Lexikon“ [3], Metavariable *strukturtyp*).

Einschränkungen für die LIKE-Klausel:

– LIKE ist nicht erlaubt bei einfachen Variablen, Vektoren und Matrizen.

– Indizierung ist nicht erlaubt, d.h., LIKE ist nur auf eine komplette Gruppe möglich.

– Mit LIKE erzeugte Gruppen dürfen keine weiteren Komponenten haben.

– LIKE ist nicht erlaubt auf mit LIKE definierte Variablen.

- LIKE auf eine andere Gruppe

  Die Struktur der Datengruppe &V1 wird in die Variable &V2 übernommen. &V2 ist identisch mit &V1.

```
DCL VAR 1 &V1,
          2  V11 CHAR,
          2 V12 INT,
        1 &V2 LIKE &V1;      → 1 &V2,
                                 2 V11 CHAR,
                                 2 V12 INT;
```

- LIKE innerhalb einer Gruppe

  Die Komponente V32 wird genauso definiert wie Komponente V31.

```
1 VAR 1 &V3,                    1 &V3;
        2 V31,                    2 V31;
          3 V311 CHAR,              3 V311 CHAR,
          3 V321 NUM(4,2);          3 V312 NUM(4,2),
       2 V32 LIKE &V31;      →    2 V32,
                                    3 V311 CHAR,
                                    3 V312 NUM(4,2)
```

- LIKE innerhalb einer Gruppe mit nachfolgenden Komponenten

  Die Komponente V41 wird von V42 übernommen. Nach der LIKE-Klausel folgen weitere Vereinbarungen (Komponenten V43, V44)

```
DCL VAR 1 &V4,                          1 &V4,
          2 V41,                          2 V41,
            3 V411 CHAR,                    3 V411 CHAR,
            3 V412 NUM(4,2)                 3 V412 NUM(4,2),
          2 V42 LIKE &V41,        →       2 V42,
          2 V43 CHAR,                     3 V411 CHAR,
          2 V44 CHAR,                     3 V412 NUM(4,2),
            3 V441 CHAR,                  2 V43 CHAR,
            3 V442 INT;                   2 V44,
                                            3 V441 CHAR,
                                            3 V442 INT;
```

- LIKE auf eine Tabelle

  Die Struktur der Basistabelle PERSONALDB wird in die Variable &TABLEVAR übernommen.

```
DCL VAR  &TABLEVAR LIKE TABLE PERSONALDB;      →    1 &TABLEVAR,
                                                      2 PERSNR CHAR(6),
                                                      2 NAME,
                                                        3 VORNAME CHAR(20),
                                                        3 NACHNAME CHAR (20);
```

- LIKE auf einen Cursor

```
DCL CUR_G CURSOR FOR SELECT G_NR, G_JOIN_ITEM
            FROM PERSONALDB;
DCL VAR &CUR_G LIKE CURSOR CUR_G;      →    1 &CUR_G,
                                              2 G_NR INT,
                                              2 G_JOIN_ITEM INT;
```

- LIKE auf einen View

```
CRE TEMP VIEW VIEW_G (FELD1, FELD2, FELD3, FELD4)
   AS SELECT G_NR, G_VEKT (3), G_VEKT (5), G_GEHALT + 1000
        FROM PERSONALDB;
DCL VAR &VIEW_G LIKE TABLE VIEW_G;       →    1 &VIEW_G,
                                                2 FELD1 CHAR (6),
                                                2 FELD2 (3) CHAR (15),
                                                2 FELD3 (5) NUM (7,2),
                                                2 FELD4 NUM (7,2);
```

- Nicht erlaubt: LIKE mit Indizierung (siehe oben Einschränkungen)

```
DCL VAR 1 &A (5),
    2 A1 INT,
    2 B1 INT;
...
DCL VAR &B LIKE &A(2);      → Fehler: Indizierung nicht erlaubt
                  *
```

- Nicht erlaubt: LIKE und weitere Komponenten (siehe oben Einschränkungen)

```
DCL VAR 1 &Z,
           2 Z1 LIKE &A,
              3 Z2 CHAR;      → Fehler: keine weiteren Komponenten erlaubt
                *
```

Namensvergabe bei LIKE TABLE/CURSOR

Bei der Definition von Variablen mit der Angabe LIKE TABLE/CURSOR sind Regeln bei der Namensvergabe zu beachten. Abhängig von der jeweiligen Definition können Zugriffe auf einfache Komponenten nicht eindeutig sein.

- Qualifizierung bei Vektoren

  Ausgangssituation: In der Tabelle T gibt es ein Feld V(10) CHAR.

  Feld V stellt damit einen Vektor mit dem Wiederholungsfaktor 10 dar.

  1. Keine Indizierung bei der Cursor-Definition

```
DCL C CURSOR FOR SELECT V    FROM T;   → keine Indizierung des Vektors V
DCL VAR &VC LIKE CURSOR C;             → LIKE auf den Cursor C

→     1 &VC;                           → Alle Ausprägungen ansprechbar
        2 V(10) CHAR;
```

     Keine Indizierung bei der Cursor-Definition: es entsteht die Variable &V (Vektor) mit 10-facher Ausprägung, jede davon ist ansprechbar, z.B.

```
SET &V(5) = 100;
```

  2. Angabe eines Index: Es entstehen einfache Komponenten

```
DCL C CURSOR FOR SELECT V(7) FROM T;   → Indizierung des Vektors V
DCL VAR &VC LIKE CURSOR C;             → LIKE auf den Cursor C

→     1 &VC;                           → Nicht mehr alle Ausprägungen
        2 V CHAR;                        ansprechbar, es entsteht eine
                                         einfache Komponente
```

Indizierung bei der Cursor-Definition (7. Ausprägung); es entsteht die Variable &V, d.h. bei einer Indizierung entstehen einfache Komponenten.

3. Angabe eines Indexbereichs

```
DCL C CURSOR FOR SELECT V(2–5) FROM T;   → Angabe eines Indexbereichs (2-5)
DCL VAR &VC LIKE CURSOR C:               → LIKE auf den Cursor C

→     1 &VC,                             → Alle 4 Ausprägungen ansprechbar
        2 V(4) CHAR;
```

Angabe eines Indexbereiches: es entsteht eine strukturierte Variable &V mit 4 Ausprägungen, jede davon ist eindeutig qualifizierbar.

4. Nicht erlaubt: Angabe mehrerer Vektoren

```
DCL C CURSOR FOR SELECT V(3), V(5) FROM T;   → Angabe mehrerer Vektoren
DCL VAR &VC LIKE CURSOR C;                   → LIKE auf den Cursor C

→     1 &VC,                                 → Fehler, da gleichnamige
        2 V CHAR,                              und somit nicht eindeutig
        2 V CHAR,                              qualifizierbare Komponenten
                                               entstehen („FILLER“)
```

Bei Angabe von mehr als einer Ausprägung eines Satzelementes über Indizes wird für die erste Ebene der Name übernommen (hier „V“), für alle weiteren Ebenen wird „FILLER“ genommen. Ein mit FILLER benanntes Feld kann nicht einzeln angesprochen werden. Dieses Verhalten ist wichtig bei SESAM/SQL V2, wenn z.B. bei SELECT ... FROM tab1, tab2 ... in beiden Tabellen gleichnamige Ebenen vorkommen.

- Qualifizierung bei Gruppen

  Ausgangssituation: In der Tabelle T gibt es die Datengruppe

```
1 G,
  2 A CHAR,
  2 B CHAR,
  2 C INT;
```

Die Anweisung DCL VAR &VC LIKE CURSOR C übernimmt die Datengruppe in die Variable &VC. Die Komponentennamen der Variablen sind jedoch dann nicht mehr erkennbar im Vergleich zu den ursprünglichen Namen aus der Tabelle.

```
DCL C CURSOR FOR SELECT G.C FROM T;
DCL VAR &VC LIKE CURSOR C;          →   1 &VC,
                                          2 C INT;
```

Bei der Cursor-Deklaration wird auf das Satzelement G.C zugegriffen. Bei der Definition der strukturierten Variablen &VC entsteht ein neuer Name für die einfache Komponente C. Mit anderen Worten: der ursprüngliche Pfadname G.C wird bei der Variablendefinition nicht übernommen; es entsteht vielmehr ein neuer Pfadname &VC.C.

- Qualifizierung bei Ausdrücken

  Ausgangssituation: In der Tabelle T gibt es das Satzelement A INT.

```
DCL C CURSOR FOR SELECT A + 10, A + 12 FROM T;
DCL VR &VC LIKE CURSOR C;                    →   1 &VC,
                                                   2 INT,
                                                   2 INT;
```

  DRIVE/WINDOWS gibt Komponenten einer mit LIKE CURSOR/TABLE definierten Variablen keine Namen, wenn die Komponenten Ausdrücke sind. Die einfachen Komponenten von &VC sind nicht ansprechbar, wohl aber die Variable &VC. Es liegt kein Programmfehler vor. Die Variable &VC kann mit DISPLAY ausgegeben werden.

3.3.4 REDEFINES-Klausel (Speicherbereich mehrfach belegen)

Verwenden Sie die REDEFINES-Klausel möglichst nicht statt heterogener Zuweisungen, da sie maschinenabhängig ist. Auf verschiedenen Rechnern kann es zu unterschiedlichen Ergebnissen kommen. Nutzen Sie die erweiterten CHAR- und NUM-Funktionen, bei denen das DRIVE-RTS (Runtime-System) die Portabilität gewährleistet (siehe Abschnitt „Stringfunktionen“ auf Seite 92 und „DRIVE-Lexikon“ [3], Metavariablen *charprim* und *wertefunktion*).

Sie können mit der REDEFINES-Klausel für einen Speicherbereich einer Variablen mehrere Beschreibungen angeben (siehe „DRIVE-Lexikon“ [3], Metavariable *basistyp*).

REDEFINES wird angegeben in der DECLARE VARIABLE-Anweisung, also bei der Variablen-Definition.

... *stufennummer variable1* [*grunddatentyp*] REDEFINES *variable2*

(*stufennummer*, *variable1/2* und *grunddatentyp* sind nicht Teil der REDEFINES-Klausel; sie werden hier nur zur Verdeutlichung angegeben.)

variable1 ist der Name der neu zu belegenden Variablen. Sie ist die redefinierende Variable. *variable2* benennt den Namen der ersten Definition der Variablen, die neu belegt und somit redefiniert wird.

Die durch *variable2* spezifizierte Variable muß definiert sein. Der Speicherbereich der Variablen *variable1* darf nicht größer sein als derjenige von *variable2*.

Ist *variable1* eine Komponente einer Gruppe, so muß *variable2* Komponente derselben Hauptstruktur (Stufennummer 1) sein und vor *variable1* definiert sein.

variable2 darf nicht selber eine redefinierte Variable sein. Ist *variable2* eine Gruppe, so darf *variable*1 keine Komponente von *variable2* sein.

Die Anzeigebereiche, die angeben, ob NULL-Werte vorhanden sind oder nicht, werden von REDEFINES nicht überlagert. REDEFINES ist bei der INIT- und der USING-Klausel nicht erlaubt.

Datenwerte können nur bis zu einer Länge von 1840 Zeichen am Bildschirm dargestellt werden. Längere Datenwerte werden nach dem 1840-ten Zeichen bei der Ausgabe abgeschnitten. Dieses Abschneiden können Sie durch Redefinieren eines Feldes als Vektor mit n Komponenten von je 1840 Bytes umgehen. Verwenden Sie die REDEFINES-Klausel aber möglichst nicht statt heterogener Zuweisungen, da sie maschinenabhängig ist (siehe oben).

Die redefinierte Variable (*variable2*) darf nicht indiziert angegeben werden.

- REDEFINES auf eine andere Variable:

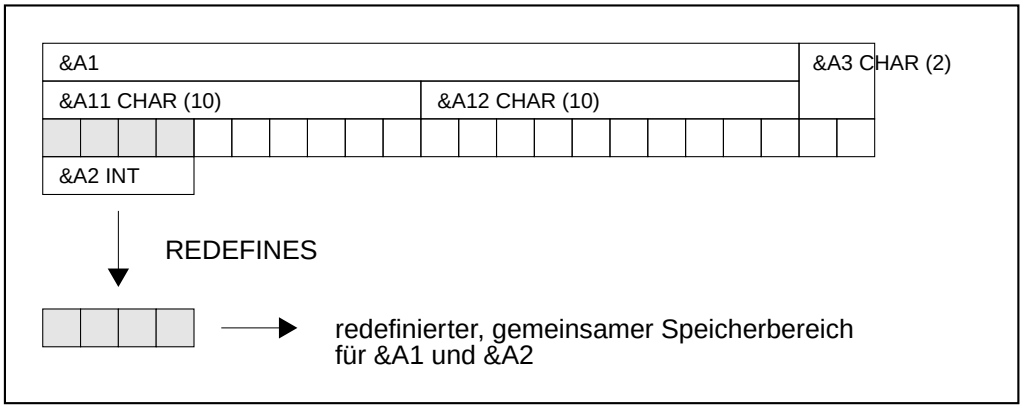
```
DCL VAR &A CHAR(8),
        &B CHAR(2) REDEFINES &A;
```

  &B ist *variable1*; &A ist *variable2*. &B belegt den &A zugeordneten Bereich neu: die ersten beiden Zeichen von CHAR(8).

- REDEFINES innerhalb einer Struktur

```
DCL VAR 1 &ALPHA,
          2 A1,
            3 A11 CHAR (10),
            3 A12 CHAR (10),
          2 A2 INT REDEFINES &A1,
          2 A3 CHAR(2);
```

  &A1 ist `variable2`; &A2 ist `variable1`. &A2 belegt den &A1 zugeordneten Bereich neu, d.h. den Bereich, der von &A11 und &A12 belegt ist; &A2 kann allerdings einen Bereich nur soweit redefinieren, wie es seine eigene Größe erlaubt: hier nur bis 4 Zeichen von &A11.



Die Neubelegung beginnt bei &A2 und endet bei der nächsten Stufennummer 2, also der Stufennummer, die vor &A3 steht.

- Mehrfache Neubelegungen durch REDEFINES

```
DCL VAR 1 &C,
          2 C1,
            3 C11     CHAR(7),
          2 C2        REDEFINES &C1,
            3 C21     CHAR(5),
          2 C3        REDEFINES &C1,
            3 C31     CHAR;
```

Die Variable &C1 wird zweimal redefiniert: von &C2 und von &C3.

- Nicht erlaubt: REDEFINES auf redefinierende Variable

```
DCL VAR 1 &BEISPIEL,
          2 ERST_ERKLAERUNG  CHAR (6),
          2 ZWEIT_ERKLAERUNG REDEFINES &ERST_ERKLAERUNG,
             3 A_1 INT,
             3 A_2 CHAR(6) REDEFINES &A_1;   → Fehler: REDEFINES auf
                           *                    redefinierende Variable
```

- Nicht erlaubt: REDEFINES auf Variablen, deren Speicherplatz kleiner ist

```
DCL VAR 1 &BEISPIEL,
          2 A_1 INT,
          2 A_2 CHAR(6) REDEFINES &A_1;   → Fehler: Der Speicherplatz von &A_2
                        *                    ist größer als der von &A_1
```

- Nicht erlaubt: REDEFINES bei INIT-Angabe

```
DCL VAR &A CHAR(8),
        &B CHAR(2) INIT 'VERBOTEN' REDEFINES &A;   → Fehler: REDEFINES
                                    *                bei INIT verboten
```

- Nicht erlaubt: REDEFINES bei USING-Angabe

```
PROC HHH USING &F CHAR(10), &J CHAR REDEFINES &F;   → Fehler: REDEFINES bei
                                     *                 USING verboten
```

- Nicht erlaubt: REDEFINES mit Indizierung

```
DCL VAR &A(10) CHAR,
        &AR CHAR REDEFINES &A;
DCL VAR &B(10) CHAR,
        &BR CHAR REDEFINES &B(1);   → Fehler: Indizierung verboten
                  *
```

- Nicht erlaubte Komponente redefiniert

```
DCL VAR 1 &F,
          2 F1 CHAR,
          2 F2 CHAR REDEFINES &F;   → Fehler: Redefinierte Variable direkte
                     *                Komponente der übergeordnetenGruppe
```

3.3.5 MASK-Klausel (Ausgabe aufbereiten)

Bei der Ausgabe einer Liste oder bei einer Bildschirmausgabe können Sie die Datendarstellung nach Ihren Wünschen aufbereiten. Zum Beispiel können Sie vorangehende Nullen durch Leerstellen ersetzen oder positive bzw. negative Zahlen als solche kennzeichnen.

Mit der MASK-Klausel legen Sie die Eigenschaften für die Ausgabeaufbereitung der Datentypen fest (siehe DRIVE-Lexikon [3], Metavariable *mask*). Der Inhalt der MASK-Klausel darf maximal 256 Zeichen lang sein.

Beispiel

Die Ausgabe einer Variablen mit sechs Vor- und zwei Nachkommastellen wird übersichtlicher dargestellt, wenn Sie folgende MASK-Klausel angeben.

```
DECLARE VARIABLE &betrag NUM(8,2) MASK '+SZZZZ9P99';
```

Enthält die Variable &betrag den Wert 000012.60, so gibt DRIVE/WINDOWS aus:

␣␣␣␣12.60

Die MASK-Klausel kann aus folgenden Angaben bestehen:

– Maskensteuerzeichen
– Text
– Wiederholungsfaktor

Die MASK-Klausel muß in einfachen Hochkommas eingeschlossen sein.

Maskensteuerzeichen

Maskensteuerzeichen für alphanumerische Datentypen:

X	EBCDIC-Zeichen

Maskensteuerzeichen für numerische Datentypen:

9	Ziffer
Z	führende Nullen durch Leerzeichen ersetzen
P	Dezimalpunkt
+	Vorzeichen wird unbedingt angegeben
-	nur negatives Vorzeichen (falls vorhanden) wird ausgegeben
S	gleitendes Vorzeichen und führende Nullen durch Leerzeichen ersetzen
E	Gleitpunktdarstellung
BWZ	Gesamtes Feld als Leerzeichen ausgeben, wenn Datenwert = 0
,	Einfügezeichen für Komma
.	Einfügezeichen für Punkt
B	Einfügezeichen für Leerzeichen
*	Schecksicherungssymbol: führende Nullen mit * ausgeben

Maskensteuerzeichen für Zeit-Datentypen:

YYYY	Jahr (vierstellig)
ZZZY	Jahr, führende Nullen durch Leerzeichen ersetzen
MO	Monat (zweistellig)
ZO	Monat, führende Null durch Leerzeichen ersetzen
DD	Tag (zweistellig)
ZD	Tag, führende Null durch Leerzeichen ersetzen

HH	Stunde (zweistellig)
ZH	Stunde, führende Null durch Leerzeichen ersetzen
MI	Minute (zweistellig)
ZI	Minute, führende Null durch Leerzeichen ersetzen
SS	Sekunde (zweistellig)
ZS	Sekunde, führende Null durch Leerzeichen ersetzen
FFF	Sekundenbruchteil (dreistellig)
ZZF	Sekundenbruchteil, führende Nullen durch Leerzeichen ersetzen
WW	Woche (zweistellig)
ZW	Woche, führende Null durch Leerzeichen ersetzen
JJJ	Julianischer Tag (= Tag im Jahr, dreistellig)
ZZJ	Julianischer Tag, führende Nullen durch Leerzeichen ersetzen
Q..Q	Tagesname
R..R	Monatsname
AP	amerikanische Angabe AM (vorm) oder PM (nachm)

Maskensteuerzeichen für den Datentyp INTERVAL:

9	Ziffer
Z	führende Nullen durch Leerzeichen ersetzen
+	Vorzeichen wird unbedingt angegeben
-	nur negatives Vorzeichen (falls vorhanden) wird ausgegeben
S	gleitendes Vorzeichen und führende Nullen durch Leerzeichen ersetzen
BWZ	Gesamtes Feld als Leerzeichen ausgeben, wenn Datenwert = 0

Text

Der Programmierer kann einen beliebigen Text angeben, der bei der Ausgabe unverändert bleibt. Der Text (auch Textteile) muß in doppelte Hochkommata eingeschlossen sein.

Wiederholungsfaktor n

n legt den Wiederholungsfaktor für ein Maskensteuerzeichen fest. Der Wiederholungsfaktor kann hinter den Maskensteuerzeichen X 9 Z S * Q und R angegeben werden (siehe Beispiele).

Beispiele

Datenwert	Definition der Darstellung	Ausgabe
12.60	NUM(6,2) MASK 'ZZZZP99'	␣␣12.60
12.60	NUM(6,2) MASK '+SZZZP99'	␣+␣12.60
12.60	NUM(6,2) MASK '****P99'	**12.60
12.60	NUM(6,2) MASK 'ZZZZP99''␣DM'''	␣␣12.60␣DM
-12.60	NUM(6,2) MASK '-ZZZZP99'	-␣␣12.60
12.60	NUM(6,2) MASK '-ZZZZP99'	␣␣␣12.60
12.60	NUM(6,2) MASK '99'	12
12.60	NUM(6,2) MASK '9999'	0012
1260	NUM(6,2) MASK 'ZZ,ZZZPZZ'	␣1,260.00
-1260	NUM(6,2) MASK 'ZZ,ZZZPZZ'	␣1,260.00
1260	NUM(6,2) MASK '-ZZ,ZZZPZZ'	␣␣1,260.00
126000	NUM(8,2) MASK 'ZZZBZZZPZZ'	126␣000.00
12.60	NUM(8,2) MASK 'ZZZBZZZPZZ'	␣␣␣␣␣12.60
12.60	NUM(6,2) MASK 'ZZZZP99 BWZ'	␣␣12.60
0	NUM(6,2) MASK 'ZZZZP99 BWZ'	␣␣␣␣␣␣␣
12.60	NUM(6,2) MASK 'E9'	␣1.2E+001
12.60	NUM(6,2) MASK 'E10'	␣1.26E+001
12.60	NUM(6,2) MASK 'E16'	␣1.26000000E+001
1260	INT MASK 'E9'	␣1.2E+003
12	INT MASK 'ZZP99'	12.00
1260	INT MASK 'ZZP99'	Fehlermeldung
1260	REAL MASK 'ZZ99'	1260
12.60	REAL MASK 'ZZ99'	␣␣13
1260	REAL MASK 'E9'	␣1.2E+003
1.2601260126	FLOAT MASK 'E16'	␣1.26012601E+000
DER␣PATE	CHAR (8) MASK '''DAS␣BUCH:␣''XXXXXXXX'	DAS␣BUCH:␣DER␣PATE
DER␣PATE	CHAR (8) MASK '''DAS␣BUCH:␣''X(8)'	DAS␣BUCH:␣DER␣PATE
1789-07-14	DATE MASK 'DD''.''MO''.''YYYY'	14.07.1789
0622-06-15	DATE MASK 'ZD''.''ZO''.''ZZZY'	15.␣6.␣622
1982-08-13	DATE MASK 'Q(10)'',␣DEN␣''ZD''.'''	FREITAG␣␣␣,␣DEN␣13.
20:15:26	TIME MASK 'HH''␣UHR␣''MI'	20␣UHR␣15

3.3.6 INIT-Klausel (Variable initialisieren)

Sie können einer Variablen mit der INIT-Klausel einen Anfangswert zuweisen. Dieser Wert ist ein Ausdruck, der nur Literale, NULL oder Funktionen, deren Argumente Literale sind, enthält. Die Initialisierungswerte, die für eine Variable vorbelegt sind, wenn keine INIT-Angabe erfolgt, können Sie im „DRIVE-Lexikon“ [3], Anweisung DECLARE VARIABLE nachlesen.

Bei einem LIKE auf eine Variable werden die Initialisierungswerte übernommen.

- Initialisierung von Variablen durch ein Literal

```
DCL VAR &ENDE CHAR INIT 'N';
DCL VAR &FREMDSPR (3) CHAR(4) INIT 'ENGL';   (alle 3 Ausprägungen erhalten den Wert ENGL)

DCL VAR 1 &A,
          2 B    CHAR(29),
          2 PLZ  NUM(5) INIT '80000',
          2 ORT CHAR(20) INIT 'MUENCHEN';
```

- Initialisierung von Variablen durch den NULL-Wert

```
DCL VAR &PROJEKT CHAR(14) INIT NULL;
```

- Nicht erlaubt: INIT bei der REDEFINES- und der USING-Klausel

```
PROC DDD USING &G INIT 6;   → INIT nicht erlaubt
```

3.3.7 CHECK-Klausel (Wertebereich prüfen)

DRIVE/WINDOWS bietet Ihnen mit der CHECK-Klausel ein mächtiges Sprachmittel für die Prüfung von Werten einschließlich des Folgedialogs im Fehlerfall. Die CHECK-Klausel hat einen geringen Realisierungsaufwand für den Programmierer und ist leicht zu verstehen: sie erspart Programmieraufwand, z.B. umständliche IF-THEN-oder CASE-Abfragen.

Mit der OPTION SCREENCHECK können Sie einstellen, ob auch Eingaben in FHS-Masken von DRIVE/WINDOWS überprüft werden (siehe „DRIVE-Lexikon“ [3]).

Der Wertebereich eines Datentyps wird durch eine Bedingung eingeschränkt und während des Programmablaufs immer dann überprüft, wenn der betreffenden Variablen ein Wert zugewiesen wird.

```
... CHECK &ARTNR > 'A00100' ... MESSAGE ...
```

An die CHECK-Klausel können Sie mit MESSAGE eine Meldung hängen. Diese Meldung wird anstelle der Standardmeldung am Bildschirm ausgegeben, wenn die durch CHECK geprüfte Bedingung nicht erfüllt ist. Dies gilt nur für Anweisungen, die im Zusammenhang mit Bildschirm-Formaten stehen: DECLARE FORM, FILL, DISPLAY.

Beispiel

```
DCL VAR &PLZ NUM(5) INIT 10000 CHECK &PLZ > 9999 AND &PLZ < 50001
               MESSAGE 'PLZ: Nur 10000 bis 50000 erlaubt!';
```

- CHECK auf eine einfache Variable

```
DCL VAR &A1 CHAR(8) CHECK &A1 <> 'x57';
```

- CHECK auf eine einfache Variable mit Meldung

```
DCL VAR &I INT CHECK &I  ≥ 0 MESSAGE 'I = NEGATIV';
```

- CHECK mit VALUE-Angabe statt Wiederholung der Variablen in CHECK-Klausel

```
DCL VAR &A CHAR (255) INIT 'A' (255) CHECK VALUE = 'A' (255);
DCL VAR &N INTEGER INIT 1000017 CHECK VALUE > 1000016;
```

- CHECK auf einen Vektor mit einer Meldung

```
DCL VAR &VEK (10) INT CHECK &VEK < 200 MESSAGE 'FEHLER AUFGETRETEN';
```

- CHECK auf eine Komponente einer Wiederholungsgruppe

```
DCL VAR 1 &W (10),
         2 W1 INT CHECK &W1 > 0,
         2 W2 INT;
```

  Für alle 10 Komponenten W1 gilt die CHECK-Klausel.

- CHECK auf eine Komponente einer Datengruppe

```
DCL VAR 1 &STR1,
          2 STR12 INT INIT 4 CHECK &STR1.STR12 > 0 OR &STR12 < -7;
```

Die CHECK-Klausel muß den Datentyp-Regeln und den gültigen Vergleichsbedingungen entsprechen. Es kann keine Prüfung erfolgen, wenn ein unzulässiger Datentyp geprüft wird, z.B.

```
DCL VAR &S CHAR(8) CHECK &S >10;          → Fehler „unzulässiger Datentyp“
DCL VAR &T NUM(8)  = 'DELETE';            → Fehler „unzulässiger Datentyp“
DCL VAR 1 &V(10),                         → Fehler „unzulässige Indexangabe“
          2 V1 INT CHECK &V(1).V1 =7;
```

Verträglichkeit zwischen CHECK und INIT

Die CHECK-Klausel muß verträglich sein mit der INIT-Klausel, d.h. der initialisierte Wert für eine Variable muß der Prüfbedingung von CHECK entsprechen. Wenn dies nicht der Fall ist, muß die explizite Angabe NOCHECK vorhanden sein (siehe DRIVE-Lexikon [3], Anweisung DECLARE VARIABLE und Metavariable *basistyp*).

```
DCL VAR &I INTEGER INIT –1 CHECK &I    0;          → Fehler: INIT-Klausel
widerspricht
DCL VAR &J CAHR(4) INIT 'JA' CHECK &J = 'NEIN';      der Prüfbedingung
```

Es ist möglich, daß die implizite Initialisierung eines Feldes der angegebenen CHECK-Klausel widerspricht, was zu einem Fehler führt, z.B.:

```
DCL VAR &PLZ NUM(5) CHECK &PLZ > 9999 AND &PLZ < 50001→ Fehler: Eine numerische
          MESSAGE 'PLZ: Nur 10000 bis 50000 erlaubt!';  Variable ist mit dem Wert 0
                                                          vorbelegt
```

Beispiel

Programm zum Löschen von Artikelsätzen. Es sollen nur Artikelsätze mit der Artikel-Nummer > A00100 gelöscht werden. Mit der CHECK-Klausel wird eine entsprechende Bereichsprüfung durchgeführt. Gibt der Programmierer eine falsche Nummer an, so erfolgt eine Fehlermeldung.

```
PROC ARTLOESCHEN;

/********** DEKLARATIONSTEIL **********/

DCL VAR &VGL CHAR(6) INIT 'A00200' CHECK &VGL > 'A00100'
                     MESSAGE 'ARTIKEL_NR MUSS > A00100 SEIN';
DCL VAR 1 &ARTIKEL,
          2 ARTNR CHAR(6),
          2 ARTTITEL CHAR(30);

DCL FORM AUSWAHLSCHIRM
    TTITLE NEWLINE1,
```

```
    ...
         ' ARTIKELSATZ LOESCHEN (ARTIKEL_NR EINGEBEN): ', RETURN &VGL, NL2,
    ...

/********** ABLAUFTEIL: UNTERPROGRAMM **********/
SUBPROC LOESCHEN;
S ARTIKEL_NR, TITEL INTO &ARTIKEL.* FROM ART.FIRMA WHERE ARTIKEL_NR = &VGL;
IF &DML_STATE = 'TABLE END'
   THEN SEND MESSAGE 'DATENSATZ NICHT VORHANDEN. BITTE DUE!';
END IF;
DELETE FROM ART.FIRMA WHERE ARTIKEL_NR = &VGL;
DISPLAY FORM
             NL 3, TAB 10, 'DER ARTIKELSATZ MIT DER ARTIKEL_NR ',
             NL  , TAB 20, &VGL ATTRIBUTE (SIGN),
             NL  , TAB 10, 'WURDE GELOESCHT.';
SET &VGL = ' ';
END SUBPROC;


/********** HAUPTTEIL DES ABLAUFTEILS **********/
WHENEVER ...
CYCLE;
  DISPLAY AUSWAHLSCHIRM;
  CALL LOESCHEN;
  ...
END CYCLE;
COMMIT WORK;
END PROC;
```

3.4 Konstanten definieren

Im Deklarationsteil eines Programms definieren Sie Konstanten mit der Anweisung DECLARE CONSTANT (siehe DRIVE-Lexikon [3], Anweisung DECLARE CONSTANT).

Bei der Definition einer Konstanten wird dieser ein Wert zugewiesen, der entweder direkt im Deklarationsteil steht oder zum Übersetzungszeitpunkt aus einer Datei gelesen und im Zwischencode gespeichert wird.

Sie können Konstanten im Programm nur dort verwenden, wo bisher Variablen zulässig waren. Wenn Sie Konstanten auch dort verwenden wollen, wo bisher Literale oder Integer-Angaben zulässig sind, müssen Sie die Precompilerfunktion `#define name wert` verwenden.

Den Konstanten können Sie als Wert ein *literal* oder eine eindeutige Meldung aus einer Meldungsdatei zuweisen.

Der Konstanten dürfen Sie während des Programmablaufs keinen Wert zuweisen.

Beispiele

Definition einer Konstanten im Deklarationsteil eines Programms:

```
DECLARE CONSTANT &KON1 'BITTE WARTEN SIE AUF DIE EINGABEAUFFORDERUNG';
```

Definition einer Konstanten, deren Wert zum Übersetzungszeitpunkt erzeugt und im Zwischencode gespeichert wird. Der Wert ist in diesem Fall ein Meldungstext, der aus einer Meldungsdatei heraus gelesen wird:

```
DECLARE CONSTANT &KON2 MSGSTRING (1 DRI);
```

3.5 Variablen Werte zuweisen

Einer Variablen können Sie mit der SET-Anweisung oder mit der INTO-Klausel Werte zuweisen. Auch mit der Angabe RETURN bei Parameterübergaben oder bei Bildschirmeingaben können Sie Variablen Werte zuweisen. Bei diesen Zuweisungen muß die Variable eindeutig angegeben werden. Den Variablen, die mit DECLARE SCREEN definiert wurden (SCREEN-Variablen), können zusätzlich Attribute zur Beschreibung der Bildschirmfelder zugewiesen werden (siehe „DRIVE-Lexikon“ [3], Anweisung SET).
Als Wert können Sie zuweisen:

mit SET oder RETURN:

– den Wert einer anderen Variablen

  ein Literal, es können auch leere Literale zugewiesen werden, d.h. eine Variable des Datentyps VARCHAR kann die Länge 0 erhalten.

  Beispiel:

```
SET &vc = ''
```

– einen Ausdruck (siehe DRIVE-Lexikon [3] , Metavariable *ausdruck*)
– Attribute (Eigenschaften) auf SCREEN-Variablen (FHS-Formate)

mit INTO:

  Werte von Satzelementen; darüber hinaus können Sie das aktuelle Datum und die aktuelle Zeit zuweisen:

```
SET &DATUM = CURRENT DATE;
```

Beim Zuweisen von Werten auf numerische Variablen werden Nachkommastellen kaufmännisch gerundet, wenn die Variablendeklaration nicht genügend Nachkommastellen zuläßt.

Mit den Funktionen CHAR und NUM sind heterogene Zuweisungen möglich (siehe Abschnitt „Stringfunktionen“ auf Seite 96).

Mit Hilfe von SET können Variablen Werte oder der NULL-Wert zugewiesen werden. Bei der Wertzuweisung muß der Datentyp des zugewiesenen Werts mit dem Datentyp der Variablen zuweisungsverträglich sein.

Wert kann sein: eine Variable, ein Ausdruck, ein Literal, ein Aggregat oder die Zahl $PI (siehe DRIVE-Lexikon [3], Metavariable *wert*).

```
SET &VAR = &PREIS * 2,5 / 100;
```

Einer strukturierten Variablen kann entweder ein strukturierter Wert oder ein einfacher Wert zugewiesen werden.

Wird einer strukturierten Variablen ein ebenfalls strukturierter Wert zugewiesen, erfolgt die Zuweisung komponentenweise.
Der zugewiesene Wert muß dieselbe Struktur haben wie die Variable, d.h., alle Komponenten müssen mit einem Wert versorgt werden. Dabei müssen die den einfachen Komponenten zugewiesenen Werte zuweisungsverträglich mit dem jeweiligen Datentyp sein.

```
DCL VAR 1 &A,
          2 STRASSE CHAR(29),
          2 PLZ  NUM(5),
          2 ORT CHAR(20);
SET &A = <'Residenzstr. 27',80333,'MUENCHEN'>;
```

Die Zuweisung erfolgt hier komponentenweise mit Hilfe eines Aggregats. Ein Aggregat definiert einen strukturierten Wert durch Angabe einer Liste von Werten. Der erste Wert dieser Liste wird der ersten einfachen Komponente (≡ STRASSE) zugeteilt, der zweite Wert der zweiten einfachen Komponente (≡ PLZ) zugeteilt, usw. Die Liste muß in spitze Klammern eingeschlossen sein; Kommata trennen die jeweiligen Werte. Die Anzahl der Werte innerhalb des Aggregats muß gleich der Anzahl der einfachen Komponenten der Variablen sein.

Alternativ können Sie den Komponenten auch einzeln Werte zuweisen:

```
SET &A.STRASSE = 'Residenzstr. 27' ,
    &A.PLZ = 80333 ,
    &A.ORT = 'Muenchen' ;
```

Sofern die Komponentennamen innerhalb des DRIVE-Programms eindeutig sind, können Sie die Komponenten auch ohne Qualifizierung ansprechen:

```
SET &STRASSE = 'Residenzstr. 27' ,
    &PLZ = 80333 ,
    &ORT = 'Muenchen' ;
```

Wird einer strukturierten Variablen nur ein einziger Wert zugewiesen, erhalten alle einfachen Komponenten diesen Wert, sofern alle denselben Datentyp besitzen (das ist bei Vektoren und Matrizen automatisch der Fall).

```
DCL VAR &SPRACHE(3) CHAR(4);
SET &SPRACHE = 'ENGL';
```

Kommen in einer strukturierten Variablen Redefinitionen vor, gilt die Zuweisung nur für die nicht mit REDEFINES definierten Komponenten.

```
DCL VAR 1 &NAME,
          2 VORNAME CHAR(15),
          2 SPITZNAME CHAR(15) REDEFINES &VORNAME,
          2 NACHNAME CHAR(20);
SET &NAME = <'Winston','Churchill'>;
```

Die Zuweisung gilt nicht für &NAME.SPITZNAME, aber da &VORNAME und &SPITZNAME denselben Speicherplatz belegen, hat auch &SPITZNAME anschließend den Wert 'Winston'.

Die Datentypen der Basisvariablen (redefinierte Variable) und der redefinierenden Variablen (... REDEFINES basisvariable) müssen nicht unbedingt übereinstimmen.
Allerdings kann es zu Konvertierungsfehlern kommen, wenn auf den zugewiesenen Wert über eine Basisvariable bzw. redefinierende Variable mit einem „falschen“ Datentyp zugegriffen wird.

```
DCL VAR 1 &NAME,
          2 VORNAME CHAR(15),
          2 SPITZNAME INTEGER  REDEFINES &VORNAME,
          2 NACHNAME CHAR(20);

SET &NAME.SPITZNAME = 4;        → richtig
SET &NAME = <4 ,'Churchill'>;   → falsch
```

Zuweisen von Werten im Rahmen von DRIVE- und SQL-Anweisungen

Im Rahmen von DRIVE- und SQL-Anweisungen werden Variablen Werte zugewiesen bei

- Rückgabe von Parametern an ein rufendes Programm
- Bildschirmeingaben
- Anweisungen mit INTO-Klausel

Rückgabe von Parametern an ein rufendes Programm

Variablen werden Werte zugewiesen, wenn sie in der USING-Klausel der CALL-Anweisung mit RETURN gekennzeichnet werden: Das gerufene Unterprogramm gibt Werte für diese Parameter an das rufende Programm zurück (siehe „DRIVE-Lexikon“ [3], Anweisung CALL).

Bildschirmeingaben

Werte für Variablen können am Bildschirm eingegeben werden, wenn die Variablen in der entsprechenden FILL-Anweisung mit RETURN als Eingabefelder gekennzeichnet werden (siehe „DRIVE-Lexikon“ [3], Anweisung FILL formatname).

Anweisungen mit INTO-Klausel

Bei folgenden Anweisungen werden Werte von Satzelementen aus der Datenbank den angegebenen Variablen zugewiesen (siehe DRIVE-Lexikon [3]).

– CYCLE ... INTO
– FETCH ... INTO
– INSERT ... RETURN INTO
– SELECT ... INTO

Zuweisen ungültiger Werte

Wird einer Variablen ein ungültiger Wert zugewiesen, d.h. es liegt ein Verstoß gegen die CHECK-Klausel vor, wird im Standardfall das Programm abgebrochen. Ein ungültiger Wert hinsichtlich des Datentyps führt zur Fehlermeldung CONVERSION ERROR.

Wenn Sie einen Programmabbruch verhindern wollen, müssen Sie einen Fehlerausgang definieren mit einer der folgenden Anweisungen:

– WHENEVER &ERROR IN ('CHECK ERROR') CONTINUE
– WHENEVER &ERROR IN ('CHECK ERROR') CALL subprogname
– WHENEVER &ERROR IN ('CONVERSION ERROR') CONTINUE
– WHENEVER &ERROR IN ('CONVERSION ERROR') CALL subprogname

In allen Fällen gilt:

– die fehlerhafte Zuweisung wird nicht ausgeführt
– der ursprüngliche Inhalt der Variablen bleibt bestehen
– DRIVE/WINDOWS versorgt die Systemvariable &ERROR_STATE: Die Komponente &ERROR erhält den Wert 'CHECK ERROR' oder 'CONVERSION ERROR', die Komponente &VAR_NAME wird mit dem Namen der Variablen versorgt, die den Fehler verursachte.

Beispiel 1

Einfachen Variablen Werte zuweisen

```
SET &VAR = &VAR2;
SET &VAR = '100'
```

Beispiel 2

Vektoren Werte zuweisen

```
SET &VEKT(1) = '20';
SET &FREMDSPR(3) = 'JA';
SET &FREMDSPR = <'N','N','JA'>;
```

Beispiel 3

Matrizen Werte zuweisen

```
DCL VAR &MATR(3,12) INT;
SET &MATR = '200';
```

Allen Feldern einer Matrix den gleichen Wert zuweisen:

```
SET &MATR(1,3) = 1000;
```

Einem bestimmten Feld einer Matrix einen Wert zuweisen:

```
SET &MATR( 1,12) = 14000
```

Einem bestimmten Bereich einer Matrix Werte zuweisen:

```
SET &MATR (1,1-4) = <100,800,600,1000>;
```

Einer Teilmatrix Werte zuweisen:

```
SET &MATR (1-2,11-12) = <1, &MATR(3,4), &MATR(2,6), 7>;
```

Beispiel 4

Komponenten von Gruppen Werte zuweisen

```
DCL VAR 1 &VAR_A,
          2 A2 (2),
            3 A3,
              4 A4 (5),
                5 A5           INT INIT 1,
                5 A6 (2),
                  6 A7         INT,
                  6 A8         INT,
              4 A9,
                5 A10,
                  6 A11 (10)   INT INIT 2,
                  6 A5         INT,
          2 A12 (3)            INT;
```

Für eine Wertzuweisung auf die einfache Komponente A8 mit dem Datentyp INTEGER müssen Sie folgende Namensqualifizierung vornehmen:

```
SET &VAR_A.A2(2).A3.A4(2).A6(1).A8 = 100;
```

Der Name A8 ist zwar eindeutig und auch nicht weiter strukturiert, muß aber hier doch qualifiziert angegeben werden. Der Grund ist, daß in einer höheren Stufe der Datengruppe mehrere Wiederholungsgruppen vorkommen. Für diese sind Indexangaben erforderlich.

Datums- und Zeitangaben

Sie können einer Variablen beliebige Datums- und Zeitangaben sowie das aktuelle Datum (CURRENT DATE), die aktuelle Uhrzeit (CURRENT TIME) oder den aktuellen Zeitstempel (CURRENT TIMESTAMP) zuweisen. Voraussetzung ist eine Variable mit dem entsprechenden Zeit-Datentyp. Bei DRIVE-Formaten ist eine Ausgabeaufbereitung (MASK) möglich.

Beispiele

Die Variable &datum wird mit dem Datum 1969-07-20 initialisiert, die Variable &zeit mit der Uhrzeit 22:30:10.

```
DECLARE VARIABLE &datum DATE INIT DATE (1969-07-20);
DECLARE VARIABLE &zeit  TIME INIT TIME (22:30:10);
```

Die folgenden Deklarationen sind falsch:

```
DECLARE VARIABLE &datum DATE INIT DATE (1955-02-29)
```

Fehler: Es gibt keinen 29.02.1955.

```
DECLARE VARIABLE &zeit  TIME INIT TIME (24:00:00)
```

Fehler: Mitternacht wird mit 00:00:00 angegeben.

Die folgenden Deklarationen haben eine Ausgabeaufbereitung:

```
DECLARE VARIABLE &DATUM DATE MASK 'Q(10)''␣DEN␣''ZD''.␣''R(10)YYYY';
SET &DATUM = CURRENT DATE;
```

ergibt: MONTAG␣␣␣␣␣DEN␣␣9.␣JANUAR␣␣␣␣1996

```
DECLARE VARIABLE &ZEIT TIME MASK 'HH''␣UHR␣''MI';
SET &ZEIT = CURRENT TIME;
```

ergibt: 11␣UHR␣55

Sie können einer Variable eine Zeitspanne zuweisen. Voraussetzung ist eine Variable mit dem Datentyp INTERVAL. Mit dem Datentyp INTERVAL wird eine Einheit (z.B. YEARS, DAYS) festgelegt.

Beispiele

Für die Variable &dauer vom Datentyp INTERVAL wird die Einheit „DAYS“ definiert.

```
DECLARE VARIABLE &dauer INTERVAL DAYS;
```

Der Variablen &tage vom Datentyp INTERVAL wird die Differenz zweier Datumsangaben zugewiesen. Diese Zeitspanne mit der Einheit DAYS wird umgerechnet in die Einheiten HOURS, MINUTES und SECONDS den Variablen &stunden, &minuten und &sekunden zugewiesen.

```
DECLARE VARIABLE &tage     INTERVAL DAYS;
DECLARE VARIABLE &stunden  INTERVAL HOURS;
DECLARE VARIABLE &minuten  INTERVAL MINUTES;
DECLARE VARIABLE &sekunden INTERVAL SECONDS;
...
SET &tage=&datum1 - &datum2;
SET &stunden=&tage;
SET &minuten=&tage;
SET &sekunden=&tage;
```

Sie können mit Datums- und Zeitangaben sowie mit Zeitspannen rechnen. Dabei können Sie die Funktionen CURRENT DATE, CURRENT TIME und CURRENT TIMESTAMP benutzen.

Beispiel

Den Variablen &tage und &sek vom Datentyp INTERVAL mit den Einheiten „DAYS“ und „SECONDS“ werden Zeitspannen zugewiesen, die sich mit Hilfe der aktuellen Angaben errechnen.

```
DECLARE VARIABLE &datum1 DATE,
                 &datum2 DATE,
                 &zeit1  TIME,
                 &zeit2  TIME,
                 &tage   INTERVAL DAYS,
                 &sek    INTERVAL SECONDS;
...
SET &datum1 = CURRENT DATE,
    &zeit1  = CURRENT TIME,
    &tage   = (&datum1 - &datum2) DAYS,
    &sek    = (&zeit1 - &zeit2) SECONDS;
```

3.5.1 Abkürzung „.*“

„.*“ ist eine abkürzende Schreibweise für eine Liste von Komponenten von Variablen, die auf der nächsten Stufe einer Gruppe liegen.

Beispiel

```
DCL C1 CURSOR FOR S ELEM1, ELEM2, ELEM3, ELEM4 FROM MITARBEITER;
DCL VAR 1 &CURVAR,
          2 PERS_NR     CHAR(06),
          2 NACHNAME    CHAR(20),
          2 VORNAME     CHAR(20),
          2 ADRESSE,
            3 LAND      CHAR(03),
            3 STRASSE   CHAR(26),
            3 PLZ       CHAR(10),
            3 ORT       CHAR(20);

FETCH C1 INTO &CURVAR.*;
```

In den Cursor C1 werden 4 Satzelemente aus der Basistabelle MITARBEITER gelegt. Um auf diese Elemente zugreifen zu können, muß eine Variable angelegt werden, die dem Cursoraufbau entspricht (= &CURVAR). Mit jedem FETCH wird ein Satz aus der Datenbank in der Variablen &CURVAR abgelegt. In diesem Beispiel sollen alle Elemente pro Satz in die Variable gelegt werden. „.*“ hat die gleiche Wirkung wie die Angabe jedes einzelnen Elements.

Verhalten bei SET

„.*“ ist nur auf der rechten Seite bei der SET-Zuweisung erlaubt.

Beispiel

```
SET &MITARB =
<'P00009','VON HAALEN','WALTER','USA','30 THIRD AVENUE','NY 11217','NEW YORK'>;
```

Diese lange Liste brauchen Sie nicht unbedingt anzugeben, es genügt:

```
SET &MITARB = <&CURVAR.*>;
```

Einsatzbereich von „.*“

Die Abkürzung „.*“ kann eingesetzt werden bei
– CALL (nicht bei RETURN)
– DISPLAY (nicht bei RETURN)
– DO
– FETCH
– FILL (nicht bei RETURN)
– INSERT
– SELECT
– SEND MESSAGE
– SET (nur rechte Seite)

Attribute an SCREEN-Variablen zuweisen

SCREEN-Variablen können mittels SET Werte und Attribute zugewiesen werden.

Ist die Variable strukturiert, beziehen sich die Feldattribute nur auf die einfachen Komponenten. Eine Liste der möglichen Attribute finden Sie im „DRIVE-Lexikon“ [3], Anweisung SET.

Sie können einer SCREEN-Variablen bzw. einem Datenfeld einer SCREEN-Variablen auch mit der gleichen SET-Anweisung einen Wert und Attribute zuweisen.

Bei Attributen unterscheidet man zwei Arten:

– Feldattribute kennzeichnen Eigenschaften von speziellen Feldern eines FHS-Formats.

– Globalattribute kennzeichnen globale Eigenschaften eines FHS-Formats.

Die zugewiesenen Attribute dürfen sich nicht gegenseitig ausschließen (z.B. dürfen nicht zwei Farben gleichzeitig zugewiesen werden). Verschiedene Feldattribute dürfen in einer Liste kombiniert werden.

Wie Sie in DRIVE/WINDOWS mittels SCREEN-Variablen FHS-Formate einsetzen können, ist ausführlich in Abschnitt „FHS-Formate einsetzen“ auf Seite 143 beschrieben.
Die folgenden Abschnitte stellen nur kurz die jeweilige Form der SET-Anweisung vor.

Feldattribute zuweisen

Wenn Sie in Ihrem Programm eine SCREEN-Variable mit DECLARE SCREEN vereinbaren, fügt DRIVE/WINDOWS intern ein DECLARE VAR in das Programm ein. Die einzelnen Felder des entsprechenden FHS-Formats werden dabei als Komponenten der SCREEN-Variablen dargestellt.

Die eingefügte Deklaration können Sie sich in der Protokolldatei ansehen, wenn Sie Ihr Programm mit der DRIVE-Anweisung COMPILE und OPTION LISTING=LIB/...übersetzt haben: das eingefügte DECLARE VAR steht direkt hinter dem entsprechenden DECLARE SCREEN.

Bei der Zuweisung von Feldattributen können Sie sich auf die in dem DECLARE VAR verwendeten Namen beziehen.

```
SET screen_variable.feldname1 WITH ATTRIBUTE (MUST);
SET screen_variable.feldname2 WITH ATTRIBUTE (BLUE, CURSOR);
```

Globalattribute zuweisen

Mit Hilfe von SET ... ATTRIBUTE ... können SCREEN-Variablen Globalattribute zugewiesen werden.

```
SET screenformat WITH ATTRIBUTE (ALARM, NOINIT);
```

Feldattribute zuweisen bei fehlerhaften FHS-Formaten

Mit Hilfe von SET ... ERRORATTRIBUTE ... können Feldern mit fehlerhaftem Datenwert Feldattribute, die das fehlerhafte Feld kennzeichnen, zugewiesen werden.

Alle Felder eines FHS-Formats mit EDIT_STATE ≠ 'V' (VALID) erhalten die angegebenen Feldattribute.

```
SET screenformat WITH ERRORATTRIBUTE (INVERSE);
```

Diese Anweisung bewirkt, daß alle Felder eines FHS-Formats, die der Anwender falsch versorgt hat, invers ausgegeben werden.

Hinweis auf CLEAR

Die folgende SET-Anweisung entspricht einer CLEAR-Anweisung:

```
SET &VAR = 'initwert' NOCHECK CLEAR &VAR
```

3.6 Zuweisungsverträglichkeit bei der Datenkonvertierung

In der folgenden Tabelle sind die zulässigen Zuweisungen bzw. Vergleiche von DRIVE-Variablen aufgeführt. Dabei wird zwischen zuweisungsverträglich (z.B. SET &A=&B) und vergleichbar (IF &A=&B) unterschieden.

Die Regeln gelten für DRIVE-Anweisungen. Für SQL-Anweisungen gelten die Regeln und Konventionen der jeweiligen Datenbanksysteme.

Durch bestimmte Funktionen und Rechenoperationen kann implizit der Datentyp geändert werden:

`CHARACTER(&NUMVARIABLE)` → Datentyp CHARACTER

`(&DATUM1-&DATUM2)` → Datentyp INTERVAL

Folgende Datenkonvertierungen sind in DRIVE/WINDOWS möglich:

von	**nach**												
	CHAR	**DEC**	**NUM**	**INT**	**SMINT**	**DATE**	**TIME**	**TIME(3)**	**TIME-STAMP(3)**	**IV**	**REAL**	**VAR-CHAR**	**DOUBLE PRECI-SION**
CHAR	1	2	2	2	2	3	4	4	5	–	2	7	2
DEC	8	9	9	10	10	–	–	–	–	14	11	12	11
NUM	8	9	9	10	10	–	–	–	–	14	11	12	11
INT	8	9	9	8	13	–	–	–	–	14	9	12	9
SMINT	8	9	9	9	+	–	–	–	–	14	9	12	9
DATE	6	–	–	–	–	+	–	–	19	–	–	6	–
TIME	6	–	–	–	–	–	+	19	19	–	–	6	–
TIME(3)	6	–	–	–	–	–	19	+	19	–	–	6	–
TIME-STAMP(3)	6	–	–	–	–	19	19	19	+	–	–	6	–
IV	–	15	15	15	15	–	–	–	–	18	15	–	15
REAL	16	11	11	10	10	–	–	–	–	14	+	16	9
VAR-CHAR	1	2	2	2	2	3	4	4	5	–	2	17	2
FLOAT	16	9	9	10	10	–	–	–	–	14	9	16	+

Erläuterung:

– Die Datenkonvertierung ist nicht zuweisungsverträglich und vergleichbar.

\+ Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

1 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.
Wenn das Ursprungsfeld länger als das Zielfeld ist, dann wird rechts abgeschnitten, falls die abgeschnittenen Zeichen Leerzeichen sind. Andernfalls wird 'CONVERSION ERROR' zurückgemeldet.
Wenn das Zielfeld länger als das Ursprungsfeld ist, wird nur bei der Konvertierung vom Datentyp CHAR rechts mit Leerzeichen aufgefüllt.

2 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion NUMERIC.
Die Zeichenreihe wird bis zum ersten nicht erkennbaren Zeichen gelesen und in den entsprechenden Datentyp umgewandelt.

CHAR → SMINT
CHAR → INT
Die Zeichendarstellung der Zahl kann aufgebaut werden als eine Folge von Zwischenraum-Zeichen, Vorzeichen und eine Ziffernfolge.

CHAR → DEC
CHAR → NUM
CHAR → FLOAT
CHAR → DOUBLE PRECISION
Die Zeichendarstellung der Zahl kann aufgebaut werden als eine Folge von Zwischenraum-Zeichen, Vorzeichen, eine Ziffernfolge mit oder ohne Dezimalpunkt. Auch ein Exponent (e oder E) mit einem Vorzeichen kann enthalten sein.

3 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion DATE.

CHAR → DATE
Bei der Konvertierung nach DATE muß das CHAR-Feld *jahr-monat-tag* (YYYY-MO-DD) enthalten. Außerdem müssen die Dateninhalte den Datumsregeln entsprechen.

4 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion TIME.

CHAR → TIME
Bei der Konvertierung nach TIME muß das CHAR-Feld *stunde*:*minute*:*sekunde* (HH:MI:SS) enthalten. Außerdem müssen die Dateninhalte den Zeitregeln entsprechen.

CHAR → TIME(3)
Bei der Konvertierung nach TIME muß das CHAR-Feld *stunde*:*minute*:*sekunde*.*bruchteil* (HH:MI:SS.FFF) enthalten. Außerdem müssen die Dateninhalte den Zeitregeln entsprechen.

5 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion TIMESTAMP.

CHAR → TIMESTAMP(3)
Bei der Konvertierung nach TIMESTAMP(3) muß das CHAR-Feld *jahr-monat-tag␣stunde*:*minute*:*sekunde.bruchteil* (YYYY-MO-DD␣HH:MI:SS.FFF) enthalten. Außerdem müssen die Dateninhalte den Zeitregeln entsprechen.

6 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion CHARACTER.

DATE → CHAR
TIME → CHAR
TIME(3) → CHAR
TIMESTAMP(3) → CHAR
DATE → VARCHAR
TIME → VARCHAR
TIME(3) → VARCHAR
TIMESTAMP(3) → VARCHAR

Abhängig vom Typ des Ursprungsfelds enthält das CHAR- oder VARCHAR-Feld nach der Konvertierung folgende Zeitwerte:

DATE:	*jahr-monat-tag* (YYYY-MO-DD)
TIME:	*stunde*:*minute*:*sekunde* (HH:MI:SS)
TIME(3):	*stunde*:*minute*:*sekunde.bruchteil* (HH:MI:SS.FFF)
TIMESTAMP(3):	*jahr-monat-tag␣stunde*:*minute*:*sekunde.bruchteil* (YYYY-MO-DD␣HH:MI:SS.FFF)

Bei der Konvertierung nach CHAR gilt:
Ist die Ziellänge größer als die Quellenlänge, wird mit Leerzeichen aufgefüllt. Im umgekehrten Fall wird die Konvertierung mit einem Syntaxfehler abgebrochen.

Bei der Konvertierung nach VARCHAR gilt:
Nach der Konvertierung enthält das VARCHAR-Feld einen Zeitwert. Die Ziellänge ist die aktuelle VARCHAR-Länge, die auch nicht mit Leerzeichen aufgefüllt wird. Ist die Ziellänge zu klein, wird die Konvertierung mit einem Syntaxfehler abgebrochen.

7 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

Wenn das Ursprungsfeld länger als die maximale Länge des Zielfelds vom Typ VARCHAR ist, dann wird in dieser maximalen Länge übertragen.

8 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion CHARACTER.

Wenn das Ursprungsfeld kürzer als das Zielfeld ist, dann wird rechts mit Leerzeichen aufgefüllt. Im umgekehrten Fall wird ein CONVERSION ERROR gemeldet.

9 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

Es dürfen keine führenden signifikanten Stellen verlorengehen.

10 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

Ist der Ursprungswert größer, als das Zielfeld erlaubt, dann wird ein CONVERSION ERROR gemeldet. Falls Nachkommastellen abgeschnitten wurden, wird kein Fehler gemeldet.

11 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

Diese Konvertierungen werden von den Funktionen „dectodbl“ bzw. „deccvdbl“ durchgeführt. Es werden explizit keine Wertebereichsüberprüfungen vorgenommen.

12 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion CHARACTER.

Der Ablauf ist der gleiche wie beim Zielfeld vom Typ CHAR, es wird links nicht mit Leerzeichen aufgefüllt.

13 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

Ist der Ursprungswert größer als das Zielfeld erlaubt, dann wird ein CONVERSION ERROR gemeldet.

14 Die Datenkonvertierung ist nur zuweisungsverträglich, aber nicht vergleichbar.

Die Zielvariable darf im Sinne von INTERVAL nur eindimensional sein (z.B. SECOND to SECOND, siehe DRIVE-Lexikon [3], Metavariable *datumzeiteinheit*). Der Ursprungswert darf außerdem nicht größer sein als das Zielfeld erlaubt.

15 Die Datenkonvertierung ist nur zuweisungsverträglich, aber nicht vergleichbar.

Die Ursprungsvariable darf im Sinne von INTERVAL nur eindimensional sein (z.B. SECOND to SECOND, siehe DRIVE-Lexikon [3], Metavariable *datumzeiteinheit*). Der Ursprungswert darf außerdem nicht größer sein als das Zielfeld erlaubt.

16 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar nur mit Hilfe der Konvertierungsfunktion CHARACTER.

Die Ausgabe erfolgt im Format: [–]Ziffer.ZiffernE[+ | –]Ziffern

Das Argument wird mit einer Ziffer vor dem Dezimalpunkt und einer Anzahl von Ziffern nach dem Dezimalpunkt (Genauigkeitsangabe) ausgegeben.

Wenn das Ursprungsfeld kürzer als das Zielfeld ist, dann wird rechts mit Leerzeichen aufgefüllt. Dies gilt nur bei der Konvertierung vom Datentyp CHAR, nicht aber bei VARCHAR.

17 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

Als Länge des Ursprungsfeldes wird die Länge des Typs VARCHAR genommen. Die Länge des Zielfeldes, die angegeben werden muß, ist die maximale VARCHAR-Länge des Ziels. Sollte diese kleiner sein als die Zeichenkette des Ursprungsfeldes, dann wird rechtsbündig abgeschnitten.

18 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar.

Es werden nur eindimensionale DRIVE-Intervalle (z.B. SECOND to SECOND, siehe DRIVE-Lexikon [3], Metavariable *datumzeiteinheit*) konvertiert. Bei der Konvertierung von INTERVAL muß die Intervalleinheit beachtet werden. Welche Intervalleinheiten vergleichbar sind und wie sie umgerechnet werden, finden Sie im Abschnitt „Intervalleinheiten umrechnen“ auf Seite 80.

Bei Zuweisungen auf Variablen vom Typ INTERVAL wird automatisch in die Einheit der Zielvariablen konvertiert.

19 Die Datenkonvertierung ist zuweisungsverträglich und vergleichbar. DRIVE/WINDOWS führt Vergleiche durch, indem zunächst nach TIMESTAMP(3) konvertiert und dann chronologisch verglichen wird.

DATE → TIMESTAMP(3)
Bei der Konvertierung wird die Uhrzeit *stunde*:*minute*:*sekunde.bruchteil* mit Nullen aufgefüllt (00:00:00.000).

TIME → TIME(3)
Bei der Konvertierung werden die Sekundenbruchteile mit Nullen aufgefüllt (HH:MI:SS.000).

TIME → TIMESTAMP(3)
Bei der Konvertierung wird das aktuelle Datum *jahr-monat-tag* (YYYY-MO-DD) ergänzt und die Sekundenbruchteile mit Nullen aufgefüllt (HH:MI:SS.000).

TIME(3) → TIME
Bei der Konvertierung werden die Sekundenbruchteile (FFF) abgeschnitten.

TIME(3) → TIMESTAMP(3)
Bei der Konvertierung wird das aktuelle Datum *jahr-monat-tag* (YYYY-MO-DD) ergänzt.

TIMESTAMP(3) → DATE
Bei der Konvertierung wird *stunde*:*minute*:*sekunde.bruchteil* (HH:MI:SS.FFF) abgeschnitten.

TIMESTAMP(3) → TIME
Bei der Konvertierung werden *jahr-monat-tag* (YYYY-MO-DD) und die Sekundenbruchteile (FFF) abgeschnittten.

TIMESTAMP(3) → TIME(3)
Bei der Konvertierung wird *jahr-monat-tag* (YYYY-MO-DD) abgeschnittten.

Intervalleinheiten umrechnen

Die folgende Tabelle zeigt, welche Intervalleinheiten in andere umgerechnet werden können. Das Ergebnis ist eine Ganzzahl, Nachkommastellen werden abgeschnitten.

von	nach						
	YEARS	MONTHS	DAYS	HOURS	MINUTES	SECONDS	FRAC-TIONS
YEARS	+	+	–	–	–	–	–
MONTHS	+	+	–	–	–	–	–
DAYS	–	–	+	+	+	+	+
HOURS	–	–	+	+	+	+	+
MINUTES	–	–	+	+	+	+	+
SECONDS	–	–	+	+	+	+	+
FRACTIONS	–	–	+	+	+	+	+

+: Die Intervalleinheit kann umgerechnet werden.

–: Die Intervalleinheit kann nicht umgerechnet werden.

Intervalleinheiten werden in Einzelschritten solange in die nächstgrößere oder nächstkleinere Einheit umgerechnet bis die Zieleinheit erreicht ist. Dabei gelten folgende Regeln:

1 YEARS = 12 MONTHS

1 DAYS = 24 HOURS
1 HOURS = 60 MINUTES
1 MINUTES = 60 SECONDS
1 SECONDS = 1000 FRACTIONS

DRIVE/WINDOWS führt Berechnungen mit den Intervallleinheiten YEARS und MONTHS durch, indem die Einheiten YEARS nach MONTHS umgerechnet werden. Bei Vergleichen werden die Ergebnisse numerisch verglichen.

DRIVE/WINDOWS führt Berechnungen mit den Intervallleinheiten DAYS, HOURS, MINUTES und SECONDS durch, indem die Einheiten DAYS, HOURS und MINUTES nach SECONDS umgerechnet werden. Bei Vergleichen werden die Ergebnisse numerisch verglichen.

DRIVE/WINDOWS führt Berechnungen mit den Intervallleinheiten DAYS, HOURS, MINUTES, SECONDS und FRACTIONS durch, indem die Einheiten DAYS, HOURS, MINUTES und SECONDS nach FRACTIONS umgerechnet werden. Bei Vergleichen werden die Ergebnisse numerisch verglichen.

4 Programmierlogik

Dieses Kapitel beschreibt, wie Sie:

- Bedingungen, bedingte Verzweigungen und Schleifen programmieren (IF, CASE, CYCLE, ab Seite 82)
- Fehler abfangen und Endekriterien festlegen (WHENEVER, ab Seite 88)
- Meldungen sprachunabhängig lesen (MSGSTRING, ab Seite 91)
- Folgen von Anweisungen als COPY-Elemente in Programme einfügen (COPY, Seite 92)
- Stringfunktionen und numerische Funktionen einsetzen (ab Seite 92)
- Anweisungen dynamisch ausführen lassen (EXECUTE, (ab Seite 101)
- interne Unterprogramme aufrufen (SUBPROCEDURE, Seite 109)
- externe DRIVE-Programme aufrufen (CALL, Seite 110)
- externe Programme aufrufen (CALL MODULE, ab Seite 110)

4.1 Anweisungen zur Ablaufsteuerung

Programme können Bedingungen, bedingte Verzweigungen und Schleifen enthalten. Sie dürfen beliebig geschachtelt werden, sich aber nicht überlappen.

Das Programmieren von Bedingungen, bedingten Verzweigungen und Schleifen ist nur im Programm-Modus erlaubt.

4.1.1 Bedingungen programmieren (IF)

IF-Strukturen bieten die Möglichkeit, abhängig vom Wahrheitswert einer Bedingung alternative Programmteile zu durchlaufen.

IF bedingung = 'TRUE'	
JA	NEIN
THEN-Zweig	ELSE-Zweig

Liefert die Bedingung den Wahrheitswert „TRUE“, wird der THEN-Zweig durchlaufen, ansonsten der ELSE-Zweig.

Die erste Anweisung einer IF-Struktur muß IF und die letzte END IF sein. Abhängig vom Wahrheitswert der Bedingung entscheidet sich, welcher Zweig durchlaufen wird.

```
IF bedingung THEN
     anweisung;          }
     .                   } THEN-Zweig
     .                   }
ELSE
   anweisung;            }
   .                     } ELSE-Zweig
   .                     }
 END IF;
```

Wenn der ELSE-Zweig leer ist, kann auch die Anweisung ELSE fehlen. Im Falle „bedingung <> TRUE“, d.h. „FALSE“ oder „UNKNOWN“, wird das Programm dann hinter der entsprechenden END IF-Anweisung fortgesetzt.

IF-Strukturen dürfen geschachtelt werden, d. h., sowohl der THEN-Zweig als auch der ELSE-Zweig dürfen wiederum IF-Strukturen enthalten.

Schachtelung von IF-Bedingungen und Schleifen

Sie können IF-Bedingungen und Schleifen schachteln. Beachten Sie, daß eine Schleife innerhalb einer IF-Abfrage mit END CYCLE abgeschlossen sein muß, bevor die IF-Abfrage beendet ist. Umgekehrt gilt das gleiche: Eine IF-Abfrage innerhalb einer Schleife (CYCLE) muß mit END IF abgeschlossen werden, bevor die Schleife mit END CYCLE beendet ist.

Beispiel

Die folgende Schleife mit IF-Strukturen ist ein Ausschnitt aus einem Programm.
Das Programm bietet einen Menübildschirm an, in dem der Anwender sich Mitarbeiter anzeigen lassen kann, ausgewählt nach Herkunftsland und Einkommen.
Der Menübildschirm bietet für Herkunftsland und Einkommen Vorschläge an, die der Anwender durch Ankreuzen auswählen kann. In der folgenden Schleife wird für diese Auswahlfelder mittels IF-Anweisungen nacheinander abgefragt, ob sie leer sind oder ausgewählt wurden. Wurde ein Feld ausgewählt, wird die Variable für die Datenbankabfrage entsprechend versorgt.

```
CYCLE WHILE &INDEX < 5;
  IF &L(&INDEX) <> ' ' THEN
       SET &HLAND = &H1(&INDEX);
  END IF;
  IF &G(&INDEX) <> ' ' THEN
       SET &G_UNTER = &INDEX * 2000;
       SET &G_OBER = &G_UNTER +2000;
  END IF;
  SET &INDEX = &INDEX + 1;
END CYCLE;
```

4.1.2 Bedingte Verzweigungen programmieren (CASE)

Mit der Anweisung CASE legen Sie bedingte Verzweigungen fest. DRIVE/WINDOWS vergleicht Werte mit vorgegebenen Mustern und verzweigt zu weiteren Anweisungen in einem OF-Zweig, wenn der Vergleich den Wahrheitswert TRUE ergibt (siehe DRIVE-Lexikon [3], Anweisung CASE). Der OF-Zweig ist eine Anweisungsfolge zwischen zwei OF-Anweisungen.

Der Vergleich ergibt sich aus:

– der in einem OF-Zweig formulierten Bedingung oder

– einer Bedingung, die sich aus einem Ausdruck bei OF, zusammen mit dem bei CASE formulierten Ausdruck zu einer Bedingung mit Gleichheitszeichen zusammensetzt.

Wenn Sie CASE ALL angeben, durchläuft DRIVE/WINDOWS alle OF-Zweige mit dem Vergleichswert TRUE. Wenn Sie den Operand ALL nicht angeben, wird nur der erste OF-Zweig mit dem Vergleichswert TRUE durchlaufen und dann nach END CASE verzweigt.

Beachten Sie, daß die erste Anweisung nach CASE die Anweisung OF sein muß.

Ein CASE-Block muß immer mit END CASE abgeschlossen werden.

Schachtelung von bedingten Verzweigungen

Anweisungen mit CASE dürfen beliebig oft geschachtelt werden, d.h. in den OF-Zweigen darf wieder CASE ... END CASE vorkommen. CASE, IF und CYCLE dürfen geschachtelt werden, sich aber nicht überlappen.

Beispiel 1

```
...
CASE &VAR1 - &VAR2;

     OF 0 SET &VAR3 = 'TRUE';
     OF 1 SET &VAR3 = 'FALSE';
     OF REST SET &VAR3 = 'UNKNOWN';

END CASE;
...
```

Beispiel 2

```
...
CASE ALL;

          OF &VAR1 > &VAR2 SET &VAR3 = 'TRUE';
          OF &VAR1 < &VAR2 SET &VAR3 = 'FALSE';
          OF REST SET &VAR3 = 'UNKNOWN';

END CASE;
...
```

Beispiel 3

```
...
CYCLE FOR &INDEX = 1 TO 4;
      CASE ALL;
           OF &L(&INDEX) <> ' ';
                SET &HLAND &H1(&INDEX);
           OF &G(&INDEX) <> ' ';
                SET &G_UNTER = &INDEX * 2000;
                SET &G_OBER = &G_UNTER + 2000;
      END CASE;
END CYCLE;
...
```

4.1.3 Schleifen programmieren (CYCLE)

Unter einer Schleife versteht man eine Folge von Anweisungen, die beliebig oft durchlaufen wird.
Die erste Anweisung einer Schleife muß CYCLE und die letzte END CYCLE sein. Die Anweisungen dazwischen bilden den Verarbeitungsteil der Schleife.

Mit der Anweisung CONTINUE CYCLE kann vorzeitig zum Ende der Schleife gesprungen werden. In Abhängigkeit von der Schleifenbedingung wird die Schleife erneut durchlaufen oder abgebrochen.

Mit der Anweisung BREAK CYCLE kann die Schleife vorzeitig beendet werden.

Nach Beendigung der Schleife wird das Programm mit der Anweisung fortgesetzt, die dem END CYCLE dieser Schleife folgt.

```
CYCLE ...;
.
.          } Verarbeitungsteil
.
END CYCLE;
```

Man unterscheidet vier Arten von Schleifen:

– CYCLE-Schleifen
– Cursorschleifen
– WHILE-Schleifen
– FOR-Schleifen

Alle Schleifenarten dürfen beliebig geschachtelt werden. Zwei Einschränkungen gibt es. Innerhalb einer Cursorschleife darf keine weitere Cursorschleife mit demselben Cursor vorkommen. Innerhalb einer FOR-Schleife darf keine weitere FOR-Schleife mit derselben Laufvariablen vorkommen.

CYCLE-Schleifen

Wenn CYCLE ohne Operand angegeben wird, spricht man von einer Endlosschleife.

```
CYCLE;
...
IF bedingung THEN BREAK CYCLE;
...
END CYCLE;
```

Die Endebedingung für diese Schleifenart wird im Verarbeitungsteil abgefragt und die Schleife wird nur durch die Anweisung BREAK CYCLE abgebrochen.
Fehlt die Anweisung BREAK CYCLE in der Schleife, können Sie die Schleife nur beenden, indem Sie das Programm abbrechen.

Cursorschleifen

Zu Beginn jeder Schleife wird ein Satz aus der Cursortabelle in die angegebene Variable übertragen. Solange Einträge in der Cursortabelle vorhanden sind, wird die Schleife durchlaufen. Enthält die Cursortabelle keinen Eintrag mehr (nach dem letzten Treffer oder bei leerer Treffermenge), wird die Schleife beendet.

```
DECLARE cursorname CURSOR FOR select-ausdruck;
DECLARE VAR &satz LIKE CURSOR cursorname;
...
CYCLE cursorname INTO &satz.* ;
...
END CYCLE;
```

Der Verarbeitungsteil dieser Schleife darf keine Cursorschleife enthalten, die sich auf denselben Cursor bezieht.

WHILE-Schleifen

Zu Beginn jeder Schleife wird überprüft, ob die angegebene Bedingung noch erfüllt ist. Solange die Überprüfung den Wert „TRUE“ liefert, wird die Schleife durchlaufen.

```
CYCLE WHILE bedingung;
...
END CYCLE;
```

FOR-Schleifen

Zu Beginn der Schleifenverarbeitung wird einer Variablen ein Wert zugewiesen und überprüft, ob dieser Wert kleiner oder gleich dem Endewert ist. Wenn das der Fall ist, dann wird die Anweisungsfolge zwischen CYCLE FOR und END CYCLE abgearbeitet. Andernfalls wird die Schleife über END CYCLE beendet. Die vorgegebene Schrittweite ist 1. Sie können aber auch eine eigene Schrittweite angegeben. Eine negative Schrittweite bedeutet, daß die Überprüfung mit dem Endewert auf größer oder gleich erfolgt.

```
...
CYCLE FOR &INDEX = 1 TO 4
CASE ALL
OF &L(&INDEX) <> ' ';
SET &HLAND &H1(&INDEX);
OF &G(&INDEX) <> ' ';
SET &G_UNTER = &INDEX * 2000;
SET &G_OBER = &G_UNTER + 2000;
END CASE;
END CYCLE;
...
```

4.2 Fehler abfangen, Endekriterien

Tritt in einem Programm ein Ablauffehler auf, wird das Programm im Standardfall abgebrochen.
Wenn Sie einen solchen Programmabbruch verhindern wollen, müssen Sie mit der Anweisung WHENEVER einen Fehlerausgang definieren. Sie haben auch die Möglichkeit, das Betätigen einer K/F-Taste als „Fehler“ zu betrachten und dafür einen Fehlerausgang zu definieren (siehe „DRIVE-Lexikon“ [3], Anweisung WHENEVER).

Bei Ablauffehlern in Programmen unterscheidet man zwei Fehlerklassen:

ERROR (DRIVE-Fehlermeldungen)

DML_STATE (Datenbank-Fehlermeldungen)

Je nachdem, ob Sie Fehler der Klasse ERROR oder der Klasse DML_STATE abfangen wollen, müssen Sie eine der folgenden Varianten der WHENEVER-Anweisung verwenden:

– WHENEVER &ERROR IN (error, ...) CONTINUE
– WHENEVER &DML_STATE IN (status, ...) CONTINUE
– WHENEVER &ERROR IN (error, ...) BREAK
– WHENEVER &DML_STATE IN (status, ...) BREAK
– WHENEVER &ERROR IN (error, ...) CALL subprogname
– WHENEVER &DML_STATE IN (status, ...) CALL subprogname

Sie können mit WHENEVER alle Einträge der &ERROR- und &DML_STATE-Systemvariablen abfragen (siehe Abschnitt „Systemvariablen“ auf Seite 20).

Beispiel

```
WHENEVER &ERROR IN ('INDEX ERROR')
   CALL fehlerbearb;
```

Reaktionen bei den Fehlerklassen ERROR und DML_STATE

Werden die Anweisungen fehlerfrei ausgeführt, erhalten die Felder &ERROR bzw. &DML_STATE den Wert 'OK' zugewiesen. Für SESAM/SQL V2 erhält &SQL_STATE den Wert 00000 zugewiesen.

Tritt ein Ablauffehler auf, der zu den bei WHENEVER beschriebenen Fehlerklassen gehört, dann kommt es zu unterschiedlichen Aktionen:

– Die Ausführung der aktuellen Anweisung wird in jedem Fall abgebrochen. Geänderte Variablenwerte sind dann nicht mehr gültig, die Variablen erhalten ihre alten Werte.

Eine Ausnahme bildet die Anweisung DISPLAY SCREEN, bei der die Datenwerte aus technischen Gründen teilformatspezifisch und nicht anweisungsspezifisch konsistent gehalten werden.

– Ist zum aufgetretenen Fehler keine Aktion definiert, wird das Programm, außer bei „TABLE END“, „DIRTY READ“, „OK END“, „TOO LONG“ und „TOO SHORT“abgebrochen. Das Verhalten ist so, als wäre die Aktion BREAK angegeben.

– Ist zum aufgetretenen Fehler die Aktion CONTINUE definiert, wird das Programm hinter der fehlerhaften Anweisung fortgeführt.

– Ist zum aufgetretenen Fehler die Aktion CALL *subprogname* definiert, wird zunächst das interne Unterprogramm *subprogname* aufgerufen und das rufende Programm danach hinter der fehlerhaften Anweisung fortgesetzt.

  Tritt allerdings bei der Abarbeitung des Unterprogramms *subprogname* ein weiterer Fehler auf, wird das Programm unabhängig von definierten Fehlerausgängen abgebrochen. Beim Ausführen der SYSTEM-Anweisung wird in jedem Fall, d.h. bei fehlerfreiem Ausführen und im Fehlerfall, die entsprechende Systemvariable mit dem Returncode des gerufenen Subsystems versorgt (siehe „DRIVE-Lexikon“ [3], Anweisung WHENEVER).

  Bei der Abarbeitung des internen Unterprogramms *subprogname* wird die Systemvariable &ERROR_STATE nicht mehr verändert, so daß die darin enthaltene Fehlerinformation in diesem Unterprogramm abgefragt werden kann. Auch wenn in *subprogname* ein externes Unterprogramm aufgerufen wird, bleibt der Inhalt von &ERROR_STATE erhalten.

– Wenn das Datenbanksystem SESAM oder UDS eine Transaktion intern zurücksetzt (SQL-Code < -1000), kann das DRIVE-Programm nicht auf den SQL-Code reagieren, der das interne Rücksetzen der Transaktion ausgelöst hat. In diesem Fall können Sie also keinen Fehlerausgang mit WHENEVER programmieren.

4.2.1 Reaktionen bei WHENEVER &KFKEY

Mit der Anweisung WHENEVER &KFKEY können Sie in DRIVE-Programmen auf die Betätigung der K/F-Tasten reagieren. Dabei müssen Sie folgendes beachten:

– In der DRIVE-Systemvariablen &ERROR_STATE werden folgende Komponenten versorgt:

  – FORMAT_NAME (gegebenenfalls) LINE_NR DISPLACEMENT STATEMENT

  – D.h., daß im Gegensatz zu allen anderen &KFKEY-Fällen keine Fehlerinformation in &ERROR abgelegt wird, weil die DRIVE-Systemvariable &KFKEY immer mit dem Wert der aktuellen K- oder F-Taste versorgt wird.

– Bei einer FILL/DISPLAY-Anweisung mit Überlauf wird sofort WHENEVER &KFKEY ausgeführt. Folgebildschirme bei DISPLAY werden nicht mehr ausgegeben. Als nächste Anweisung wird entweder CONTINUE, CALL oder BREAK ausgeführt. Nach der WHENEVER-Behandlung werden die nach DISPLAY folgenden Anweisungen ausgeführt.

– Die Fehlerbehandlung bei der Überprüfung einer Dateneingabe mit der CHECK-Klausel hat eine höhere Priorität als die Aktionen im Falle einer WHENEVER &KFKEY-Behandlung.

– In der Systemvariablen &KFKEY steht die bei der letzten Eingabe verwendete K- bzw. F-Taste. &KFKEY enthält nur dann eine Tastenbezeichnung, wenn für diese Taste in der PARAMETER-Anweisung KFKEY='*taste*' angegeben war. Sonst wird die Systemvariable mit Leerzeichen versorgt.

Durch das Einführen der KFKEY-WHENEVER-Funktion ergibt sich beim Drücken der KF-Tasten als Ersatz für DUE beim Ausgeben großer Datenmengen, die mehr als einen Bildschirm füllen, ein anderes Verhalten (siehe DRIVE-Lexikon [3]).

4.2.2 Fehlerausgang bei Zuweisung ungültiger Variablenwerte

Wird einer Variablen ein ungültiger Wert zugewiesen, wird im Standardfall das Programm abgebrochen.
Wenn Sie einen solchen Programmabbruch verhindern wollen, müssen Sie einen 'CHECK ERROR'- oder 'CONVERSION ERROR'-Fehlerausgang definieren mit einer der beiden Anweisungen (siehe Abschnitt „Fehler abfangen, Endekriterien“ auf Seite 88):

– WHENEVER &ERROR IN ('CHECK ERROR', 'CONVERSION ERROR') CONTINUE

– WHENEVER &ERROR IN ('CHECK ERROR', 'CONVERSION ERROR')
CALL subprogname

In beiden Fällen gilt:

– die fehlerhafte Zuweisung wird nicht ausgeführt,

– der ursprüngliche Inhalt der Variablen bleibt bestehen,

– DRIVE/WINDOWS versorgt die Systemvariable &ERROR_STATE: Die Komponente &ERROR erhält den Wert 'CHECK ERROR' oder 'CONVERSION ERROR', die Komponente &VAR_NAME wird mit dem Namen der Variablen versorgt, die den Fehler verursachte sowie mit LINE_NR, DISPLACEMENT STATEMENT.

4.3 Meldungen sprachunabhängig lesen

Mit MSGSTRING (*numausdruck1* [[, *numausdruck2*], *name*]) können Sie Meldungen aus einer MIP-Meldungsdatei lesen. In der Meldungsdatei sind die Meldungen mit einer Meldungsnummer und einem Namenskürzel gekennzeichnet.

Der erste Parameter *numausdruck1* bezeichnet die Meldungsnummer, der dritte Parameter *name* das Namenskürzel in der Meldungsdatei. Der zweite Parameter *numausdruck2* ist für das BS2000 nicht notwendig und wird ignoriert, falls er angegeben wird.

Bei DECLARE-Anweisungen (z.B. DECLARE VARIABLE, DECLARE CONSTANT) erfolgt der Zugriff auf die Meldung zum Übersetzungszeitpunkt, bei ausführbaren Anweisungen (z.B. SET) zum Ablaufzeitpunkt. Das bedeutet, für vollständig sprachunabhängige Programme muß der Zugriff zum Ablaufzeitpunkt erfolgen.

Beispiele

Sie weisen der Variablen &v die DRIVE-Meldung 19 zu. Dies geschieht zum Ablaufzeitpunkt.

```
SET &v = MSGSTRING (19, DRI)
```

Sie weisen der Konstanten &const dieselbe Meldung zum Übersetzungszeitpunkt zu.

```
DECLARE CONSTANT &const MSGSTRING (19, DRI)
```

Sie weisen den Konstanten &msg1, &msg2 und &msg3 die DRIVE-Meldungen DRI0022, DRI0003 und DRI0011 zu und geben dann diese Konstanten in einem Bildschirmformat aus. Vorausgesetzt ist, daß die Meldungsdatei zugewiesen wurde (MC FILE=(ADD=SYSMSA.DRIVE.020)). XXX Läuft das Beispiel?

```
PROCEDURE "test.msg";
...
SET &meld='DRI';
...
DECLARE CONSTANT &msg1 MSGSTRING (22);
DECLARE CONSTANT &msg2 MSGSTRING (3,5,&meld);
DECLARE CONSTANT &msg3 MSGSTRING (11,DRI);
...
DISPLAY FORM LINE &msg1, &msg2, &msg3;
...
END PROCEDURE;
```

4.4 COPY-Elemente einfügen

COPY-Elemente, die in einer DRIVE-Bibliothek gespeichert sind, können mit der Anweisung COPY in ein Programm eingefügt werden. COPY-Anweisungen können sowohl im Deklarations- als auch im Verarbeitungsteil stehen. COPY-Elemente dürfen nicht geschachtelt werden, d.h. innerhalb eines COPY-Elements sind COPY-Anweisungen nicht erlaubt.

COPY-Elemente bestehen aus einer Folge oder Teilen von Anweisungen. Bei der Übersetzung eines Programms wird das COPY-Element in die Source an die Stelle der COPY-Anweisung kopiert und die Übersetzung des Programms fortgesetzt.

Da im Programm-Modus COPY-Anweisungen mit „;“ abgeschlossen werden müssen, kann mitten in einer Anweisung ein „;“ vorkommen bzw. können zwei Semikolons am Ende einer Anweisung stehen.

Beispiel 1

```
DECLARE C1 COPY A(B); WHERE PLZ='80000' OR COPY A(C); AND COPY A(D);
        ORDER BY PLZ;
```

Beispiel 2

```
DECLARE C1 CURSOR COPY A(B);;
```

Das erste Semikolon beendet die COPY-Anweisung, das zweite Semikolon beendet die DECLARE-Anweisung

4.5 Funktionen

Mit *charprim* und *wertefunktion* können Sie neben Anweisungen auch verschiedene vordefinierte Funktionen ausführen (siehe „DRIVE-Lexikon“ [3], Metavariablen *charprim*, *charausdruck* und *wertefunktion*). Diese Funktionen sind Stringfunktionen bzw. Numerische Funktionen.

4.5.1 Stringfunktionen

Eine Zeichenkette (= String) kann bei DRIVE/WINDOWS maximal 32000 Zeichen lang sein.

DRIVE/WINDOWS erlaubt folgende Stringfunktionen:

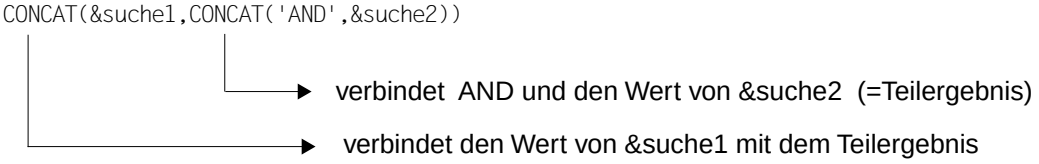
- Verknüpfen von Zeichenketten (CONCAT)
- Auswählen von Teilketten (SUBSTRING)
- Ersetzen und Ändern von Teilketten (UPDSTRING)
- Löschen von Teilketten (DELSTRING)
- Linksbündiges Abschneiden von Zeichenketten (SHIFTLEFTSTRING)
- Umsetzen von Kleinbuchstaben in Großbuchstaben (UPPERSTRING)
- Umsetzen von Großbuchstaben in Kleinbuchstaben (LOWERSTRING)
- Umsetzen von Zeichen (TRSTRING)
- Lesen von Meldungen aus einer Meldungsdatei (MSGSTRING) (siehe Abschnitt „Meldungen sprachunabhängig lesen“ auf Seite 91
- Verknüpfen aller atomaren Felder von strukturierten Variablen (CHARACTER)

Zeichenketten verknüpfen (Konkatenation)

```
CONCAT (&s1, &s2)  oder
&s1 || &s2
```

Sie verknüpfen Zeichenketten mit Hilfe des Schlüsselwortes CONCAT oder des Verknüpfungsoperators „||“ (siehe „DRIVE-Lexikon“ [3], Metavariablen *charausdruck* und *charprim*). Klammern umschließen bei CONCAT die Ausdrücke, die verbunden werden sollen; ein Komma muß jeden Ausdruck abschließen.

Die Auswertungsreihenfolge kann mit Hilfe von Klammern beeinflußt werden. Klammern können geschachtelt werden, ebenso wie CONCAT oder „||“.



Beispiel

```
SET &suche1 = '(LAND = &hland)';
SET &suche2 = '(GEHALT >= &g_unter) AND (GEHALT <= &g_ober)';
...
SET &suche = &suche1 || '␣AND␣' || &suche2;
```

Die Variable &suche enthält nun folgenden Wert:

```
"(LAND = &hland) AND ( GEHALT >= &g_unter ) AND ( GEHALT <= &g_ober) ";
```

Teilketten auswählen

```
SUBSTRING (&s1, startpos, länge)
```

Mit SUBSTRING bestimmen Sie, welcher Teil eines Ausdruck dargestellt sein soll: ab angegebener Position *startpos* in der Länge *länge*.

Beispiel

```
SET &s1 = '4GL DRIVE/WINDOWS';
...
SET &s1 = SUBSTR (&s1,5,5)
```

Die Variable &s1 enthält den Wert „DRIVE“.

Teilketten ersetzen und ändern

```
UPDSTRING (&s1, &u1, updpos)
```

Mit UPDSTRING „überschreiben“ Sie den Inhalt eines Ausdrucks durch den Inhalt eines anderen Ausdrucks ab der Position *updpos*.

Beispiel

```
SET &s1 = 'DRIVE V6.0␣␣␣␣␣␣␣␣␣';
SET &u1 = '/WINDOWS V2.0';
...
SET &s1 = UPDSTR (&s1, &u1, 6);
```

Die Variable &s1 enthält den Wert „DRIVE/WINDOWS V2.0“.

Teilketten löschen

```
DELSTRING (&s1, startpos, länge)
```

Mit DELSTRING löschen Sie Teilketten ab einer bestimmten Position (*startpos*) entweder ganz oder auf einer bestimmten Länge (mit der Angabe von *länge*).

Beispiele

Für die folgenden Beispiele gilt:

```
&s1 = 'DRIVE/WINDOWS V2.0';

...
SET &v1 = DELSTRING (&s1,14);
```

Die Variable &v1 enthält den Wert „DRIVE/WINDOWS“.
Der Ausdruck `DELSTRING (&s1,14)` entspricht dem Ausdruck `SUBSTRING (&s1,1,13)`.

```
...
SET &v2 = DELSTRING (&s1,6,8);
```

Die Variable &v2 enthält den Wert „DRIVE V2.0“.
Der Ausdruck `DELSTRING (&s1,6,8)` entspricht dem Ausdruck `SUBSTRING (&s1,1,5) || SUBSTRING (&s1,14,5)`.

Zeichenketten linksbündig abschneiden

```
SHIFTLEFTSTR (&s1)
```

Mit SHIFTLEFTSTRING löschen Sie linksbündige Zeichen, deren hexadezimale Darstellung kleiner gleich dem Leerzeichen (≤ X'40') ist.

Beispiel

```
SET &s1 = '␣␣␣␣DRIVE/WINDOWS V2.0';
...
SET &v1 = SHIFTLEFTSTRING (&S1);
```

Die Variable &v1 enthält den Wert „DRIVE/WINDOWS V2.0“.

Kleinbuchstaben in Großbuchstaben umsetzen

```
UPPERSTRING (&s1)
```

Mit UPPERSTRING ersetzen Sie alle Kleinbuchstaben durch Großbuchstaben.

Beispiel

```
&s1 = 'Drive/Windows v2.0';

...
SET &v1 = UPPERSTRING (&s1);
```

Die Variable &v1 enthält den Wert „DRIVE/WINDOWS V2.0“.

Großbuchstaben in Kleinbuchstaben umsetzen

```
LOWERSTRING (&s1)
```

Mit LOWERSTRING ersetzen Sie alle Großbuchstaben durch Kleinbuchstaben.

Beispiel

```
&s1 = 'Drive/Windows v2.0';

...
SET &v1 = LOWERSTRING (&s1);
```

Die Variable &v1 enthält den Wert „drive/windows v2.0“.

Zeichen umsetzen

```
TRSTRING (&s1, &s2, &s3)
```

Mit TRSTRING ersetzen Sie zeichenweise Zeichen in *&s1*. Die zu ersetzenden Zeichen stehen in *&s2*, die ersetzenden Zeichen in *&s3*. Ein Zeichen *c* aus &s1 wird nur dann umgesetzt, wenn es auch in *&s2* vorhanden ist. Die Position *n* des Zeichens *c* in *s2* bestimmt das Zeichen, durch das es ersetzt wird. Das ersetzende Zeichen steht an Position *n* in *&s3*.

&s2 mit den zu ersetzenden Zeichen und &s3 mit den ersetzenden Zeichen sollen gleich lang sein. Die Positionen eines zu ersetzenden Zeichen in &s2 und eines ersetzenden Zeichen in &s3 sind gleich.

Beispiel

```
&s1 = PEKING ;
&s2 = JKPTU ;
&s3 = IJBDO ;

...
SET &v1 = TRSTRING (&s1,&s2,&s3);
```

Die Variable &v1 enthält den Wert „BEJING“.

Komponenten strukturierter Variablen verknüpfen

```
CHARACTER (&var)
```

Mit CHARACTER verbinden Sie alle einfachen Komponenten, d.h. alle atomaren Felder einer Variablen, logisch miteinander. Voraussetzung dafür ist, daß alle einfachen Komponenten vom Datentyp CHARACTER oder VARCHAR sind.

Beispiel

```
DECLARE VARIABLE 1 &t,
                   2 t1 CHARACTER (13) INIT 'DRIVE/WINDOWS',
                   2 t2 CHARACTER,
                     3 t21 CHARACTER (8) INIT '␣bietet␣',
                     3 t22 CHARACTER (13);
...
```

```
SET &t22 = 'Konkatenation';
...
SET &t = CHARACTER(&t);
```

Die Variable &t enthält den Wert „DRIVE/WINDOWS bietet Konkatenation“.

CHARACTER-Variablen numerische, Datums- oder Zeit-Variablen zuweisen

Außerdem können Sie mit der CHARACTER-Funktion Datenwerte numerischer Variablen an CHARACTER-Variablen zuweisen, so daß die Redefinition von numerischen Variablen nicht mehr nötig ist. Bei der CHARACTER-Funktion gewährleistet das DRIVE-RTS (Runtime-System) die Portabilität der Programme, während Programme mit der REDEFINES-Klausel maschinenabhängig sind und nicht portabel.

Beispiele

Die numerische Variable &n erhält den Datentyp CHARACTER und wird der Variablen &c zugewiesen.

```
SET &c = CHARACTER(&n);          &n muß den Datentyp NUM haben
```

statt

```
DECLARE VARIABLE &n NUM(2),
                 &c CHARACTER(2) REDEFINES &n;
```

Die Variablen &t (Datentyp TIME) und &d (Datentyp DATE) erhalten den Datentyp CHARACTER und werden den Variablen &t und &d zugewiesen.

```
DECLARE VARIABLE &c10 CHAR(10),
                 &c8  CHAR( 8);
DECLARE VARIABLE &t TIME,
                 &d DATE;
...
SET &c10 = CHARACTER(&d);
SET &c8  = CHARACTER(&t);
```

4.5.2 Numerische Funktionen

Mit *wertefunktion* berechnen Sie einen Wert, indem eine Funktion auf genau ein Argument angewendet wird (siehe „DRIVE-Lexikon“ [3], Metavariable *wertefunktion*).

Länge einer Zeichenkette feststellen

```
CHARLENGTH (charausdruck)
```

Mit CHARLENGTH erhalten Sie die Position des letzten alphanumerischen Zeichens in der Zeichenkette *charausdruck*. Das Zeichen kann auch ein Leerzeichen sein. Bei einer Leerzeichenkette wird der Wert 0 zurückgeliefert.

Beispiel

```
DECLARE CONSTANT &co1 '12345';
DECLARE VARIABLE &vc1 VARCHAR (10) INIT '.........';
DECLARE VARIABLE &vc2 VARCHAR (32000) INIT '␣' NOCHECK;

DECLARE VARIABLE &n1 NUM (3),
DECLARE VARIABLE &n2 NUM (3),
DECLARE VARIABLE &n3 NUM (3),
...
SET &n1 = CHARLENGTH (&co1);
SET &n2 = CHARLENGTH (&vc1);
SET &n3 = CHARLENGTH (&vc2);
```

Die Variable &n1 enthält den Wert 5, die Variable &n2 den Wert 10 und die Variable &n3 den Wert 1.

Position des Zeichens ausgeben, dem in einer Zeichenkette nur Leerzeichen folgen

```
LENGTH (charausdruck)
```

Mit LENGTH erhalten Sie die Position des Zeichens in der Zeichenkette *charausdruck*, dem nur Leerzeichen folgen. Bei einer Leerzeichenkette wird der Wert 0 zurückgeliefert.

Beispiel

```
DECLARE CONSTANT &co1 '123456789012345678901234567890␣␣␣␣␣␣␣␣␣␣';

DECLARE VARIABLE &vc1 VARCHAR (32000) INIT '␣';

DECLARE VARIABLE &n1 NUM (3),
DECLARE VARIABLE &n2 NUM (3),
...
SET &n1 = LENGTH (&co1);
SET &n2 = LENGTH (&vc1);
```

Die Variable &n1 enthält den Wert 30, die Variable &n2 den Wert 0.

Position von Zeichenketten in einer Zeichenkette ausgeben

```
POSITION (charausdruck1 IN charausdruck2 [, n ] )
```

Mit POSITION erhalten Sie die Position von Zeichenketten in Zeichenketten. Wenn ein Wiederholungsfaktor n angegeben ist, liefert POSITION die Position, an der *charausdruck1* zum n-ten Mal in *charausdruck2* vorkommt.

Beispiel

```
DECLARE CONSTANT &co1 '12345678901234567890123456789OXXXXXXXXXX';

DECLARE VARIABLE &vc1 VARCHAR (10) INIT '3.14158265';
DECLARE VARIABLE &vc2 VARCHAR (32000) INIT '-' NOCHECK CHECK &vc2 IS NUMERIC;

DECLARE VARIABLE &f2 INTEGER INIT 2;
DECLARE VARIABLE &f3 SMINT INIT 1;

DECLARE VARIABLE &n1 NUM (3),
DECLARE VARIABLE &n2 NUM (3),
DECLARE VARIABLE &n3 NUM (3),
...
SET &n1 = POSITION ('XXXX' IN &co1,1);
SET &n2 = POSITION ('5' IN &vc1,&f2);
SET &n3 = POSITION ('-' IN &vc2,&f3);
```

Die Variable &n1 enthält den Wert 31, die Variable &n2 den Wert 10 (= diejenige Position mit der zweiten 5) und die Variable &n3 den Wert 1.

MOD - Rest einer Division bestimmen

Mit MODULO berechnen Sie den Rest der ganzzahligen Division eines numerischen Ausdrucks durch einen numerischen Ausdruck (siehe DRIVE-Lexikon [3]). Das Ergebnis hat das Vorzeichen des Dividenden.

Beispiele

Der Rest der Division von &zahl durch &teiler wird ausgegeben.

```
DECLARE VARIABLE &zahl   NUM (5);
DECLARE VARIABLE &teiler NUM (5);
DECLARE VARIABLE &rest   NUM (5);
...
SET &rest=MODULO(&zahl,&teiler);
SEND MESSAGE 'Der Rest von ', &zahl, ' durch ' &teiler, ' ist: ', &rest;
```

Es wird geprüft, ob &zahl durch &teiler ohne Rest teilbar ist.

```
DECLARE VARIABLE &zahl   NUM (7);
DECLARE VARIABLE &teiler NUM (7);
...
IF MODULO(&zahl,&teiler) = 0
   THEN
      SEND MESSAGE &zahl, ' ist durch ' &teiler, ' ohne Rest teilbar.';
   ELSE
      SEND MESSAGE &zahl, ' ist nicht ohne Rest durch ' &teiler, ' teilbar.';
END IF;
```

Numerischen Wert prüfen und in Zeichenfolge konvertieren

Mit NUMERIC überprüfen Sie, ob eine Zeichenfolge einen numerischen Wert repräsentiert (siehe „DRIVE-Lexikon“ [3], Metavariable *bedingung*).

Ist dies der Fall, können Sie diese Zeichenfolge in einen numerischen Wert konvertieren (siehe „DRIVE-Lexikon“ [3], Metavariable *wertefunktion*).

Beispiel

```
DECLARE VARIABLE &n NUM (10),
DECLARE VARIABLE &c CHARCTER (10),

SET &c = 'ABCDEF';

   IF &c IS NOT NUMERIC
      THEN SEND MESSAGE 'VARIABLE NICHT NUMERISCH';
   END IF;
...
 SET &c = '1000';

 IF &c IS NUMERIC
     THEN SET &n = NUMERIC(&c)
     ELSE SET &n = 0;
 END IF;
```

ROUND Numerische Werte runden

```
ROUND (numausdruck [, s1 ])
```

Mit ROUND runden Sie numerische Werte. Legen Sie keine Stelle *s1* fest, wird die erste Nachkommastelle gerundet, d.h. das Ergebnis ist eine ganze Zahl.

Ist der Wert für die Rundungstelle *s1* positiv, wird auf diese Nachkommastelle gerundet.

Ist der Wert für die Rundungsstelle *s1* negativ, wird diese Vorkommastelle gerundet und wie die folgenden Vorkommastellen mit Null aufgefüllt (siehe DRIVE-Lexikon [3], Metavariable *wertefunktion*).

Beispiel

```
ROUND (1234.2234, 2)  = 1234,22
ROUND (1234,2234, -2) = 1200
ROUND (1234,2234)     = 1234
```

TRUNC Numerische Werte abschneiden

```
TRUNC (numausdruck [, s1 ])
```

Mit TRUNC schneiden Sie numerische Werte ab. Legen Sie keine Stelle *s1* fest, werden die Nachkommastellen abgeschnitten, d.h. das Ergebnis ist eine ganze Zahl.

Ist der Wert für die Schnittstelle *s1* positiv, wird nach dieser Stelle abgeschnitten.

Ist der Wert für die Schnittstelle *s1* negativ, wird an dieser Vorkommastelle abgeschnitten und mit Null für die ganze Zahl aufgefüllt.

Beispiel

```
TRUNC  (1234.2234, 2)  = 1234,22
TRUNC  (1234.2234)     = 1234
TRUNC  (1234.2234, -2) = 1200
```

4.6 Anweisungen dynamisch ausführen (EXECUTE)

Bei der dynamischen Anweisungsausführung werden Anweisungen erst zum Ablaufzeitpunkt generiert, analysiert und anschließend ausgeführt.

Dazu geben Sie an: EXECUTE gefolgt von *charausdruck*. *charausdruck* kann auch ein Literal oder eine Variable sein. Damit ist es möglich, Anweisungen dynamisch auszuführen. Sie können zum Beispiel einen Cursor dynamisch öffnen:

```
EXECUTE 'OPEN cursor1';
```

Jede DRIVE-Anweisung ist in einem Programm mit einem Strichpunkt abzuschließen. Steht die Anweisung jedoch in einer EXECUTE-Anweisung, braucht nur die EXECUTE-Anweisung mit einem Semikolon abgeschlossen zu werden.

Wenn *charausdruck* als Literal angegeben wird, darf dieses Literal max. 256 Zeichen lang sein. Deshalb müssen ggf. sehr lange Anweisungen „zeilenweise“ umgesetzt werden.

Welche DRIVE-Anweisungen mit EXECUTE dynamisch ausgeführt werden können, zeigt die folgende Tabelle.

DRIVE-Anweisung	Bedeutung
CALL	externes Unterprogramm aufrufen
DECLARE FORM	Kompakt-Bildschirmformat definieren und ausgeben
DECLARE LIST	Kompakt-Listenformat definieren und ausgeben
DISPLAY	Format ausgeben
DO	Dialog-Programm starten
ENTER	UTM-Asynchronvorgang starten
FILL	Format aufbauen
LIST	UTM-Druckaufträge ausgeben
SEND MESSAGE	Meldungen ausgeben
SET	Werte und Feldattribute zuweisen
UNSAVE	Programme, COPY-Elemente,Benutzerkennsätze löschen

EXECUTE 'CALL *subprogname*' ist nicht erlaubt.

SQL-Anweisungen, die mit EXECUTE dynamisch ausgeführt werden können:

SQL-Anweisung	Bedeutung
ALTER TABLE	Tabellenstruktur ändern
CLOSE cursor	Cursor schließen
COMMIT WORK	Transaktion beenden
CREATE SCHEMA	Schema erzeugen
CREATE TABLE	Tabelle erzeugen
CREATE TEMPORARY VIEW	View deklarieren
CREATE VIEW	View deklarieren
DECLARE ... CURSOR ...	Cursor deklarieren
DELETE	Datensatz löschen
DROP CURSOR	Cursor freigeben
DROP CURSORS	alle Cursor freigeben
DROP SCHEMA	Schema löschen
DROP TABLE	Tabelle löschen
DROP VIEW	View freigeben
DROP VIEWS	alle Views freigeben

SQL-Anweisung	Bedeutung
FETCH	Cursor positionieren und Variablen versorgen
GRANT	Zugriffsrechte vergeben
INSERT	Datensatz einfügen
OPEN	Cursor öffnen
PERMIT	Benutzeridentifikation angeben
PRAGMA	Pragma spezifizieren
RESTORE	Cursorposition aufheben
REVOKE	Zugriffsrechte entziehen
ROLLBACK WORK	Transaktion zurücksetzen
SELECT	Datensatz auswählen
SET CATALOG	Datenbanknamen voreinstellen
SET SCHEMA	Schemanamen voreinstellen
SET SESSION	Berechtigungsschlüssel festlegen
SET TRANSACTION	Transaktionsbedingungen festlegen
STORE	Cursorposition speichern
UPDATE	Datensatz ändern

Soll in einer dynamischen Anweisung eine deklarative Größe verwendet werden, muß diese Größe im Programmablauf vorher definiert worden sein: entweder statisch im Deklarationsteil oder dynamisch in einem vorhergehenden EXECUTE.

Wenn Sie mit dem Datenbanksystem SESAM V2 arbeiten, können Sie in einer dynamischen SQL-Anweisung keine statischen SQL-Objekte referenzieren.

Werden deklarative Größen dynamisch definiert, kann beim statischen Aufbau eines Programms bei nicht-deklarativen Anweisungen auf diese Größen nicht zugegriffen werden.

Werden in einem Programm Cursor oder Views dynamisch deklariert, sind diese bis zum Ende des Programms gültig. Vorzeitig kann ihre Gültigkeit nur durch ein dynamisches DROP, das explizit angegeben wird, aufgehoben werden.

4.6.1 Dynamische SQL-Anweisungen

Sie können mit EXECUTE die in der Tabelle auf Seite 102 angeführten Anweisungen dynamisch ausführen, indem Sie sich mit den Stringfunktionen (siehe Abschnitt „Stringfunktionen“ auf Seite 92) zum Ablaufzeitpunkt einen String erzeugen, dessen Inhalt eine dynamisch ausführbare SQL-Anweisung ist.

Wollen Sie eine Cursordeklaration variabel halten, so können Sie einen variablen Cursor deklarieren. Dazu lassen Sie bei der statischen DCL ... CURSOR-Anweisung die FOR-Klausel weg und deklarieren sie zum Ablaufzeitpunkt mit der EXECUTE-Anweisung nach. Die statische und dynamische Cursordeklaration müssen bis auf die FOR-Klausel gleich sein. Alle anderen Anweisungen, die sich auf diesen deklarierten Cursor beziehen (OPEN, FETCH, CYCLE, DROP ... CURSOR), können Sie für den variablen Cursor statisch angeben, wodurch die Performance steigt, weil zum Ausführungszeitpunkt keine Generierung und Analyse der Anweisung mehr erforderlich sind.

Wollen Sie neben der Suchbedingung auch den Namen und die Zugriffsart variabel halten, müssen Sie den Cursor voll dynamisch deklarieren.

Beispiel1

Eine Cursordeklaration soll in der Suchbedingung variabel gehalten werden. Statisch wird eine DECLARE ... CURSOR-Anweisung ohne FOR-Klausel angegeben.

```
DECLARE cur_anzeige SCROLL CURSOR;
```

Mit einer EXECUTE-Anweisung wird die Suchbedingung zum Ablaufzeitpunkt nachdeklariert.

```
EXECUTE 'DECLARE cur_anzeige SCROLL CURSOR FOR SELECT land, nachname,
                                                      vorname, gehalt
         FROM db_mitarbeiter
         WHERE (land = &hland) AND
               (gehalt >= &g_unter) AND (gehalt <=&g_ober);';
```

Alle Anweisungen, die sich auf den Cursor beziehen, können statisch im Programm angegeben werden.

```
CYCLE cur_anzeige INTO &anzeigesatz.*;
END CYCLE;
DROP cur_anzeige;
...
COMMIT WORK;
EXECUTE 'DECLARE cur_aendern CURSOR FOR SELECT land, nachname, vorname, gehalt
         FROM db_mitarbeiter
         WHERE (land = &hland) AND
               (gehalt = &höchstsatz);';
```

Zwischen der DROP-Anweisung und einer EXECUTE 'DECLARE ...'-Anweisung auf den gleichen Cursornamen (statische Cursordeklaration ohne FOR-Klausel) sollte ein COMMIT WORK stehen.

Beispiel2

Eine Cursordeklaration soll vollkommen variabel gehalten werden. Zum Ablaufzeitpunkt wird mit Hilfe der Konkatenation die Anweisung für die Deklaration aufgebaut und mit EXECUTE dynamisch ausgeführt. Die Suchbedingung steht in der Variablen &suche.

```
EXECUTE CONCAT (&cursordec,CONCAT(&suche,';'));
```

Die zusammengesetzte DECLARE CURSOR-Anweisung lautet:

```
DECLARE cur_anzeige CURSOR FOR SELECT land, nachname, vorname, gehalt
        FROM db_mitarbeiter
        WHERE (land = &hland) AND
              (gehalt >=&g_unter) AND (gehalt <= &g_ober);
```

Das Programm arbeitet mit dem Cursor „cur_anzeige“ weiter, indem alle Anweisungen, die sich auf den Cursor beziehen, dynamisch mit EXECUTE gegeben werden.

```
EXECUTE 'OPEN cur_anzeige;';
CYCLE;
  EXECUTE 'FETCH cur_anzeige INTO &anzeigesatz.*;';
  IF &DML_STATE = 'TABLE END' THEN
     EXECUTE 'CLOSE cur_anzeige;';
     BREAK CYCLE;
  END IF;
END CYCLE;
...
EXECUTE 'DROP cur_anzeige;';
```

4.6.2 Beispiel

Das Programm „MITANZ“ ermöglicht die Auswahl von Mitarbeitern nach unterschiedlichen Ländern. Zusätzlich kann abgefragt werden, welche Mitarbeiter sich in einem gewissen Gehaltsrahmen bewegen.

Programm-Aufruf:

```
DO MITANZ
```

```
                              M I T A R B E I T E R



          KREUZEN SIE ZUTREFFENDES AN :


          DEUTSCHLAND X           GEHALT ZWISCHEN 2000 UND 4000
          USA                     GEHALT ZWISCHEN 4000 UND 6000 X
          SCHWEIZ                 GEHALT ZWISCHEN 6000 UND 8000
          ENGLAND                 GEHALT UEBER 8000




_____________________________________________________________________________
```

Durch Ankreuzen (X) und DUE erfolgt die Ausgabe aller Mitarbeiter aus Deutschland, deren Gehalt zwischen 4000 und 6000 liegt:

```
                              MITARBEITERANZEIGE
                              __________________

FRG  Mitscher             Fred                 4900.00
FRG  Jansen               Jutta                4700.00
FRG  Olsen                Mattes               4000.00
FRG  Mattsen              Gunter               4800.00
FRG  Nonnen               Paul                 4500.00
FRG  Andermatt            Sylvia               5100.00
FRG  Zimmermann           Peter Johannes       5800.00
FRG  Sennert              Gustl                5800.00
FRG  Ammerl               Sepp                 5700.00
FRG  Dormagen             Siegfried            5500.00
FRG  Pollinger            Nora                 4000.00
FRG  Bergner              Erich                4500.00


=============================================================================
```

Source MITANZ:

```
PROCEDURE MITDYNAMIC;

/***** DEKLARATIONSTEIL *****/

DCL VAR &L(4) CHAR(1),           /* Länder-Vektor zum Ankreuzen */
        &G(4) CHAR(1),           /* Gehalts-Vektor zum Ankreuzen */
        &H1(4) CHAR(3),          /* Hilfs-Vektor für die Länderangabe */
        &G_UNTER NUM(7,2) ,
        &G_OBER NUM(7,2) ,
        &INDEX NUM(1) INIT 1,
        &HLAND CHAR(3),
        &SUCHE1 CHAR(60),
        &SUCHE2 CHAR(60),
        &SUCHE CHAR (120),
DCL ANZ CURSOR;
DCL FORM MITARBEITERABFRAGE                          /* Format-Definition */
      TTITLE NL 1,                                   /* für die Abfrage   */
             TAB 30,'M I T A R B E I T E R',NL 4,
             TAB 10,'KREUZEN SIE ZUTREFFENDES AN :',NL 3,
             TAB 10,'DEUTSCHLAND ',RETURN &L(1),
             TAB 35,'GEHALT ZWISCHEN 2000 UND 4000 ',RETURN &G(1),NL 2,
             TAB 10,'USA ',RETURN &L(2),
             TAB 35,'GEHALT ZWISCHEN 4000 UND 6000 ',RETURN &G(2),NL 2,
             TAB 10,'SCHWEIZ ',RETURN &L(3),
             TAB 35,'GEHALT ZWISCHEN 6000 UND 8000 ',RETURN &G(3),NL 2,
             TAB 10 ,'ENGLAND ',RETURN &L(4),
             TAB 35,'GEHALT UEBER 8000 ',RETURN &G(4)
      BTITLE ' ','-'(80),NL 1;
DCL FORM MITARBEITERAUSGABE                          /* Format-Definition */
     TTITLE NL 1,                                    /* für die Ausgabe   */
            ' '(30),'MITARBEITERANZEIGE',NL 1,
            ' '(29),'-'(20)
     BTITLE '='(80);

/***** VERARBEITUNGSTEIL *****/

DISPLAY MITARBEITERABFRAGE;
SET &H1(1) = 'FRG';
SET &H1(2) = 'USA';
SET &H1(3) = 'CH';
SET &H1(4) = 'ENG';
```

```
CYCLE WHILE &INDEX < 5;
    IF &L(&INDEX) <> ' ' THEN
         SET &HLAND = &H1(&INDEX);         /* Feststellen welches Land   */
    END IF;                                /* angekreuzt ist             */
    IF &G(&INDEX) <> ' ' THEN
         SET &G_UNTER = &INDEX * 2000;     /* Feststellen welches Gehalt */
         SET &G_OBER = &G_UNTER +2000;     /* angekreuzt ist             */
    END IF;
    SET &INDEX = &INDEX + 1;
END CYCLE;
IF &G_OBER = 10000 THEN
    SET &G_OBER =20000;
END IF;
SET &SUCHE1 =' (LAND = &HLAND)';
SET &SUCHE2 =' (GEHALT >= &G_UNTER) AND (GEHALT >= &G_OBER)';

/*** Abfragen nach den angekreuzten Ländern und/oder Gehaltsbereichen ***/
/***       Die Variable &SUCHE wird dabei mit einem Wert belegt        ***/

CASE;

   OF (&HLAND = ' ') AND (&G_UNTER <> 0) SET &SUCHE = &SUCHE2;
   OF (&HLAND <> ' ') AND (&G_UNTER = 0) SET &SUCHE = &SUCHE1;
   OF (&HLAND <> ' ') AND (&G_UNTER <> 0) SET &SUCHE =
                      CONCAT(&SUCHE1,CONCAT(' AND ',&SUCHE2));
   OF (&HLAND = ' ') AND (&G_UNTER = 0) SET &SUCHE = ' (GEHALT > 1) ';

END CASE;

/*** Dynamische Anweisungsausführung ***/
EXEC CONCAT ('DCL ANZ CURSOR FOR S LAND,NACHNAME,VORNAME,GEHALT FROM
      SESAMDBMIT WHERE', &SUCHE);
/*** Statischer Programmteil ***
CYCLE ANZ INTO &ANZEIGESATZ.*;
     FILL MITARBEITERAUSGABE TABLE VALUES &ANZEIGESATZ;
END CYCLE;
DISPLAY MITARBEITERAUSGABE;
ROLLBACK WORK;
END PROCEDURE;
```

4.7 Interne Unterprogramme aufrufen

Unter einem internen Unterprogramm versteht man eine benannte Anweisungsfolge, die innerhalb eines Programms beliebig oft aufgerufen wird. Innerhalb eines Programms können interne Unterprogramme definiert werden. Alle internen Unterprogramme müssen vollständig im Deklarationsteil nach den DECLARE-Anweisungen definiert werden.

Die erste Anweisung eines internen Unterprogramms muß SUBPROCEDURE gefolgt von dem Namen des internen Unterprogramms sein. Die letzte Anweisung muß END SUBPROCEDURE sein. Aufgerufen wird das Unterprogramm mit CALL. Steht innerhalb eines internen Unterprogramms eine BREAK SUBPROCEDURE-Anweisung, wird das Programm mit der Anweisung fortgesetzt, die dem CALL-Aufruf für dieses interne Unterprogramm folgt.

Innerhalb eines internen Unterprogramms dürfen keine DECLARE-Anweisungen stehen. Es dürfen andere interne Unterprogramme nur aufgerufen werden, wenn sie vorher vollständig definiert wurden.

Strukturierungsanweisungen (CASE, CYCLE, IF) müssen innerhalb des internen Unterprogramms auch wieder abgeschlossen werden (END CASE, END CYCLE, END IF). Eine BREAK CYCLE-Anweisung kann keine außerhalb dieses internen Unterprogramms beginnende Schleife abbrechen.

Mit den Anweisungen DISPATCH und END DISPATCH können Sie alle CALL-Anweisungen zur mehrstufigen Verteilten Transaktionsverarbeitung bündeln (siehe Abschnitt „Parallele Verteilte Verarbeitung anstoßen mit DISPATCH“ auf Seite 343).

Steht innerhalb eines internen Unterprogramms ein BREAK PROCEDURE, wird sowohl das interne Unterprogramm als auch das gesamte Programm abgebrochen.

4.8 Externe DRIVE-Unterprogramme aufrufen

Beim Erstellen von komplexen Programmen ist es zweckmäßig, Teile des Programms in externen Unterprogrammen auszulagern. Externe Unterprogramme sind eigenständige Programme, die von anderen Programmen beliebig oft aufgerufen werden können. Externe Unterprogramme unterscheiden sich nach der Sprache, in der sie geschrieben sind:

– in DRIVE/WINDOWS

– in anderen Programmiersprachen

Beiden externen Unterprogrammarten ist gemeinsam, daß sie von verschiedenen Programmen aufgerufen werden können, aber nur einmal definiert werden müssen. In beiden Fällen können vom rufenden Programm an das externe Unterprogramm Ein-/Ausgabeparameter übergeben werden.

Externe Unterprogramme haben dieselbe Programmstruktur wie rufende Programme: Startanweisung, Deklarationsteil, Verarbeitungsteil, Endanweisung (siehe Abschnitt „Programmstruktur“ auf Seite 7).

Die externen Unterprogramme, die in DRIVE/WINDOWS programmiert sind, können mit der Anweisung DO als Dialog-Programme, mit der Anweisung ENTER als UTM-Asynchronvorgänge oder mit der Anweisung CALL aufgerufen werden.

Wird das externe Unterprogramm mit DO aufgerufen, wird das rufende Programm abgebrochen und das externe Unterprogramm ausgeführt.

Wird das externe Unterprogramm mit ENTER aufgerufen, wird das externe Unterprogramm als UTM-Asynchronvorgang gestartet und das rufende Programm läuft weiter.

Wird das externe Unterprogramm mit CALL aufgerufen, wird das rufende Programm unterbrochen, das externe Unterprogramm ausgeführt und anschließend das rufende Programm mit der Anweisung, die dem CALL-Aufruf für das externe Unterprogramm folgt, fortgesetzt.
Mit CALL gerufene externe Unterprogramme können auch Parameter zurückgeben.

4.9 Externe fremdsprachige Unterprogramme

Externe Unterprogramme in anderen Programmiersprachen rufen Sie mit der Anweisung CALL MODULE auf und geben eine Direktive entsprechend der Programmiersprache an, in der das Programm geschrieben ist. Die Standard-Voreinstellung ist COBOL, C-Programme werden mit der Direktive C aufgerufen (siehe „DRIVE-Lexikon“ [3]). Das rufende Programm wird dann unterbrochen, das externe Unterprogramm ausgeführt und anschließend das rufende Programm nach dem CALL-Aufruf fortgesetzt.

Sie bereiten den Einsatz von externen Unterprogrammen in anderen Programmiersprachen in folgenden Schritten vor:

– das externe Unterprogramm erstellen

– das externe Unterprogramm übersetzen (nachgeladen wird es zur Ausführung)

Erstellen des externen Programms

Beim Erstellen des externen Unterprogramms müssen Sie folgende Punkte beachten.

In externen Unterprogrammen anderer Programmiersprachen dürfen keine Zugriffe auf Datenbanken erfolgen.

In externen fremdsprachigen Unterprogrammen darf außerdem nicht auf Betriebsmittel von DRIVE/WINDOWS zugegriffen werden, d.h. folgende Funktionen und Aufrufe dürfen Sie nicht verwenden:

– Benutzung von Ein- und Ausgabe-Operationen (z.B. Ausgeben von Daten an der Datensichtstation),

– Benutzung von STXIT

– KDCS-Aufrufe im UTM-Betrieb

– Zugriffe auf DRIVE-Speicherbereiche. Sollen Daten eines externen Unterprogramms für ein weiteres Unterprogramm verwendet werden, müssen sie über Variablen des rufenden Programms übergeben werden.

– IPK-Aufrufe

Wie Sie Daten an externe Unterprogramme anderer Programmiersprachen übergeben, ist im Abschnitt „Daten an Unterprogramme in anderen Programmiersprachen übergeben“ auf Seite 112 beschrieben.

Externe Unterprogramme sind nur ablauffähig, wenn sie mit dem Laufzeitsystem der jeweiligen Programmiersprache zusammengebunden werden. Werden sie nicht statisch mit dem jeweiligen Laufzeitsystem zusammengebunden, müssen sie beim ersten Aufruf des Programms dynamisch gebunden werden. Zum dynamischen Binden muß das Laufzeitsystem in einer der folgenden Bibliotheken, die vom dynamischen Bindelader (DLL) durchsucht werden, enthalten sein:

– Bibliothek, in der das externe Unterprogramm enthalten ist,
– für COBOL: Bibliothek, die mit SET-TASKLIB LIB = ... als Bindemodul-Bibliothek zugewiesen wurde,
– Bibliothek $TSOS.TASKLIB.

4.9.1 Daten an Unterprogramme in anderen Programmiersprachen übergeben

Mit CALL MODULE können Sie ein Unterprogramm (C, COBOL, ASSEMBLER ...) aufrufen sowie Parameter an das Unterprogramm übergeben.

Bei der CALL MODULE-Anweisung müssen Sie eine Direktive entsprechend der Programmiersprache angeben, in der das Programm geschrieben ist. Die Standard-Voreinstellung ist COBOL, C-Programme werden mit der Direktive C aufgerufen (siehe DRIVE-Lexikon [3]).

Beim Vergeben von Namen müssen Sie sich an die Regeln der jeweiligen Programmiersprache halten. Außerdem dürfen keine reservierten Wörter der jeweiligen Programmiersprache oder von DRIVE/WINDOWS verwendet werden, und der Name darf weder mit 'idr' noch mit 'dri' beginnen.

Die Anzahl und die Formate der Übergabe-Parameter bei CALL muß mit den Felddefinitionen im aufgerufenen Unterprogramm übereinstimmen. An der CALL-Schnittstelle kann keine entsprechende Prüfung durchgeführt werden.

Als Parameter können nur Variablen übergeben werden. Dabei werden die Parameter differenziert nach:

– Eingabe-Parametern:

  Daten, die von einem rufenden Programm an ein Unterprogramm übergeben werden (z.B. als Operand für Rechenoperationen). Nachdem das Unterprogramm beendet ist, werden keine entsprechenden (Ergebnis-)Daten an das rufende Programm zurückgegeben.

– Ein-/Ausgabe-Parametern:

  Daten, die von einem rufenden Programm an ein Unterprogramm übergeben und dort bearbeitet werden. Nachdem das Unterprogramm beendet ist, werden die Ergebnisdaten an das rufende Programm zur weiteren Bearbeitung zurückgegeben.
  Ein-/Ausgabe-Parameter dürfen pro CALL-Anweisung nur jeweils einmal angegeben werden.

  Hat der Eingabe-Parameter den Datentyp SMALLINT, sind für den Ausgabe-Parameter alle Werte zwischen -32768 und +32767 erlaubt.

  Hat der Eingabe-Parameter den Datentyp INTEGER, sind für den Ausgabe-Parameter alle Werte zwischen -2147483648 und + 217483647 erlaubt.

  Hat der Eingabe-Parameter den Datentyp DATE oder TIME, erwartet DRIVE/WINDOWS als Rückgabewert ein gültiges Datum oder eine gültige Uhrzeit in der entsprechenden Form (siehe DRIVE-Lexikon [3], Metavariable *datumzeitterm*).

Bei der Berechnung der Gesamtlänge aller Werte gehen nur die Längen der Ausgabe-Parameter ein. Für Eingabe-Parameter wird kein zusätzlicher Speicherplatz benötigt, da die Werte der Ausgabe-Parameter mit den Werten der jeweiligen Eingabe-Parameter überschrieben werden.

Eingabe-Parameter oder Ein-Ausgabe-Parameter können aus mehreren Feldelementen eines Vektors oder einer Matrix bestehen.

Aufbau des Parameter-Übergabebereichs

Für die Übergabe der Parameter zwischen rufendem Programm und Unterprogramm richtet DRIVE/WINDOWS einen Verständigungsbereich ein. Dieser Bereich hat eine maximale Länge von 31 KByte. Er ist unterteilt in einen Parameter-Adreßbereich und einen Parameter-Wertebereich. Beide Bereiche haben keine vorgegebene Größe, sondern sind abhängig von der Anzahl und der Datenwertlänge der in der USING-Klausel angegebenen Parameter.

– Parameter-Adreßbereich:
  Für jeden Parameter der USING-Klausel wird im Parameter-Adreßbereich eine Adresse abgelegt. Sie weist auf die entsprechende Stelle im Parameter-Wertebereich, ab der der Datenwert des Parameters abgelegt wird.

– Parameter-Wertebereich:
  Die Parameterwerte im Parameter-Wertebereich werden auf Doppelwortgrenze ausgerichtet. Besteht ein Parameter aus mehreren Komponenten (z.B. Vektor, Matrix, strukturierte Variable), wird der erste Komponentenwert des Parameters auf Doppelwortgrenze ausgerichtet; die Datenwerte aller nachfolgenden Parameterkomponenten folgen unausgerichtet dahinter.

NULL-Werte mit der Indikatorvariablen ablegen

Wird ein Parameter in der USING-Klausel zusammen mit der Angabe INDICATOR angegeben, wird eine Variable, die „Indikatorvariable“, als zusätzlicher Parameter angelegt. Im Parameter-Adreßbereich unmittelbar hinter der Adresse auf den Parameterwert wird eine Adresse auf den Indikatorwert abgelegt. Der Indikatorwert gibt an, ob der zugehörige Parameterwert als NULL-Wert zu interpretieren ist oder nicht.

Indikatorwerte werden auf Doppelwortgrenze ausgerichtet. Besteht ein Parameter aus mehreren Komponenten, wird für jede Komponente ein Indikatorwert im Parameter-Wertebereich abgelegt. In diesem Fall wird der Indikatorwert zum ersten Komponentenwert auf Doppelwortgrenze ausgerichtet; die Indikatorwerte aller nachfolgenden Parameterkomponenten folgen unausgerichtet dahinter.

Hat ein Parameter in der USING-Klausel zum Zeitpunkt der Übergabe den Wert NULL, wird zwar im Parameter-Wertebereich Platz für den eigentlichen Parameterwert reserviert, dieser aber nicht mit einem Wert versorgt. Fehlt in einem solchen Fall gar die Angabe einer INDICATOR-Klausel, wird das Programm abgebrochen.

Aufbau einer Indikatorvariablen

Als Indikatorvariable wird eine Variable der Länge 2 Byte vom Datentyp SMALLINTEGER abgelegt. Ist der Indikatorwert negativ, ist der zugehörige Parameterdatenwert als NULL-Wert zu interpretieren.

Beispiel

Die Variablen &NAME und &PROJEKT sollen an ein externes Unterprogramm übergeben werden. Mit der Angabe INDICATOR in der USING-Klausel wird im Parameterübergabebereich zusätzlich zu jeder Variablen eine Indikatorvariable abgelegt.

```
SET &NAME = 'SCHMIDT';
SET &PROJEKT = NULL;
CALL MODULE modul USING &NAME INDICATOR, &PROJEKT INDICATOR;
```

Endekennzeichen für den Übergabebereich

Für COBOL und Assembler gilt: In der Werteadresse des letzten Parameters der USING-Klausel wird das höchstwertigste Bit als Endekennzeichen auf 1 gesetzt (wird der letzte Parameter mit einer INDICATOR-Klausel angegeben, wird das höchstwertigste Bit der zugehörigen Indikatorwertadresse auf 1 gesetzt).

Beispiel für den Aufbau eines Übergabebereichs

Ausschnitt aus dem DRIVE-Programm:

```
DCL VAR &I    INT,
        &D(3) DEC(5,2),
        &C    CHAR,
        1 &S,
          2 A1,
            3 B1 NUM(10,3),
            3 B2 DATE,
          2 A2,
            3 B3 SMALLINT,
            3 B4 TIME;

CALL MODULE testprog USING RET &I, &D IND, &C IND, RET &S IND;
```

Aufbau des Übergabebereichs:
Die Striche (——) bedeuten eine Ausrichtung auf Doppelwortgrenze.

Werte-Adresse von &I
Werte-Adresse von &D
Indikator-Adresse von &D
Werte-Adresse von &C
Indikator-Adresse von &C
Werte-Adresse von &S
Indikator-Adresse von &S
Wert von &I
Wert von &D(1)
Wert von &D(2)
Wert von &D(3)
Indikatorwert von &D(1)
Indikatorwert von &D(2)
Indikatorwert von &D(3)
Wert von &C
Indikatorwert von &C
Wert von &B1
Wert von &B2
Wert von &B3
Wert von &B4

Indikatorwert von &B1
Indikatorwert von &B2
Indikatorwert von &B3
Indikatorwert von &B4

4.9.2 Externe Unterprogramme in C

Der Name des externen Unterprogramms in C darf kein reserviertes Wort von C oder Schlüsselwort von DRIVE/WINDOWS sein. Der Name darf nicht mit „idr“ oder „dri“ beginnen.

An der Schnittstelle des C-Programms werden Zeiger auf die entsprechenden Datentypen der Parameter erwartet.

Felddefinitionen im Unterprogramm

Die Formate der Übergabe-Parameter des rufenden Programms müssen mit den entsprechenden Felddefinitionen im Unterprogramm übereinstimmen (siehe folgende Tabelle). Parameter der DRIVE-Datentypen NUM, DEC oder XDEC sollten Sie im DRIVE-Programm einer Hilfsvariablen vom Typ DOUBLE PRECISION zuweisen und die Hilfsvariable an das C-Programm übergeben, da dieser Datentyp im C-Programm leichter zu bearbeiten ist.

DRIVE-Formate	**C-Felddefinitionen**
par1 CHAR	char *par1
par1 CHAR(n)	char *par1 oder char par1[n+1] oder char *par1[n+1] oder char par1[ ]
par1 NUM	double *par1
par1 DEC	double *par1
par1 XDEC	double *par1
par1 INTEGER	long *par1
par1 SMALLINT	short *par1
par1 VARCHAR(n)	char *par1
par1 INTERVAL	long *par1
par1 REAL	float *par1
par1 DOUBLE PRECISION / FLOAT	double *par1

DRIVE-Formate	C-Felddefinitionen
par1 DATE	char par1[11]
par1 TIME	char par1[9]
par1 TIME(3)	char par1[13]
par1 TIMESTAMP(3)	char par1[24]

Übergabe von CHARACTER-, VARCHAR- und Zeit-Datentypen

– In allen drei Fällen wird die Adresse der DRIVE-Variablen übergeben.

– Bei CHARACTER und Zeit-Datentypen steht ab der angegebenen Adresse die Zeichenkette in ihrer maximale Länge (evtl. mit Leerzeichen gefüllt). Das Ganze wird mit '\0' abgeschlossen.

– Ein mit RETURN spezifizierter VARCHAR-Typ wird in maximaler Länge übergeben, wobei der Wert mit '\0' abgeschlossen wird. Das Längenfeld wird nicht mit in das C-Programm übergeben.

– Bei einem VARCHAR ohne RETURN wird nur die aktuelle Länge übergeben. Den Abschluß bildet das NULL-Zeichen. Das Längenfeld wird nicht mit in das C-Programm übergeben.

Bei der Übergabe von Werten und Parametern sollte das C-Unterprogramm den C-String nicht verändern.
Zeichenketten werden in C ab der Position Null, in DRIVE/WINDOWS dagegen ab Position Eins indiziert.

Jeder C-String sollte mit dem NULL-Zeichen abgeschlossen werden ('\0', d.i. binär 0), insbesondere dann, wenn er wieder an das DRIVE-Programm zurückgegeben wird.

Bei der Übergabe von DRIVE-Datengruppen (Strukturen, Vektoren, Matrizen) wird die Kompatibilität zu entsprechenden C-Datentypen nicht gewährleistet.

Bei Verwendung des DRIVE-Datentyps INTEGER sollte der zugehörige C-Parameter vom Typ `long` sein. Den maschinenabhängigen C-Datentyp `int` dürfen Sie für DRIVE-INTEGER-Parameter nur verwenden, wenn sichergestellt ist, daß `int` 4 byte lang ist.

Beispiel

DRIVE-Programm:

```
PROC A;
DCL VAR &INT1 INT;
DCL VAR &CHAR1 VARCHAR(12);
...
...
SET &INT1 = 2;
SET &CHAR1 = 'Hello ';
CALL C MODULE cprogramm USING RETURN &INT1,&CHAR1;
...
...
END PROC;
```

C-Programm:

```
cprogramm(long *int1,char *char1)
{
long a;
 ...
 ...
 a = strcat(char1,"World");
 *int1 = *int1 + 5;
}
```

4.9.3 Externe Unterprogramme in COBOL

Felddefinitionen im Unterprogramm

Die Formate der Übergabe-Parameter des rufenden Programms müssen mit den entsprechenden Felddefinitionen im Unterprogramm übereinstimmen.

DRIVE-Formate	**COBOL-Felddefinitionen**
par1 CHAR	pic x
par1 CHAR(n)	pic x(n)
par1 VARCHAR	pic x(n)
par1 DECIMAL(n,m)	pic S9(n-m)V9(m) comp-3
par1 XDEC(n,m)	pic S9(n-m)V9(m) comp-3
par1 NUMERIC(n,m)	pic S9(n-m)V9(m)
par1 INTEGER	pic S9(8) comp-5

DRIVE-Formate	COBOL-Felddefinitionen
par1 INTERVAL	pic S9(8) comp-5
par1 SMALLINT	pic S9(4) comp-5
par1 REAL	comp-1
par1 FLOAT / DOUBLE PRECISION	comp-2
par1 DATE	pic x(10)
par1 TIME	pic x(8)

Bei der Übergabe von DRIVE-Datengruppen (Strukturen, Vektoren, Matrizen) wird die Kompatibilität zu entsprechenden C-Datentypen nicht gewährleistet.

Um den maximalen Wertebereich von Übergabe-Parametern der Formate INTEGER oder SMALLINT auszunutzen, muß beim Übersetzen des Unterprogramms der COBRUN-Parameter TRUNCATE=NO angegeben werden.

Ohne Angabe dieses COBRUN-Parameters liegt der maximale Wert

– für Übergabe-Parameter im Format INTEGER bei 999999999,
– für Übergabe-Parameter im Format SMALLINT bei 9999.

4.9.4 Externe Unterprogramme in ASSEMBLER

Programme können nur ASSEMBLER-Unterprogramme aufrufen, die hinsichtlich der Belegung von Registern folgenden Konventionen genügen:

Register 1 Adresse der Parameterleiste.

Die Parameterleiste enthält die Anfangsadressen aller Parameter, die zwischen dem rufenden Programm und dem externen Unterprogramm ausgetauscht werden.
Innerhalb der Parameterleiste beginnt im NXS-Betrieb die letzte Anfangsadresse, wie bei COBOL, mit X'80' (d.h. das höchstwertige Bit der letzten Anfangsadresse ist auf 1 gesetzt).
Im XS-Betrieb muß es nicht zwingend X'80' sein.

Register 13 Adresse des Sicherstellungsbereichs.

Der Sicherstellungsbereich wird von dem rufenden Programm bereitgestellt. Er dient dazu, die Registerinhalte des rufenden Programms zu sichern.

Register 14 Rücksprungadresse.

Register 14 enthält beim Aufruf eines externen Unterprogramms die Adresse für den Rücksprung in das rufende Programm.

Register 15 Einsprungadresse.

Register 15 enthält die Einsprungadresse des aufgerufenen externen Unterprogramms.

Felddefinitionen im Unterprogramm

Die Formate der Übergabe-Parameter des rufenden Programms müssen mit den entsprechenden Felddefinitionen im Unterprogramm übereinstimmen (siehe folgende Tabelle):

DRIVE-Formate	**ASSEMBLER Konstanten-Definitionen**
par1 CHAR	par1 DS CL1
par1 CHAR(n)	par1 DS CLn
par1 VARCHAR(n)	par1 DS CLn
par1 NUM(n,m)	par1 DS CLn
par1 DEC(n,m)	par1 DS PLn
par1 XDEC(n,m)	par1 DS PLn
par1 INTEGER	par1 DS F
par1 SMALLINT	par1 DS H
par1 REAL	par1 DS E
par1 FLOAT	par1 DS D
par1 DATE	par1 DS CL10
par1 TIME	par1 DS CL8
par1 INTERVAL	par1 DS F

4.9.5 Externe Unterprogramme testen

Unterprogramme sollten nur im TIAM-Betrieb getestet werden. Im UTM-Betrieb führen Fehler bei der Parameter-Übergabe zu einem Abbruch der UTM-Anwendung.

Im TIAM-Betrieb können bei Unterprogramm-Aufrufen folgende Fehlersitutationen eintreten:

– Das rufende Programm wird unterbrochen. An der Datensichtstation werden folgende Meldungen ausgegeben:

```
BLS0334 SYMBOL 'modul' CANNOT BE FOUND. LOADING ABORTED.

BLS0334 SYMBOL 'modul' CANNOT BE FOUND. LOADING ABORTED.

BLS0334 SYMBOL 'modul' CANNOT BE FOUND. LOADING ABORTED.

BLS0334 SYMBOL 'modul' CANNOT BE FOUND. LOADING ABORTED.

BLS0334 SYMBOL 'modul' CANNOT BE FOUND. LOADING ABORTED.
```

Ursache:

Der aufgerufene Unterprogramm-Modul 'modul' befindet sich in keiner der von DRIVE/WINDOWS durchsuchten Programm-Bibliotheken.

Maßnahme:

Der Modulbibliothek, die das aufgerufene Unterprogramm enthält, muß der Dateikettungsname USEROML zugewiesen werden.

– Nach dem Aufruf eines COBOL- oder C-Unterprogramms wird das rufende Programm unterbrochen. An der Datensichtstation werden folgende Meldungen ausgegeben:

```
BLS0335 UNRESOLVED EXTERNAL REFERENCE "modul1[,modul2,...]" BLS0336 CONTINUE
PROCESSING? REPLY (Y=YES; N=NO)
```

Nach Eingabe von „Y“ oder „N“ werden die Inhalte der Register an der Datensichtstation ausgegeben und ein Dump erzeugt.

Ursache:

Das COBOL-Laufzeitsystem ist weder statisch mit dem aufgerufenen Unterprogramm zusammengebunden, noch befindet es sich in einer vom dynamischen Bindelader (DLL) durchsuchten Modulbibliothek.

Maßnahmen:

1. Das Unterprogramm muß statisch mit dem entsprechenden Laufzeitsystem zusammengebunden werden. Dafür kann das Softwareprodukt TSOSLNK verwendet werden.

2. Das entsprechende Laufzeitsystem muß in eine vom dynamischen Bindelader (DLL) durchsuchte Modulbibliothek gebracht werden. Je nachdem, ob bereits mit dem BS2000-Kommando SET−TASKLIB LIB =... eine Programm-Bibliothek als Bindemodul-Bibliothek zugewiesen wurde, bestehen folgende Möglichkeiten:

   – Es wurde bereits eine Programm-Bibliothek als Bindemodul-Bibliothek zugewiesen:

     Mit Hilfe des Softwareproduktes LMS muß das entsprechende Laufzeitsystem in eine vom dynamischen Bindelader (DLL) durchsuchte Modulbibliothek kopiert werden.

   – Es wurde noch keine Programm-Bibliothek als Bindemodul-Bibliothek zugewiesen:

     Mit dem BS2000-Kommando SET TASKLIB LIB =... muß die Modulbibliothek, in der das entsprechende Laufzeitsystem gespeichert ist, als Bindemodul-Bibliothek zugewiesen werden. Alternativ besteht die Möglichkeit, das entsprechende Laufzeitsystem in eine vom dynamischen Bindelader (DLL) durchsuchte Modulbibliothek zu kopieren. Dafür kann das Softwareprodukt LMS verwendet werden.

– Nach dem Aufruf eines Unterprogramms wird das rufende Programm unterbrochen. Die Inhalte der Register werden an der Datensichtstation ausgegeben und ein Dump erzeugt.

  Ursachen:

  – Die Feldbeschreibungen im aufgerufenen Unterprogramm stimmen nicht überein mit Anzahl und Format der Übergabe-Parameter im rufenden Programm.

  – In einem aufgerufenen ASSEMBLER-Unterprogramm werden die Register-Konventionen (siehe Abschnitt „Externe Unterprogramme in ASSEMBLER“ auf Seite 119) nicht eingehalten.

  – Falsche Programmierlogik im Unterprogramm.

  Maßnahme:

  Die Register-Konventionen im aufgerufenen Unterprogramm müssen überprüft werden. Zur Unterstützung bei der Fehlersuche kann das Software-Produkt „Dialogtesthilfe AID“ verwendet werden.

– Das Unterprogramm arbeitet fehlerfrei. Jedoch sind die Werte, die vom Unterprogramm an das rufende Programm zurückgegeben werden, fehlerhaft oder unvollständig.

  Ursache:

  Anzahl oder Formate der Übergabe-Parameter bei CALL des rufenden Programms stimmen nicht mit den Felddefinitionen im Unterprogramm überein.

  Maßnahme:

  Die Felddefinitionen im aufgerufenen Unterprogramm müssen überprüft und die fehlerhaften Felddefinitionen korrigiert werden. Anschließend muß das Unterprogramm neu übersetzt werden.

5 DRIVE-Bildschirmformate

DRIVE/WINDOWS bietet Funktionen, mit denen Sie Bildschirmformate erstellen können. Bildschirmformate programmieren Sie entweder über DRIVE-Anweisungen oder erstellen Sie mit Hilfe von FHS-Formaten.

In diesem Kapitel erfahren Sie, wie Sie:

- DRIVE-Bildschirmformate erstellen und einsetzen (ab Seite 126)
- FHS-Bildschirmformate in DRIVE/WINDOWS einsetzen (ab Seite 143)
- Kurzmeldungen mit Hilfe der Meldungszeile ausgeben (Seite 162)
- NULL-Werte in Formaten darstellen (ab Seite 163)

  und welche

- Einschränkungen es beim Einsatz von IBM-Terminals gibt (ab Seite 166).

5.1 DRIVE-Bildschirmformate erstellen und einsetzen

DRIVE-Bildschirmformate zur Ausgabe und Eingabe von Daten werden innerhalb eines DRIVE-Programms erstellt. Die Formate können Sie somit - ohne DRIVE/WINDOWS zu verlassen - leicht programmieren und, falls erforderlich, ohne großen Aufwand veränderten Anforderungen anpassen.

Die Anweisungen zur Programmierung eines DRIVE-Formats sind:

DECLARE FORM formatname

Die Anweisung legt einen Speicherbereich für ein Bildschirmformat an, definiert dafür einen Rahmen und legt ein Layout (z.B. Kopfzeilen) fest. Sie muß im Deklarationsteil des Programms stehen.

FILL formatname

Die Anweisung füllt den FILL-Bereich (siehe Bild 1) mit Inhalt. Sie kann für beliebige mit DECLARE FORM definierte Formate angegeben werden. Sie muß im Ablaufteil des Programms stehen.

DISPLAY formatname

Die Anweisung schließt eine Ausgabeformatierung ab und stößt eine Ausgabe an. Konkret geschieht folgendes: Die Kopf- und Fußzeilen (siehe Bild 1) werden erst jetzt mit Inhalt gefüllt; dann wird das komplette Format am Bildschirm ausgegeben. Die Anweisung muß im Ablaufteil des Programms stehen.

DRIVE/WINDOWS bietet zusätzlich die Möglichkeit, ein Format zur „ad hoc“-Bildschirmausgabe mitten im Ablaufteil eines Programms einzusetzen. Dieses Bildschirmformat wird im Ablaufteil des Programms mit **einer** Anweisung definiert, gefüllt und ausgegeben:

DISPLAY FORM

In den DRIVE-Handbüchern wird diese Anweisung auch als Kompaktanweisung und das entsprechende Format als Kompakt-Format bezeichnet, falls es aus Gründen der Eindeutigkeit notwendig ist.

Die folgende Darstellung zeigt die DRIVE-Bildschirmformate und die zur Programmierung dafür notwendigen Anweisungen.

Erstellen und Einsetzen eines DRIVE-Bildschirmformats mit den Anweisungen	
DECLARE FORM formatname FILL formatname DISPLAY formatname	DISPLAY FORM
↓ **DRIVE-Format**	↓ **Kompakt-Format** (Sonderform des DRIVE-Formats)

Vorteile des DRIVE-Formats:

– Es können beliebig viele DRIVE-Formate deklariert werden.

– Jedes dieser Formate kann über seinen Formatnamen an jeder gewünschten Stelle des Programms mit Inhalt gefüllt und/oder am Bildschirm ausgegeben werden.

– Der Inhalt des FILL-Bereiches bleibt nach einem DISPLAY bis zum nächsten FILL erhalten und kann daher, falls gewünscht, mehrmals ausgegeben werden.

Vorteile des Kompakt-Formats:

– Es können beliebig viele Kompakt-Formate deklariert werden.

– Das Kompakt-Format wird an beliebiger Stelle des Programms zugleich deklariert, gefüllt und am Bildschirm ausgegeben. Der Inhalt des FILL-Bereiches bleibt allerdings nicht erhalten, da kein entsprechender Speicherbereich definiert ist.

5.1.1 DRIVE-Bildschirmformat definieren

Mit der Anweisung DECLARE FORM (Syntax und Regeln siehe DRIVE-Lexikon [3]) können Sie ein Bildschirmformat zur Ein- und Ausgabe von Daten aufbauen.

Festlegen der Bildschirmdimension

Sie können mit den Angaben COLUMNS und LINES den Umfang des Bildschirms dimensionieren, d.h. die Anzahl der Spalten und Zeilen pro Bildschirmseite festlegen. Der Standardwert für COLUMNS ist die Spaltenanzahl des aktuellen Bildschirms (z.B. für den Bild-

schirm 97801: 80 Spalten). Der Standardwert für LINES ist die Zeilenanzahl des aktuellen Bildschirms - 1 (z.B. für den Bildschirm 97801: 24-1=23 Zeilen). Die letzte Bildschirmzeile ist für die Meldungszeile reserviert (siehe Abschnitt „Meldung am Bildschirm ausgeben“ auf Seite 162).

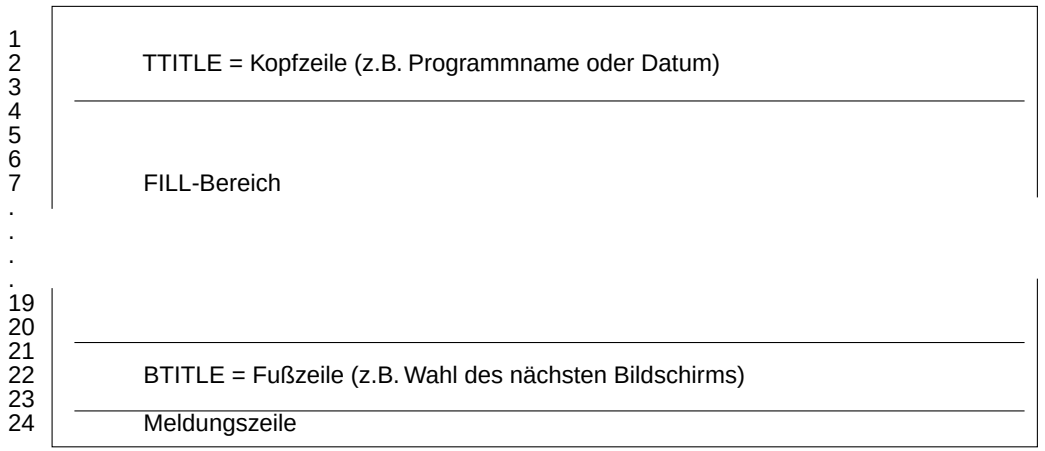
Beispiel

```
DECLARE FORM F1 COLUMNS 60, LINES 12;
```

Eine Bildschirmseite wird auf 60 Spalten und 12 Zeilen dimensioniert.

Festlegen der Kopf- und Fußzeilen

Sie können für das Bildschirmformat Kopf- und Fußzeilen festlegen. Dazu benötigen Sie die Operanden TTITLE bzw. BTITLE. Das folgende Bild zeigt, wie ein Bildschirmformat in Kopf- und Fußzeilen aufgeteilt werden könnte. Der Bereich zwischen Kopf- und Fußzeilen ist der FILL-Bereich. Das folgende Bild zeigt eine Bildschirmaufteilung mit Kopf- und Fußzeilen.



Pro Bildschirm können maximal n_1 TTITLE + n_2 BTITLE + n_3 FILL = Zeilenanzahl des aktuellen Bildschirms (LINES) -1 definiert werden. Die letzte Bildschirmzeile ist als Meldungszeile reserviert. Es gilt:
$1 \leq n_3 \leq$ LINES -1 und $0 \leq n_1, n_2 \leq$ LINES-2. Mindestens eine Zeile ist für den FILL-Bereich vorgesehen. Somit können pro Bildschirm maximal (LINES-2) Zeilen für TTITLE und BTITLE verwendet werden.

Definieren von Ausgabe- und Eingabefeldern (Bildschirmfelder)

„Bildschirmfeld“ ist der Überbegriff für Ausgabe- und Eingabefeld. Ausgabefelder sind nicht überschreibbar.

Sie können in die Kopf- und Fußzeilen Ausgabe- und Eingabefelder setzen. Einschränkungen und weitergehende Regeln lesen Sie bitte im „DRIVE-Lexikon“ [3] bei der Anweisung DECLARE FORM nach.

Ein Auszug aus der Syntaxdarstellung zu DECLARE FORM soll die unten erwähnten Möglichkeiten für das Definieren von Ausgabe- und Eingabefeldern in Kopfzeilen in übersichtlicher Form zusammenfassen (für Fußzeilen analog mit BTITLE):

```
...
TTITLE [ format ] { [ RETURN ] ausdruck [ INIT ausdruck1 [ NOCHECK ] ]
                                        [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |

                      NEWLINE n |
                      TABULATOR n |
                      BLANK n }, ...
...
```

Beim Definieren von Bildschirmfeldern stehen folgende Möglichkeiten zur Verfügung:

– Definieren von Ausgabefeldern

  Sie können Ausgabefelder definieren durch Angabe von *ausdruck* (siehe DRIVE-Lexikon [3], Metavariable *ausdruck*). *ausdruck* kann u.a. sein: Variable, Literal, arithmetischer Ausdruck, Stringfunktion, z.B.

```
DECLARE FORM ARTIKELNEUAUFNAHME
                              ... &AUFGABENART ...
```

– Definieren mehrerer Bildschirmfelder

  Sie können mehrere Bildschirmfelder für ein Format definieren. Es muß ein Komma zwischen den Felddefinitionen stehen, z.B.

```
... &AUFGABENART, &DATE, (&OPERAND + 3)/2, CONCAT (&VARA, &VARB )...
```

– Einsetzen des Wiederholungsfaktors

  Sie können sich Schreibaufwand ersparen, wenn Sie den Wiederholungsfaktor verwenden. Der Wiederholungsfaktor darf nur bei Literalen verwendet werden. (Form: `literal'(n)`, siehe „DRIVE-Lexikon“ [3], Metavariable *literal*).

Der Wiederholungsfaktor *n* ist eine ganze Zahl, die in Klammern unmittelbar hinter dem Literal anzugeben ist, z.B.

```
... '␣'(3), ...
```

– Definieren von Eingabefeldern

  Sie können Eingabefelder definieren durch Angabe von RETURN vor *ausdruck* (siehe DRIVE-Lexikon [3], Metavariable *ausdruck*). *ausdruck* muß eine Variable sein, die nicht mit „.*“ qualifiziert ist. Jede Variable kann pro Bildschirmformat nur einmal als Eingabefeld verwendet werden.

```
... 'BEARBEITER:␣', RETURN &NAME ...
```

  Innerhalb eines Bildschirmformats kann der gleiche Variablenname als Eingabe- und Ausgabefeld verwendet werden.

```
... 'BEARBEITER:␣', RETURN &NAME, &NAME  ...
```

– Initialisieren von Bildschirmfeldern

  Sie können Ausgabe- und Eingabefeldern mit der INIT-Klausel einen Anfangswert zuweisen.

  INIT ist nur zulässig, wenn das Ausgabe- und/oder Eingabefeld eine Variable ist.

  Initialisierungswerte, die für eine Variable von DRIVE/WINDOWS vorbelegt sind, wenn keine INIT-Angabe erfolgt, können Sie nachlesen im DRIVE-Lexikon [3], Anweisung DECLARE VARIABLE.

  Der Initialisierungswert muß ein Literal, NULL oder eine Funktion sein, deren Argumente Literale (aber nicht CURRENT DATE/TIME) sind.

```
... &DB INIT 'ARTIKELDATENBANK' ...
```

  Die INIT-Klausel muß mit der CHECK-Klausel, die bei der Variablendeklaration festgelegt wurde, verträglich sein (siehe DRIVE-Lexik0n [3], Anweisung DECLARE VARIABLE und Metavariable *check*). D.h. der Initialisierungswert für ein Ausgabe- und/oder Eingabefeld muß der Prüfbedingung von CHECK entsprechen.

– NOCHECK-Klausel

  Mit der Angabe NOCHECK können Sie festlegen, daß eine definierte CHECK-Klausel für die Zuweisung des Initialisierungswerts nicht ausgewertet wird.

  NOCHECK ist nicht erlaubt bei redefinierten Variablen oder einer Variablen, die eine andere redefiniert.

– Festlegen von Feldeigenschaften

  Sie können Ausgabe- und Eingabefelder mit Hilfe des Operanden ATTRIBUTE mit Feldeigenschaften belegen, (siehe „DRIVE-Lexikon“ [3], Metavariable *attribute*), z.B. Helligkeitseinstellung:

  ```
  ... 'BEARBEITER:␣', RETURN &NAME ATTRIBUTE (HIGHINTENSITY) ...
  ```

– Definieren der Darstellungsmöglichkeit für Ein- und Ausgabefelder

  Sie können für Ausgabe- und Eingabefelder mit Hilfe des Operanden MASK unterschiedliche Darstellungsarten definieren. Z.B. das Maskensteuerzeichen „X“ für alphanumerische Datentypen, oder die Unterdrückung der Ausgabe von führenden Nullen mit „Z“ (siehe „DRIVE-Lexikon“ [3], Metavariable *mask*), z.B.

  `... &DATE MASK 'ZD''.''ZO''.''YYYY'` ergibt ␣5.␣8.1990

– Positionieren der Bildschirmfelder

  Sie können die Ausgabe- und Eingabefelder innerhalb der Kopf- bzw. Fußzeilen exakt positionieren mit Hilfe von Angaben wie Leerzeichen, Tabulator oder Zeilenvorschub. Mehrere Positionsangaben hintereinander sind möglich. Sie müssen durch ein Komma getrennt sein, z.B.

  ```
  ... NL 1, BLANK 45, ...
  ```
  ergibt eine neue Zeile (NEWLINE), in der die ersten 45 Spalten mit Leerzeichen belegt sind

  Es bestehen Abhängigkeiten bezüglich Angaben zur Positionierung der Bildschirmfelder und Angaben zur Datenanordnung über *format* (siehe folgenden Abschnitt Festlegen der Datenanordnung).

– Ausgeben von Datenbank-Inhalten (siehe Abschnitt „DRIVE-Bildschirmformat mit Inhalt füllen“ auf Seite 134). Bei einem Bildschirmüberlauf wird der TTITLE als Seitenkopf und der BTITLE als Seitenfuß ausgegeben.

  Es erfolgt zum Ausführungszeitpunkt ein Prozedurabbruch, wenn sich die TITLE-Angaben in ihrer Gesamtlänge zwischen zwei Ausgaben ändern.

Beispiel

```
DCL &v VARCHAR(10) INIT '12345';
...
DCL FORM F1 TTITLE SUBSTRING(&v,1);
...
DISPLAY F1; ------------------------------------------------------------------------------   (1)
...
SET &v='1234567';
...
DISPLAY F1; ------------------------------------------------------------------------------   (2)
```

(1) Gesamtlänge von TITILE: 5

(2) Gesamtlänge von TTITLE: 7 führt zu Prozedurabbruch

Festlegen der Datenanordnung

Mit DECLARE FORM können Sie für die Daten der Kopf- und Fußzeilen bestimmen, wie diese am Bildschirm anzuordnen sind, z.B. eine tabellenförmige Ausgabe der Bildschirminhalte. DRIVE/WINDOWS bietet die Wahl zwischen Standardformaten und freier Layoutsteuerung (siehe „DRIVE-Lexikon“ [3], Metavariable *format*).

Die Standardformate können Sie folgendermaßen definieren:

– Ausgabe erfolgt tabellenförmig (Operand TABLE)
– Ausgabe erfolgt zeilenweise (Operand LINE)
– Ausgabe erfolgt hintereinander angeordnet (Operand SEQUENCE)

Eine freie Layoutsteuerung definieren Sie mit dem Operanden FREE.

Bei Standardformaten und bei freier Layoutsteuerung können Sie zusätzlich beeinflussen, ob folgendes ausgegeben werden soll:

– Variablennamen (Operand NAMES)
– Dateninhalte bzw. -werte der auszugebenden Elemente (Operand VALUES)
– beides (NAMES und VALUES)

Durch Kombinieren dieser Operanden können Sie sich Ihr gewünschtes Layout gestalten. Beispiele zu diesen Operanden finden Sie im Abschnitt „DRIVE-Bildschirmformat mit Inhalt füllen“ auf Seite 134.

Die freie Layouterstellung erfolgt mit dem Operanden FREE, der standardmäßig vorbelegt ist: Die Dateninhalte bzw. -werte der auszugebenden Elemente werden lückenlos hintereinander ausgegeben, ohne Rücksicht auf das Zeilenende, d.h. bei Zeilenende erfolgt Umbruch. Die Variablennamen werden nicht ausgegeben.

Es bestehen folgende Abhängigkeiten bezüglich Angaben zur Positionierung der Bildschirmfelder und Angaben zur Datenanordnung über *format*:

– Wird für *format* TABLE angegeben, sind nachfolgend die Angaben NEWLINE und NEWPAGE nicht erlaubt.

– Wird für *format* LINE angegeben, erfolgt bei den Angaben TABULATOR und BLANK ein Zeilenvorschub mit Leerzeile.

Festlegen eines Zeichens für die NULL-Wert-Darstellung

Sie können durch Angabe von NULL *nullwert* definieren, wie der NULL-Wert im Format dargestellt werden soll. Eine in der PARAMETER-Anweisung vereinbarte Nullwertdarstellung wird hiermit überschrieben. Sie können für ein alphanumerisches und/oder numerisches

Datenfeld je ein NULL-Wertzeichen festlegen. Das NULL-Wertzeichen wird als alphanumerisches Literal der Länge 1 bestimmt (siehe DRIVE-Lexikon [3], Metavariable *nullwert*), z.B.

```
DECLARE FORM F1 NULL CHARTYPE='␣' NUMTYPE='0'...
```

Festlegen der Lebensdauer eines DRIVE-Formats

In einem internen Unterprogramm (mit CALL aufgerufen) wird ein DRIVE-Format definiert:

– Der Inhalt eines mit PERMANENT definierten Formats bleibt über Unterprogrammende hinaus erhalten.

– Der Inhalt eines mit TEMPORARY definierten Formats bleibt bei Unterprogrammende nicht erhalten.

Beispiel: Definieren des DRIVE-Formats ARTIKELNEUAUFNAHME

```
PROCEDURE INSFORM;
...
/* VARIABLENDEKLARATION */
DECLARE VAR &DB CHAR (16),
            &AUFGABENART CHAR (15) INIT 'NEUAUFNAHME',
            &NAME CHAR (20),
            &DATUM DATE MASK 'ZD''.''ZO''.''YYYY',
           ...
            &E CHAR (1) INIT 'N'
               CHECK &E = 'J' OR &E = 'N';
/* DEKLARATION DES DRIVE-FORMATS */
DECLARE FORM ARTIKELNEUAUFNAHME PERMANENT
        TTITLE NL 1,
               '-'(80),
              '␣'(3), &DB INIT 'ARTIKELDATENBANK', TAB 70, &DATUM
```

```
                NL 1,'␣'(3), &AUFGABENART,
                NL 1, '␣'(3), 'BEARBEITER:␣', RETURN &NAME
                                    ATTRIBUTE (HIGHINTENSITY),
                NL 1, '-'(80),
                NL 1
        BTITLE NL 2,
                '-'(80),
                '␣'(3), 'ENDE (J/N):␣', RETURN &E INIT 'N',
                NL 1, '-'(80),
                NL 1;
/* ENDE DER FORMATDEKLARATION */
...
```

Die Definition des DRIVE-Formats mit dem Namen ARTIKELNEUAUFNAHME legt den folgenden Speicherbereich für ein Bildschirmformat an:

```
      123...  (Spalten)
1
2     ____________________________________________________________________________
3       ARTIKELDATENBANK                                                 1.1.1996
4       NEUAUFNAHME
5       BEARBEITER:
6     ____________________________________________________________________________
7
.
.
.
.
19
20    ____________________________________________________________________________
21      ENDE (J/N): N
22    ____________________________________________________________________________
23
24    Meldungszeile
```

5.1.2 DRIVE-Bildschirmformat mit Inhalt füllen

Voraussetzung für das Füllen eines DRIVE-Formats mit Inhalt ist dessen Definition (siehe Abschnitt „DRIVE-Bildschirmformat definieren“ auf Seite 127).

FILL formatname füllt den FILL-Bereich des Speicherbereichs mit Inhalt (Syntax und Regeln siehe „DRIVE-Lexikon“ [3], Anweisung FILL formatname). Diese Anweisung können Sie für beliebig viele mit DECLARE FORM definierte Formate angeben. Jedes FILL leitet eine neue Zeile ein.

Definieren von Ausgabe- und Eingabefeldern

Mit FILL formatname können Sie in den FILL-Bereich Ein- und Ausgabefelder setzen. Mit dieser Anweisung gestaltete Bildschirmformate bestehen somit aus nicht überschreibbaren Ausgabefeldern und überschreibbaren Ausgabe-Eingabefeldern.

Die Syntaxdarstellung zu FILL formatname soll die Möglichkeiten für das Definieren von Ausgabe- und Eingabefeldern im FILL-Bereich in übersichtlicher Form zusammenfassen:

```
FILL formatname [ format ]

        { [ RETURN ] ausdruck [ INIT ausdruck1 [ NOCHECK ] ]
            [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |
          NEWLINE n |
          NEWPAGE n |
          TABULATOR n |
          BLANK n }, ...
```

Beim Definieren von Bildschirmfeldern im FILL-Bereich stehen die gleichen Möglichkeiten zur Verfügung wie beim Definieren von Ausgabe- und Eingabefeldern in den Kopf- und Fußzeilen (siehe Abschnitt „DRIVE-Bildschirmformat definieren“ auf Seite 127). Deshalb wird an dieser Stelle nicht noch einmal darauf eingegangen, mit Ausnahme des nun folgenden Aspekts:

Ausgeben von Datenbank-Inhalten

Datenbank-Inhalte, also Datenbankfelder, können nicht über ihren Namen in einem Format ausgegeben werden. Es müssen Variablen definiert werden, die die gewünschten Werte der Datenbankfelder enthalten. Dies geschieht folgendermaßen: Eine Variable bekommt den Inhalt eines Cursors zugewiesen. (Eine strukturierte Variable kann ja wie ein Cursor aufgebaut sein.)

- Hat die FILL-Anweisung einen Bildschirmüberlauf zur Folge, so wird implizit eine DISPLAY-Anweisung ausgeführt (siehe Abschnitt „DRIVE-Bildschirmformat auf dem Bildschirm ausgeben“ auf Seite 139).
- Eingabefelder dürfen nur für eine Bildschirmseite verwendet werden. Sie müssen darauf achten, daß bei mehr als einem Eingabefeld auf einer Seite bei einem Bildschirmüberlauf Eingabefelder auf die nächste Bildschirmseite „rutschen“ können. DRIVE/WINDOWS meldet dann einen Fehler.

- Paßt eine Ausgabevariable (ohne RETURN) nicht auf eine Bildschirmseite, wird sie zur Ausgabe abgeschnitten. Die letzten drei Zeichen der Ausgabevariable werden mit „>>>“ gekennzeichnet.
- Paßt eine Eingabevariable (mit RETURN) nicht auf ein Bildschirmseite, erfolgt zum Ausführungszeitpunkt Prozedurabbruch.

Beispiele

Cursorinhalt einer Variable zuweisen und auf Bildschirm ausgeben.

```
...
DECLARE VAR &CURVAR LIKE CURSOR C1;
DECLARE FORM ARTIKELNEUAUFNAHME
        TTITLE NL 3,
               '␣'(80), &DB INIT 'ARTIKELDATENBANK' ...
        ...
FETCH C1 INTO &CURVAR.*;
...
/* AUSGABE DIESER VARIABLENWERTE IN TABELLENFORM */
FILL ARTIKELNEUAUFNAHME TABLE &CURVAR;
...
DISPLAY ARTIKELNEUAUFNAHME;
...
```

Der Wert der Variablen &ARTIKEl_NR soll in einem gegen Überschreiben geschützten Bildschirmfeld ausgegeben werden.

```
FILL f1 &ARTIKEL_NR;
```

Der Wert der Variablen &ARTIKEL_NR soll in einem überschreibbaren Bildschirmfeld ausgegeben werden. Der evtl. im Bildschirmformat überschriebene Wert soll wieder der Variablen &ARTIKEL_NR zugewiesen werden.

```
FILL f1 RETURN &ARTIKEL_NR;
```

Der Wert A00130 soll in einem überschreibbaren Bildschirmfeld ausgegeben werden. Der evtl. im Bildschirmformat überschriebene Wert soll der Variablen &ARTIKEL_NR zugewiesen werden.

```
FILL f1 RETURN &ARTIKEL_NR INIT 'A00130';
```

Festlegen der Datenanordnung

Mit FILL formatname können Sie die Anordnung der Daten im FILL-Bereich definieren und gleichzeitig füllen. DRIVE/WINDOWS bietet die Wahl zwischen Standardformaten und freier Layoutsteuerung (siehe „DRIVE-Lexikon“ [3], Metavariable *format*). Die Standardformate können Sie folgendermaßen definieren:

– Ausgabe erfolgt tabellenförmig (Operand TABLE)
– Ausgabe erfolgt zeilenweise (Operand LINE)
– Ausgabe erfolgt hintereinander angeordnet (Operand SEQUENCE)

Eine freie Layoutsteuerung definieren Sie mit dem Operanden FREE.

Bei Standardformaten und bei freier Layoutsteuerung können Sie zusätzlich beeinflussen, ob folgendes ausgegeben werden soll:

– Variablennamen (Operand NAMES)
– Dateninhalte der auszugebenden Elemente, also der Variablen (Operand VALUES)
– beides (NAMES + VALUES)

Durch Kombinieren dieser Operanden können Sie Ihr gewünschtes Layout gestalten. Die Voreinstellung ist FREE.

Beispiele

Sowohl Variablennamen als auch Datenwerte sollen in Tabellenform ausgegeben werden (eine Programmschleife sorgt dafür, daß mehrere Sätze gelesen werden):

```
CYCLE ...
FILL f1 TABLE NAMES VALUES &ADRESSE ...
END CYCLE;
```

```
NAME_____________VORNAME_________STRASSE_________
HUBER            MAX             AUENSTR.
NAME_____________VORNAME_________STRASSE_________
MEIER            GERHARD         SPESSARTSTR.
NAME_____________VORNAME_________STRASSE_________
SEILER           FRANZ-GEORG     OBERE_MUEHLRADS
...
```

Diese Darstellungsform erzeugt zu den Datenwerten jeweils auch die Variablennamen (≅ Überschriftenzeile). Der Name der „Oberen Mühlradstraße“ wird nicht ganz ausgegeben, da die Variable &ADRESSE nur 15 Zeichen lang ist.

Wollen Sie mehrere Werte unter einer einzigen Überschriftenzeile, müssen Sie NAMES und VALUES außerhalb der Schleife angeben:

```
FILL f1 TABLE NAMES &ADRESSE ...
CYCLE ...
FILL f1 TABLE VALUES &ADRESSE ...
END CYCLE;
```

```
NAME_____________VORNAME_________STRASSE_________
HUBER            MAX             AUENSTR.
MEIER            GERHARD         SPESSARTSTR.
SEILER           FRANZ-GEORG     OBERE_MUEHLRADS
...
```

Nur Variablennamen sollen zeilenweise ausgegeben werden:

```
FILL f1 LINE NAMES &ADRESSE ...
```

```
NAME ␣:
VORNAME ␣:
STRASSE ␣:
```

Sowohl Variablennamen als auch Datenwerte sollen hintereinander (horizontal) ausgegeben werden:

```
FILL f1 SEQUENCE NAMES VALUES &ADRESSE ...
```

```
NAME ␣:␣HUBER       ␣VORNAME␣:␣MAX            ␣STRASSE␣:␣AUENSTR.
NAME ␣:␣MEIER       ␣VORNAME␣:␣GERHARD        ␣STRASSE␣:␣SPESSARTSTR.
NAME ␣:␣SEILER      ␣VORNAME␣:␣FRANZ-GEORG    ␣STRASSE␣:␣OBERE_MUEHLRADS
```

Die freie Layouterstellung erfolgt mit dem Operanden FREE, der standardmäßig vorbelegt ist:

Die Dateninhalte bzw. -werte der auszugebenden Elemente (≅ Variablen) werden lükkenlos hintereinander ausgegeben, ohne Rücksicht auf das Zeilenende. Die Variablennamen werden nicht ausgegeben:

```
FILL f1 FREE ...
```

```
HUBER          MAX          AUENSTR.        MEIER       GERHARD
  SPESSARTSTR.   SEILER         FRANZ-GEORG      OBERE_MUEHLRADS ...
```

5.1.3 DRIVE-Bildschirmformat auf dem Bildschirm ausgeben

DISPLAY formatname gibt das DRIVE-Format am Bildschirm aus. Nach erfolgter Ausgabe wird der Speicher für das ausgegebene DRIVE-Bildschirmformat nicht gelöscht. Ein Format kann somit mehrmals hintereinander ausgegeben werden, wenn zwischen den DISPLAY-Anweisungen kein neues FILL-Kommando gegeben wurde.

```
DISPLAY formatname;
```

Nach dieser Anweisung erfolgt die tatsächliche Ausgabe des mit DECLARE FORM definierten und in der Regel mit FILL formatname gefüllten DRIVE-Formats auf dem Bildschirm.

Beispiel: Programm INSFORM für ein DRIVE-Format zur Aufnahme von Datensätzen

```
PROCEDURE INSFORM;
...
/* VARIABLENDEKLARATION */
DECLARE VAR &DB CHAR (16),
            &AUFGABENART CHAR (15) INIT 'NEUAUFNAHME',
            &NAME CHAR (20),
            &E CHAR (1),
          1 &AUFNAHME,
             2 AARTIKELNR CHAR(6),
             2 ATITEL CHAR(30),
             2 APREIS NUM (7,2) MASK 'ZZZZ9P99'' DM''',
             2 ASPRACHE CHAR(3),
             2 ASEITEN NUM(4) MASK 'ZZZ9',
             ...
```

```
            &DML_STATE CHAR(16) INIT' OK',;
DECLARE C1 CURSOR FOR S * FROM DBART;
/* DEKLARATION DES DRIVE-FORMATS */
DECLARE FORM ARTIKELNEUAUFNAHME
        TTITLE '␣'(80),
               '-'(80),
               '␣'(3), &DB INIT 'ARTIKELDATENBANK' TAB 70,
                                 &DATE MASK 'ZD''.''ZO''.''YYYY',
               NL 1, '␣'(3), &AUFGABENART,
               NL 1, '␣'(3), 'BEARBEITER:␣', RETURN &NAME
                                 ATTRIBUTE (HIGHINTENSITY),
               NL 1, '-'(80),
               NL 1,
               '   ARTIKEL_NR  :␣', RETURN &AARTIKELNR, NL1,
               '   TITEL       :␣', RETURN &ATITEL, NL1,
               '   PREIS       :␣', RETURN &APREIS, NL1,
               '   SPRACHE     :␣', RETURN &ASPRACHE, NL1,
               '   SEITEN      :␣', RETURN &ASEITEN, NL1,
               ...
        BTITLE NL 2,
               ...
/* ENDE DER FORMATDEKLARATION */
...
SUBPROC SATZNEUAUFNAHME;
...
 CYCLE;
   DISPLAY ARTIKELNEUAUFNAHME;
   IF &E = 'J'
    THEN BREAK SUBPROCEDURE;
   END IF;
   INSERT INTO DBART VALUES ( &AUFNAHME.* );
 END CYCLE;
END SUBPROC;
...
COMMIT WORK;
END PROC;
```

Und so sieht das erzeugte DRIVE-Format auf dem Bildschirm aus:

```
_____________________________________________________________________________

   ARTIKELDATENBANK                                                 1.1.1996
   NEUAUFNAHME
   BEARBEITER:␣
_____________________________________________________________________________

   ARTIKEL_NR  :␣
   TITEL       :␣
   PREIS       :␣
   SPRACHE     :␣
   SEITEN      :␣
   AUTOR       :␣
   VERLAG      :␣
   UEBERSETZUNG:␣

_____________________________________________________________________________

   ENDE (J/N): N
_____________________________________________________________________________
```

5.1.4 Kompakt-Bildschirmformat erstellen und einsetzen

DRIVE/WINDOWS bietet Ihnen die Möglichkeit, Formate im Ablaufteil eines Programms ad hoc zu definieren, zu füllen und auszugeben mit der Anweisung DISPLAY FORM (Syntax und Regeln siehe „DRIVE-Lexikon“ [3], Anweisung DISPLAY FORM). Diese Anweisung enthält implizit die Anweisungen DECLARE FORM und FILL formatname. Statt drei Anweisungen genügt somit eine. Das Kompakt-Format ist eine Sonderform des DRIVE-Formats.

DISPLAY FORM ist vor allem dann sinnvoll einzusetzen, wenn einmalige Bildschirmausgaben erwünscht sind. Ein einmal ausgegebenes einfaches Format kann nicht noch einmal eingesetzt werden, außer es wird nochmals mit DISPLAY FORM definiert und ausgegeben. Es existiert ja kein Formatname.

Beispiel

```
DISPLAY FORM
        NL 3, TAB 10, 'DER ARTIKELSATZ MIT DER ARTIKEL_NR ',
        NL 1, TAB 25, &VGL ATTRIBUTE (SIGN),
        NL 1, TAB 20, 'WURDE GELOESCHT!´;
```

Darstellung auf dem Bildschirm:

Der Aufbau eines Kompakt-Formats erfolgt nach den gleichen Regeln wie beim DRIVE-Format. Beachten Sie, daß die Kompaktanweisung DISPLAY FORM implizit ein DECLARE und FILL enthält.

5.1.5 Verhalten von DRIVE/WINDOWS bei fehlerhaften Bildschirm-Eingaben

Bei fehlerhaften Bildschirm-Eingaben wird das fehlerhafte Format so lange wieder ausgegeben, bis die Eingabe ohne Fehler ist. Die fehlerhaften Felder sind markiert, abhängig von der ERRORATTRIBUTE-Angabe bei PARAMETER (siehe DRIVE-Lexikon [3], Anweisung PARAMETER DYNAMIC und Metavariable *attribute*).

Beispiel

Als Attribute (≅ Eigenschaften) zur Kennzeichnung fehlerhafter Bildschirmfelder werden zugewiesen:

```
PARAMETER DYNAMIC ERRORATTRIBUTE = (HIGHINTENSITY, UNDERLINE);
```

Fehlerhaft eingegebene Bildschirmfelder sind mit hoher Helligkeitseinstellung und Unterstrich gekennzeichnet.

5.2 FHS-Formate einsetzen

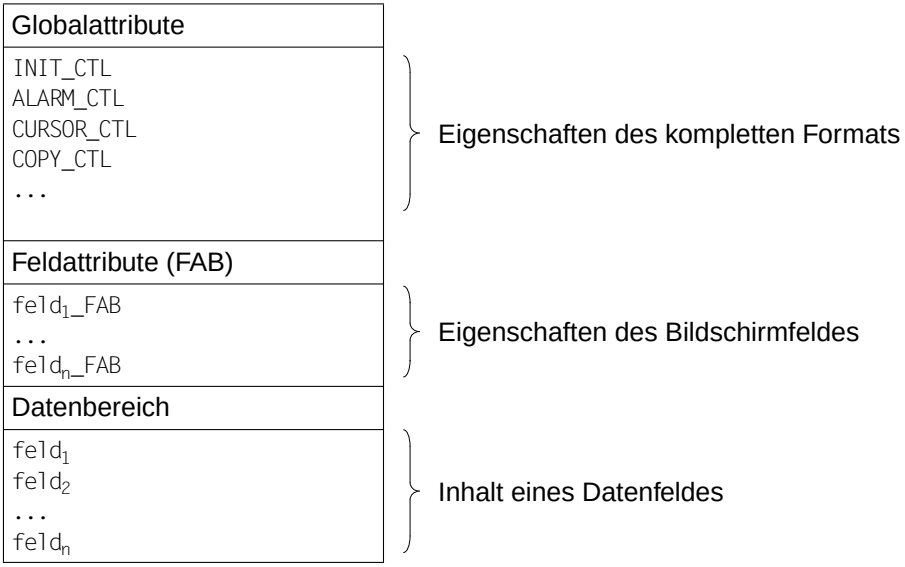
Dieser Abschnitt beschreibt, wie Sie FHS-Formate innerhalb eines DRIVE-Programms verwenden können. Sie erfahren folgendes:

- Wichtige IFG/FHS-Begriffe für den Einsatz in DRIVE/WINDOWS
- Vorbereiten des FHS-Formats für den DRIVE-Einsatz
- Einsetzen des FHS-Formats in einem DRIVE-Programm
- DRIVE-Verhalten bei fehlerhaften Feldeingaben

5.2.1 Wichtige IFG/FHS-Begriffe für den Einsatz in DRIVE/WINDOWS

In diesem Abschnitt sind die wichtigsten Begriffe zum Interaktiven Formatgenerator (IFG) und zum Format-Handling-System (FHS) erklärt, die Sie kennen müssen, um FHS-Formate in einem DRIVE-Programm einsetzen zu können. Alles, was Sie sonst noch über den FHS-Einsatz wissen müssen, erfahren Sie in den entsprechenden Handbüchern („FHS“ [20] und „IFG für FHS“ [19]).

FHS-Bildschirmformate zur Ausgabe und Eingabe von Daten werden außerhalb von DRIVE/WINDOWS erstellt. Die Formate werden mit dem Interaktiven Formatgenerator aufgebaut. Das Format-Handling-System unterstützt den Austausch von formatierten Nachrichten zwischen der Datenstation und dem DRIVE-Anwendungsprogramm. Der Datenaustausch zwischen Anwendungsprogramm und FHS findet im Datenübergabebereich statt. Dieser Bereich enthält die dem Programm zugänglichen Felder und Attribute. Die Datenstruktur im Datenübergabebereich wird durch die Adressierungshilfe mit DRIVE-Sprachmitteln beschrieben. DRIVE/WINDOWS kann über die Adressierungshilfe auf die Felder des FHS-Formats zugreifen. Dies geschieht mit Hilfe von symbolischen Feldnamen. Im folgenden ist der logische Aufbau einer Adressierungshilfe abgebildet.



DRIVE/WINDOWS unterstützt #Formate von FHS. #Formate sind Formate, bei denen Sie im Programm sowohl die Eigenschaften der Formatfelder (Feldattribute) als auch die Eigenschaften des Formats (Globalattribute) ändern können. #Formate besitzen einen erweiterten Datenübergabebereich (Extended User Area, EUA) mit getrennten Attributblökken und Feldinhalten. Mit diesem Datenübergabebereich können alle Funktionen von FHS genutzt werden. Der Datenübergabebereich besteht aus:

– Globalattributen
– Feldattributen

} Attributblöcke

– Feldinhalten

„Attribut“ bedeutet im Zusammenhang mit Formaterstellung die Eigenschaft eines Formats oder Feldes zur Darstellung, Aufbereitung oder Prüfung. Ein Attribut wird entweder bei der Formaterstellung mit dem IFG festgelegt (statisches Attribut) oder im Anwendungsprogramm über die Globalattribute und Feldattribute (dynamische Attribute).

Globalattribute bestimmen Eigenschaften für den kompletten Bildschirm. Über sie kann die Ausgabeformatierung gesteuert werden. Sie liefern FHS-Returncodes und enthalten Anzeigen, ob Felder modifiziert wurden. Damit kann ein Ausgabezyklus gesteuert sowie Darstellungseigenschaften festgelegt werden. Die Globalattribute können innerhalb eines DRIVE-Programms modifiziert bzw. abgefragt werden.

Feldattribute bestimmen die Eigenschaften eines jeden Bildschirmfeldes, die beim Definieren des Formats mit IFG festgelegt werden. Die Feldattribute können innerhalb eines DRIVE-Programms vor jeder Ausgabe modifiziert werden.

Vorteile des FHS-Formats:

– Die FHS-Formate können auch in Programmen anderer Programmiersprachen verwendet werden.

– FHS-Formate sind performanter, da sie bereits durch das Programm IFG vorübersetzt sind.

– Die Attribute können modifiziert werden.

Sie können die Adressierungshilfen mit CHECK- oder INIT-Klauseln ergänzen. Die Source der Adressierungshilfe steht in der IFG-Formatbibliothek. Nachträgliche Ergänzungen der Adressierungshilfe mit der MASK-Klausel sind nicht erlaubt. Die Anzahl der Feldattribute und Feldinhalte darf nicht verändert werden.

Eine reibungslose Zusammenarbeit zwischen FHS und DRIVE/WINDOWS ist nur dann gewährleistet, wenn die Source der Adressierungshilfe nicht manipuliert ist.

5.2.2 IFG starten

Der IFG benötigt zum Ablauf die Formatsteuerung FHS, das FHS-Formatierungsprogramm MFHSROUT und die IFG-Formatbibliothek SYSFHS.IFG.*xxx*.D (*xxx* steht für die Versionsbezeichnung).

Befindet sich die FHS-Komponente MFHSROUT nicht in der Modulbibliothek $TSOS.TASKLIB, müssen Sie die entsprechende Modulbibliothek als Bindemodul-Bibliothek zuweisen. Verwenden Sie dazu das Kommando

```
/SET-TASKLIB LIBRARY=bibliotheksname
```

oder das Kommando

```
/SET-FILE-LINK LINK-NAME=MROUTLIB,FILE-NAME=bibliotheksname
```

Die IFG-Formatbibliothek hat standardmäßig den Namen SYSFHS.IFG.080.D, kann aber beliebig benannt und unter einer beliebigen Benutzerkennung installiert werden. Weisen Sie mit dem folgenden Kommando die IFG-Formatbibliothek zu:

```
/SET-FILE-LINK LINK-NAME=IFGMAPS,FILE-NAME=bibliotheksname
```

Formate, die mit IFG ≤ V5.0 erstellt wurden, werden in Formatdateien (ISAM-Dateien) gespeichert. Der Name der IFG-Formatdatei lautet standardmäßig IFG.MAPS. Er kann wie die IFG-Formatbibliothek umbenannt werden.

Starten Sie nun das Programm IFG mit dem Kommando

```
/START-PROGRAM FROM-FILE $IFG
```

5.2.3 FHS-Format für den DRIVE-Einsatz vorbereiten

IFG-Formate müssen Sie mit der IFG-Funktion „Einsatzvorbereitung“ für den Einsatz in DRIVE/WINDOWS vorbereiten.

Einsatzvorbereitung bedeutet, daß die von Ihnen erstellten Formate von IFG vorübersetzt und als Elemente des Typs R (Objektmodul) in einer festgelegten DRIVE-Formatbibliothek gespeichert werden.
Auf diese DRIVE-Formatbibliothek, in der auch die Adressierungshilfen als Elemente des Typs S vorliegen, greift die Formatsteuerung FHS zu, mit der DRIVE/WINDOWS arbeitet.

Die in der DRIVE-Formatbibliothek gespeicherten, für den Einsatz in DRIVE/WINDOWS vorbereiteten Formate werden in den DRIVE-Handbüchern als FHS-Formate bezeichnet.

Einsatzvorbereitung im Programm IFG

IFG-Formate können erst dann in DRIVE/WINDOWS eingesetzt werden, wenn diese Formate und die zugehörigen Adressierungshilfen in einer DRIVE-Formatbibliothek (PLAM-Bibliothek) vorliegen.

Den Namen der DRIVE-Formatbibliothek legen Sie in der IFG-Funktion „Benutzerprofilverwaltung“ im Bildschirm „Angaben zu Einsatzbibliotheken“ fest.
Die für den Einsatz in DRIVE/WINDOWS vorzubereitenden Formate und Adressierungshilfen müssen in derselben Bibliothek gespeichert werden. In dieser Bibliothek können auch die IFG-Formate abgelegt werden.
Der im Benutzerprofil festgelegte Bibliotheksname wird in den Bildschirm für die IFG-Funktion „Einsatzvorbereitung“ übernommen. Dort können Sie den Bibliotheksnamen gegebenenfalls ändern.

Sie müssen noch Angaben zu Adressierungshilfen machen. Dazu geben Sie bei der IFG-Funktion „Benutzerprofilverwaltung/Angaben zu Adressierungshilfen“ ein „JA“ bei der „Programmiersprache DRIVE/WINDOWS“ an. Ein weiteres „JA“ ist bei der „Struktur des Datenuebergabebereiches“ für „getrennte Attributblöcke und Feldinhalte“ einzutragen.

Für den Einsatz in DRIVE/WINDOWS dürfen nur Teilformate verwendet werden. Um im IFG ein Teilformat zu erzeugen, müssen Sie die IFG-Funktion „Anzeigeeigenschaften des Formats ändern“ aufrufen. Die Zeile mit der Abfrage „Anfangszeile beim Einsatz als Teilformat: 00“ überschreiben Sie mit einer Ziffer > 0.

Nach diesen Vorbereitungen führen Sie die IFG-Funktion „Einsatzvorbereitung“ durch.

Zuweisen der Formatbibliothek

Standardmäßig erwartet DRIVE/WINDOWS die vorübersetzten FHS-Formate (Typ R) und die Adressierungshilfen (Typ S) in der DRIVE-Formatbibliothek mit dem Dateikettungsnamen FORMOML. Weisen Sie daher der DRIVE-Formatbibliothek, die die einzusetzenden FHS-Formate enthält, mit dem BS2000-Kommando /SET-FILE-LINK den Dateikettungsnamen FORMOML zu:

```
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=formatbibliothek
```

Ist im TIAM-Betrieb der Dateikettungsname FORMOML nicht bekannt, sucht DRIVE/WINDOWS die ausgewählten Formate in der Standard-Formatbibliothek DRI.LIB. Wollen Sie eine andere Formatbibliothek zuweisen, müssen Sie diese DRIVE/WINDOWS vor der ersten Verarbeitungsanweisung mitteilen. Geben Sie den Namen dieser Bibliothek in der PARAMETER STATIC-Anweisung mit dem Operanden FORMLIB=*formatbibliothek* an.

```
PAR STATIC FORMLIB=formatbibliothek
```

Nach der Vorbereitung eines FHS-Formats für den Einsatz in DRIVE/WINDOWS, lassen sich folgende Benennungen unterscheiden:

Name der IFG-Formatbibliothek	*formatbibliothek*
Name des IFG-Formats:	*format*
Name der DRIVE-Formatbibliothek:	*formatbibliothek* (Dateikettungsname = FORMOML)
Name der Adressierungshilfen:	*format* oder anderer Name, wenn vom Formatnamen verschieden

Der Name der IFG-Formatbibliothek muß nicht mit dem Namen der DRIVE-Formatbibliothek übereinstimmen.

Besonderheiten beim UTM-Betrieb

Wollen Sie im UTM-Betrieb FHS-Formate einsetzen, ist in der Startprozedur zwingend erforderlich:

– FILE-Kommando mit LINK=FORMOML oder

– DRIVE-Anweisung PARAMETER STATIC FORMLIB=...

Außerdem müssen Sie die FHS-Startparamter MAPLIB und DE angeben. Die Startparameter haben den Präfix „.FHS“ (siehe „FHS“ [29]):

– .FHS MAPLIB=formatbibliothek

– .FHS DE=NO

5.2.4 Behandlung bereits vorhandener Formate

Formate, die mit IFG ≤ V5.0 erstellt wurden, wurden in Formatdateien (ISAM-Dateien) gespeichert. Da ab IFG V.6.0 ausschließlich PLAM-Bibliotheken verwendet werden, müssen Formate, die mit IFG ≤ V5.0 erstellt wurden und mit IFG verändert werden sollen, umgesetzt werden. Dafür steht im IFG die Funktion „Formatdatei früherer Versionen umwandeln“ zur Verfügung (siehe „IFG“ [28]). Formate, die mit neueren IFG-Versionen erstellt wurden, müssen nicht umgewandelt werden.

Bereits vorhandene Formate, die mit der IFG Version 5.0 (und folgenden Versionen) erstellt wurden und die Sie unverändert in DRIVE/WINDOWS einsetzen wollen, müssen nicht umgesetzt werden.

5.2.5 FHS-Format in einem DRIVE-Programm einsetzen

Sie müssen folgende Anweisungen anwenden, wenn Sie ein FHS-Format in einem DRIVE-Programm einsetzen wollen:

DECLARE SCREEN screenformat

Diese Anweisung erzeugt eine Variable, in die die Daten des FHS-Formats (Attributwerte und Feldinhalte) übernommen werden. Diese Variable wird in den DRIVE/WINDOWS-Handbüchern auch SCREEN-Variable bezeichnet. screenformat ist der Name des FHS-Formats. Die Anweisung muß im Deklarationsteil des Programms stehen.

SET oder INTO-Klausel

Füllen der SCREEN-Variablen mit Inhalt und Beeinflussen der Attribute (≅ Eigenschaften) der Bildschirmfelder. Die Anweisung muß im Ablaufteil des Programms stehen.

DISPLAY screenformat

Diese Anweisung führt eine FHS-Formataufbereitung aus und gibt das Format auf Bildschirm aus. Konkret geschieht folgendes: Die SCREEN-Variable wird samt Inhalt an FHS übergeben und am Bildschirm ausgegeben. Die Anweisung muß im Ablaufteil des Programms stehen.

5.2.5.1 FHS-Format definieren

DECLARE SCREEN (Syntax und Regeln siehe „DRIVE-Lexikon“ [3]) erzeugt eine Variable, in die die Adressierungshilfe des FHS-Formats kopiert wird. Dieses Format muß bereits über IFG/FHS zur Verfügung stehen (siehe Abschnitt „FHS-Format für den DRIVE-Einsatz vorbereiten“ auf Seite 146).

In DRIVE-Programmen muß für jedes verwendete FHS-Format die Struktur des Datenübergabebereichs zur Verfügung gestellt werden: Die Attributblöcke (≅ Global- und Feldattribute) und die Feldinhalte. DECLARE SCREEN greift auf die DRIVE-Formatbibliothek zu und kopiert die Adressierungshilfe als strukturierte Variable in das Programm. Über diese SCREEN-Variable können Sie die Bildschirmfelder und die Attribute (≅ Eigenschaften) der Bildschirmfelder selbst versorgen, falls Sie die durch IFG festgelegte Standardbelegung nicht wünschen.

Wenn Sie Ihre Programme mit der Übersetzungsoption OPTION SCREEN-CHECK=OFF übersetzen lassen, wertet DRIVE/WINDOWS die von IFG erzeugten CHECK-KLauseln nicht aus. Der erzeugte Programmcode ist performanter und benötigt weniger Speicherplatz.

Beispiel

```
DECLARE SCREEN screenformat [variable];
```

Angabe eines Namens für die SCREEN-Variable

Sie können selbst einen Namen für die SCREEN-Variable *variable* angeben. Falls Sie einzelne Bildschirmfelder ansprechen wollen, also Feldattribute und Feldinhalte, so müssen Sie diesen Namen verwenden.
Falls Sie das komplette Bildschirmformat, also Globalattribute, ansprechen wollen, so müssen Sie den FHS-Formatnamen verwenden.

Keine Angabe eines Namens für die SCREEN-Variable

Falls Sie keinen Namen für die SCREEN-Variable *variable* angeben, legt DRIVE/WINDOWS automatisch eine SCREEN-Variable mit dem Namen "&screenformat" an.
Falls Sie einzelne Bildschirmfelder ansprechen wollen, also Feldattribute und Feldinhalte, so müssen Sie diesen Namen verwenden.
Falls Sie das komplette Bildschirmformat, also Globalattribute, ansprechen wollen, so müssen Sie den FHS-Formatnamen verwenden.

Die SCREEN-Variable enthält die komplette Adressierungshilfe des FHS-Formats, d.h. Felder und Attribute (≅ Eigenschaften), sowie die Speicherbereiche für Datenwerte. Sie wird in die Übersetzungsliste aufgenommen (siehe „DRIVE-Programmiersystem“ [1], Kapitel

DRIVE-Programme entwickeln, Abschnitt DRIVE-Programme analysieren und korrigieren): Unmittelbar nach der DECLARE SCREEN-Anweisung wird sie in die Programmsource kopiert (siehe folgendes Beispiel).

Im folgenden ist eine SCREEN-Variable auszugsweise abgebildet. Die drei Bereiche mit der Stufennummer 19 enthalten die Globalattribute, die Feldattribute und die Felder für die Datenwerte. Die letzte Angabe der in der SCREEN-Variablen &VARMASK abgebildeten Adressierungshilfe bezieht sich auf deren physikalische Länge: hier 128 Byte.

Beispiel für eine SCREEN-Variable:

```
...
/* Programmsource */
...
DCL SCREEN MASKE &VARMASK;

DCL VAR 1 "&VARMASK", COPY ":S:$DRIVE.SESAM.LIB" ("MASKE");
                                    /* FORMAT NAME: MASKE  */

/*************************************************************/
/*      GLOBAL ATTRIBUTE BLOCK ( @ Globalattribute)          */
/*************************************************************/
   19 MASKE_GLOBALS,
      /* FORM_RETURNCODE */
         20 RC_MAIN               INTEGER,
         ...
      /* FORM_INDICATORS */
         20 FIELDS_MOD            CHAR (1),
         20 FIELDS_VALID          CHAR (1),
         ...

/*************************************************************/
/*      FIELD ATTRIBUTE BLOCKS (≅ Feldattribute)             */
/*************************************************************/
   19 MASKE_ATTR,
      20 ARTIKEL_NR_FAB,
         21 FIELD_LEN             SMALLINT,
         21 BASIC_ATTR,
            22 INPUT_STATE        CHAR (1),
            22 INPUT_STATE_ACT    CHAR (1),
            22 EDIT_STATE         CHAR (1),
            22 OUTPUT_CTL         CHAR (1),
         21 FLD_INPUT,
            22 INPUT_CTL          CHAR (1),
            22 PROTECTION         CHAR (1),
```

```
21 DISPLAY_CTL,
           22 INTENSITY          CHAR (1),
           22 VISIBILITY         CHAR (1),
                          ...
        21 COLOR                 CHAR (1),
        ...
     20 ABEZEICHN_FAB
        LIKE &MASKE_ATTR.ARTIKEL_NR_FAB,
        ...
/*********************************************************/
/*      FIELD DATA PART (≅ Feldinhalte)                  */
/*********************************************************/
   19 MASKE_DATA,
      20 ARTIKEL_NR              CHAR (006),
      20 ABEZEICHN               CHAR (015),
      20 APREIS                  NUM  (08,00),
         CHECK &APREIS ≥0
      20 EINGABE3                 NUM  (08,00),
      ...
 &AREA_LENGTH_MASKE PERMANENT SMALLINT INIT 00128;
```

5.2.5.2 FHS-Format mit Inhalt füllen

Sie wollen ein FHS-Format mit Inhalt füllen. Dazu müssen Sie mit der Anweisung SET (Syntax und Regeln siehe „DRIVE-Lexikon“ [3]) oder mit der INTO-Klausel der SCREEN-Variablen Werte zuweisen.

Grundlage für die unten angegebenen Beispiele ist die Adressierungshilfe, die in der Variablen &VARMASK abgebildet ist (siehe Abschnitt „FHS-Format definieren“ auf Seite 149). Der FHS-Formatname ist MASKE; ein weiteres FHS-Format heißt MASKE2.

i Im UTM- und im TIAM-Betrieb ist die Ausrichtung der Datenwerte in Feldern mit dem Datentyp CHARACTER unterschiedlich. Arbeiten Sie mit der MASK-KLausel, um die unterschiedliche Ausrichtung zu vermeiden.

Wertzuweisung mit SET

Einem Feld (*variable*) des FHS-Formats (genauer: einem Feld der SCREEN-Variablen) können zugewiesen werden:
ein Wert (*ausdruck*), der NULL-Wert und/oder Feldattribute (*attribute*) (siehe Zuweisen von Feldattributen im Abschnitt „Attribute in FHS-Formaten modifizieren“ auf Seite 152).

Die zugewiesenen Werte können z.B. sein: Variablen, Literale, arithmetische Ausdrücke, Stringfunktionen usw. (siehe „DRIVE-Lexikon“ [3], Metavariable *ausdruck*).

Beispiel

```
SET &VARMASK.ARTIKEL_NR = &A,
    &VARMASK.ABEZEICHN  = 'HAMMER',
    &VARMASK.APREIS     = (&VAR1 + &VAR2) / 2,
DISPLAY MASKE;
```

Füllen eines FHS-Formats mit Datenbank-Inhalten (INTO-Klausel)

Über SELECT bzw. FETCH können Sie auf Satzelemente aus Datenbanken zugreifen, wobei die INTO-Klausel eine SCREEN-Variable als Argument angibt. Die gewünschten Satzelemente werden dabei in die SCREEN-Variable gelesen. Damit steht die SCREEN-Variable für eine FHS-Formatausgabe (DISPLAY screenformat) zur Verfügung: SELECT... INTO... bzw. FETCH... INTO...

Beispiel

```
SELECT ARTNAME, ARTPREIS FROM DBARTIKEL INTO
       &VARMASK.ABEZEICHN, &VARMASK.APREIS WHERE ARTNR > &ARTIKEL_NR;
```

Die Werte der Satzelemente ARTNAME und ARTPREIS der Datenbanksätze, deren Artikelnummer größer ist als der Wert der Variablen &ARTIKEL_NR, werden für eine FHS-Bildschirmausgabe in der Variablen &VARMASK bereitgestellt.

5.2.5.3 Attribute in FHS-Formaten modifizieren

Mit der Anweisung SET (siehe DRIVE-Lexikon [3], Anweisung SET) können Sie Global- und Feldattribute von FHS-Formaten modifizieren.

Zuweisen von Feldattributen

Sie weisen dem entsprechenden Feld mit Hilfe von SET *variable* WITH ATTRIBUTE die gewünschten Attribute (≅ Eigenschaften) zu. Es dürfen mehrere Feldattribute einem Feld zugewiesen werden.

Feldattribute bestimmen die Eigenschaft des jeweiligen Feldes, z.B. kursiv und rot oder dunkel gesteuertes oder Pflichtfeld.

Die Feldattribute, die Sie verwenden können, sind im DRIVE-Lexikon [3] unter der Metavariablen *attribute* beschrieben.

Es können nur die Feldattribute modifiziert werden, deren entsprechende Feldattributgruppe in der SCREEN-Variablen (≅ Adressierungshilfe) vorhanden ist. Welche Attributgruppen in einem Feldattributblock vorhanden sind, wird über die Benutzerprofilverwaltung im IFG festgelegt.

Beispiel 1

Einem FHS-Feld werden über die Variable &ARTIKEL_NR folgende Attribute zugewiesen: Unterstreichung, Pflichtfeld und Positionierung des Cursors (Schreibmarke) auf das Feld. FHS wertet das Feldattribut CURSOR nur aus, wenn gleichzeitig das Globalattribut CURSOR gesetzt ist.

global: `SET MASKE ATTR (CURSOR);`

feldspezifisch: `SET &ARTIKEL_NR ATTR (UNDERLINE,MUST,CURSOR);`

Beispiel 2

Rücksetzen des Attributes CURSOR.

global: `SET MASKE WITH ATTR (NOCURSOR);`

feldspezifisch: `SET &ARTIKEL_NR WITH ATTR (NOCURSOR);`

Zuweisen von Globalattributen

Sie weisen dem kompletten FHS-Format ein oder mehrere Globalattribute zu mit SET screenformat WITH ATTRIBUTE ...

Die Globalattribute, die Sie verwenden können, sind im DRIVE-Lexikon [3] unter der Metavariablen *attribute* beschrieben.

Beispiel

```
SET MASKE WITH ATTR (HARDCOPY);
```

Der gesamte Bildschirminhalt wird auf Hardcopy ausgegeben.

Zuweisen von Werten und Attributen

Sie können einem FHS-Bildschirmfeld zugleich Werte und Attribute (≅ Eigenschaften) zuweisen.

Beispiel

```
SET &VARMASK.EINGABE3 = &A WITH ATTR (INVERSE,UL);
```

Dem Bildschirmfeld wird der Wert der Variablen &A und die Eigenschaften inverse Darstellung und Unterstrich (UNDERLINE) zugewiesen.

3GL-Methode

Eine andere Möglichkeit, Attribute zu modifizieren, ist das Zuweisen des Attributwertes auf das Feld, das durch die strukturierte Variable direkt angesprochen werden kann. Dies entspräche einer 3GL-Methode, z.B.

```
SET &VARMASK.MASKE_ATTR.APREIS_FAB.FLD_INPUT.PROTECTION = 'P'; (Feldattribut)
SET &VARMASK.MASKE_DATA.APREIS = 23; (Feldinhalt)
```

Einfacher geht es mit 4GL (DRIVE/WINDOWS):

```
SET &APREIS = 23 ATTR (PROT);
```

Das Modifizieren eines Globalattributes geschieht analog, z.B.

```
SET &MASKE.MASKE_GLOBALS.COPY_CTL = 'H'; (3GL)
SET MASKE ATTR (HARDCOPY); (4GL)
```

Welche Werte welchen Feldern zugewiesen werden dürfen und was diese Werte bewirken, entnehmen Sie der FHS-Beschreibung (siehe FHS [29]).

Bei der ersten Art der Attributmodifikation (SET ... WITH ATTR) verhindert DRIVE/WINDOWS eventuelle Fehler, nicht jedoch bei der zweiten, der 3GL-Methode.

Die folgende Tabelle zeigt eine Gegenüberstellung der DRIVE-Syntax zu FHS-Feldnamen und -Inhalt.

DRIVE-Syntax	**format_name_GLOBALS.**	**CHAR**
ALARM	ALARM_CTL	A
CURSOR	CURSOR_CTL	F
NOCURSOR	CURSOR_CTL	␣
HARDCOPY	COPY_CTL	H
INIT	INIT_CTL	F
NOINIT	INIT_CTL	N
DEFAULT	<alle>	␣
DRIVE-Syntax	&var_name_FAB.	CHAR
MUST	FLD_INPUT. INPUT_CTL	M
POTMUST		P
NORMALINPUT		N
UNPROTECTED	FLD_INPUT. PROTECTION	U
PROTECTED		P

DRIVE-Syntax	format_name_GLOBALS.	CHAR
HIGHINTENSITY NORMALINTENSITY	DISPLAY_CTL. INTENSITY	H N
VISIBLE SIGN INVISIBLE	DISPLAY_CTL. VISIBILITY	V S I
UNDERLINE NOUNDERLINE	DISPLAY_CTL. UNDERLINE	Y N
INVERSE NOINVERSE	DISPLAY_CTL. INVERSE	Y N
RED GREEN YELLOW BLUE MAGNETA CYAN WHITE NOCOLOR	COLOR	1 2 3 4 5 6 7 N
CURSOR NOCURSOR	INIT_CURSOR	H N
VALID INVALID	BASIC_ATTR. EDIT_STATE	V I

5.2.5.4 Cursorpositionen programmieren

Das Globalattribut CURSOR bzw. NOCURSOR legt fest, ob das Feldattribut CURSOR ausgewertet wird (Wert F) oder nicht (Wert ␣).

Das Feldattribut CURSOR bzw. NOCURSOR legt fest, bei welchem Bildschirmfeld der Cursor positioniert wird.

FHS wertet das Feldattribut CURSOR nur aus, wenn gleichzeitig das Globalattribut CURSOR gesetzt ist. Außerdem muß in der Adressierungshilfe die Feldattributgruppe CURSOR vorhanden sein.

Beispiel

In einem Anwendungs-Programm mit Eingabemöglichkeiten soll im Falle einer fehlerhaften Bildschirm-Eingabe der Cursor auf das erste, nicht richtig versorgte Feld positioniert werden. Alle nicht richtig versorgten Felder sollen zusätzlich durch Unterstrich gekennzeichnet werden. Das FHS-Format MASKE hat u.a. die Eingabefelder &ARTIKEL_NR, &ABEZEICHN und &APREIS.

```
PROC INSFORM;
...
DCL SCREEN MASKE ERRORATTR (CURSOR, UL); ------------------------------------------------   (1)
...
SET MASKE ATTR (CURSOR); ----------------------------------------------------------------   (2)
...
DISPLAY MASKE SCREENERROR REPEAT;
...
```

(1) Einstellen aller fehlerhaften Felder mit den Attributen CURSOR und UNDERLINE

(2) Einstellen des FHS-Formats mit dem Globalattribut CURSOR

Die folgende Tabelle gibt einen Überblick, an welcher Position, d.h. bei welchem Feld der Cursor positioniert ist, abhängig von den jeweiligen Wertzuweisungen auf Global- und Feldattribute. In dieser Tabelle werden die drei Bildschirmfelder &X, &Y und &Z verwendet. Die Tabelle zeigt die wichtigsten Kombinationsmöglichkeiten.

Globalzuweisung	**SET maske**	**ATTR(CURSOR)**			**ATTR (NOCURSOR)**
Feldzuweisungen	SET &X WITH	N	N	N	*
	SET &Y WITH	C	N	N	*
	SET &Z WITH	C	C	N	*
	Position des Cursors auf	&Y	&Z	&X	&X

Legende:
Die Zeichen geben an, wie die Feldattribute gesetzt sind:
* ≅ beliebig (CURSOR) oder (NOCURSOR) oder Default
C ≅ ATTR (CURSOR)
N ≅ (NOCURSOR)

5.2.5.5 FHS-Format am Bildschirm ausgeben

DISPLAY screenformat führt eine FHS-Formataufbereitung aus und gibt das Format auf dem Bildschirm aus, z.B.

```
DISPLAY MASKE;
```

Konkret geschieht folgendes: Die SCREEN-Variable wird an FHS zur Bildschirmausgabe übergeben.

Mit Hilfe des SCREENERROR-Operanden können Sie das Verhalten von DRIVE/WINDOWS bestimmen, wenn fehlerhafte Feldeingaben vorliegen (siehe Abschnitt „Verhalten von DRIVE/WINDOWS bei fehlerhaften Feldeingaben“ auf Seite 158).

Sie können mehrere Teilformate zu einem Format zusammensetzen, indem Sie die FHS-Formatnamen in der DISPLAY-Anweisung angeben, z.B.:

```
SET &VARMASK.ARTIKEL_NR = &A,
   &VARMASK2.ARTIKEL_NR = &B,
   &VARMASK.EINGABE3 = &C;
DISPLAY MASKE, MASKE2;
```

Beispiel: Programm INSERTFHS zur Neuaufnahme von Datensätzen

```
PROC INSERTFHS;
DECLARE VARIABLE ...
...
/* FHS-Adressierungshilfe in eine SCREEN-Variable kopieren              */
DECLARE SCREEN INSMASK &INSMASK;
...
DISPLAY INSMASK;
/* Das Programm stellt nun das FHS-Bildschirmformat INSMASK zur         */
/* Verfügung. Der Anwender kann einen neuen Artikelsatz eingeben.       */
```


```
/* Weiter im Programm: Aufnahme des Artikelsatzes in die Datenbank      */
...
INSERT INTO DBART VALUES (&INSMASK_DATA.*);
...
END PROC;
```

Programmier-Hinweise:

– Erfolgt nach der Ausgabe eines FHS-Formats ein SEND MESSAGE ... WAIT, so muß der Anwender die entsprechende Meldung in der Meldungszeile durch eine beliebige Eingabe (z.B. DUE) quittieren. Mit dem Quittieren der Meldung erfolgt jedoch **nicht** die Verarbeitung der Daten, die vor dem Quittieren der Meldung in das FHS-Format eingegeben wurden. Die Verarbeitung der Daten ist erst möglich, wenn das Format nochmals mit DISPLAY ausgegeben wird.

  Erfolgt dagegen nach der Ausgabe eines FHS-Formats ein SEND MESSAGE ... WITHOUT WAIT, so bedeutet dies eine freilaufende Nachricht in der Meldungszeile. Die freilaufende Nachricht muß nicht quittiert werden. Das FHS-Format bleibt in jedem Fall am Bildschirm stehen.

– Eine Eingabeverarbeitung findet immer nur für die jeweils aktuell (≅ im aktuellen DISPLAY) ausgegebenen FHS-Formate statt.

– FHS-Formate, für die keine Daten eingegeben werden, werden im TIAM-Betrieb nicht verarbeitet. Das bedeutet, daß kein "MUST ERROR" erkannt wird. Im UTM-Betrieb werden alle Formate bearbeitet.

– UTM beendet sich nach FHS-Fehlern mit PEND ER.

5.2.6 Verhalten von DRIVE/WINDOWS bei fehlerhaften Feldeingaben

Mit Hilfe der SCREENERROR-Klausel bei der Anweisung DISPLAY können Sie das Verhalten des DRIVE-Programms bestimmen, wenn fehlerhafte Eingaben in FHS-Felder gemacht wurden (siehe „DRIVE-Lexikon“ [3], Anweisung DISPLAY screenformat). Sie können auswählen zwischen einem automatischen Fehlerdialog, den DRIVE/WINDOWS ausführt, und einem benutzergesteuerten Fehlerdialog.

Automatischer Fehlerdialog

Mit der Angabe SCREENERROR REPEAT führt DRIVE/WINDOWS einen automatischen Fehlerdialog durch, wenn fehlerhafte Feldeingaben in FHS-Formaten vorliegen.

```
DISPLAY screenformat SCREENERROR REPEAT
```

Fehlerhafte Eingaben können entstehen durch

– Verletzung von CHECK-Klauseln (siehe DRIVE-Lexikon [3], Anweisung DECLARE VARIABLE)

– Mißachtung der Eingabepflicht von Bildschirmfeldern (MUST-Eigenschaften)

– Formatierungsfehler, die von FHS erkannt werden

Bei fehlerhaft geführter Eingabe in ein FHS-Feld wiederholt DRIVE/WINDOWS solange die Bildschirmausgabe, bis die Eingabe fehlerfrei ist; der Inhalt der korrekten Felder bleibt dabei unverändert. Erst wenn in allen Feldern des FHS-Formats eine korrekte Eingabe vorgenommen wurde, werden die Daten in die SCREEN-Variable übertragen und die Steuerung an das DRIVE-Programm zurückgegeben.

Die Systemvariable &ERROR ist beim automatischen Fehlerdialog immer auf 'OK' gesetzt.

Zur Kennzeichnung der Felder, die fehlerhaft eingegeben wurden, gibt es drei Möglichkeiten:

– Standardwert (keine Angabe durch den DRIVE-Anwender)

  Fehlerhaft eingegebene FHS-Felder sind mit den Attributen HIGHINTENSITY und UNDERLINE belegt.

– PARAMETER ERRORATTRIBUTE

  Die Attribute, die bei der PARAMETER-Anweisung angegeben sind, gelten global bis zum Ende der DRIVE-Sitzung.

– ERRORATTRIBUTE-Operand bei DECLARE SCREEN

  Die Attribute, die bei der DECLARE SCREEN-Anweisung mit ERRORATTRIBUTE angegeben sind, gelten nur für dieses FHS-Format und haben immer nur für das aktuelle Programm Gültigkeit. Diese Angaben haben Vorrang vor denjenigen aus der PARAMETER-Anweisung.

Die Attribute für die Darstellung fehlerhafter FHS-Formate sind beschrieben im „DRIVE-Lexikon“ [3], Metavariable *attribute*.

Die letzte Zeile (≅ Meldungszeile) darf nicht benützt werden. Sie wird für DRIVE-Fehlermeldungen benötigt.

Beispiel

Bei einer bestimmten FHS-Bildschirmausgabe soll im Falle fehlerhafter Feldeingaben der Bildschirm solange wieder ausgegeben werden, bis alle Felder korrekt versorgt sind. Die fehlerhaften Felder sollen blinkend und mit Unterstrich dargestellt sein.

```
DECLARE SCREEN MASKE &VARMASK WITH ERRORATTRIBUTE (SIGN, UNDERLINE);
...
DISPLAY MASKE SCREENERROR REPEAT;
```

Benutzergesteuerter Fehlerdialog

Die Angabe SCREENERROR CONTINUE unterbindet den automatischen Fehlerdialog und überläßt dem DRIVE-Programmierer die Verantwortung für die weitere Verarbeitung. Mit anderen Worten: Sie haben die Möglichkeit, mit DRIVE-Mitteln einen eigenen Fehlerdialog zu programmieren.

```
DISPLAY screenformat SCREENERROR CONTINUE
```

Die letzte Zeile (≅ Meldungszeile) darf mitbenutzt werden, um anwendereigene Meldungen auszugeben. Erstellen Sie dazu ein eigenes Meldungsformat für die letzte Zeile mittels eines FHS-Teilformates. Verwenden Sie dafür **nicht** die SEND MESSAGE-Anweisung, da in diesem Fall die Formate aus dem vorherigen DISPLAY *screenformat* stehenbleiben, aber nicht bearbeitet werden.

Eine sinnvolle Vorgehensweise, den Fehlerdialog zu programmieren, besteht darin:

1. Prüfen, ob ein fehlerhaftes FHS-Format vorliegt,
2. Prüfen des fehlerhaften Formats auf fehlerhafte Eingaben
3. Kennzeichnen der fehlerhaften Felder mit SET ERRORATTRIBUTE

Prüfen, ob ein fehlerhaftes FHS-Format vorliegt

Ist ein Feld fehlerhaft oder ein Feld mit Eingabepflicht nicht versorgt (Globalattribut FIELDS_VALID = " ␣ "), so wird das Programm abgebrochen. Die Systemvariable &ERROR wird belegt mit dem Wert 'FORMAT ERROR'; die Systemvariable &FORMAT_NAME mit dem Namen des ersten fehlerhaften Teilformats und die Systemvariable &VAR_NAME mit dem Namen des ersten fehlerhaften Eingabefelds (siehe Abschnitt „Fehler abfangen, Endekriterien“ auf Seite 88).

Bei Abbruch durch eine fehlerhafte K-/F-Taste wird ebenfalls &ERROR mit 'FORMAT ERROR' belegt. &FORMAT_NAME enthält den String "% K/F-ERROR"; &VAR_NAME den Initialisierungswert.

Prüfen des FHS-Formats auf fehlerhafte Eingaben

FHS liefert an der Schnittstelle zu DRIVE/WINDOWS (≅ SCREEN-Variable) einen globalen sowie einen feldspezifischen Fehler-Returncode. Mit dem Feldattribut EDIT_STATE können Sie den Inhalt des FHS-Formats, d.h. Werte und Feldeigenschaften, überprüfen. Sie haben damit die Möglichkeit, in Ihrem Programm einen Fehlerdialog zu programmieren.

FHS setzt auf Grund von IFG/FHS-Regeln das Feldattribut gegebenenfalls auf INVALID bzw. MUST_ERROR. Zusätzlich setzt DRIVE/WINDOWS für alle Felder, deren Eingabe eine eventuell vorliegende CHECK-Bedingung verletzt, EDIT_STATE auf INVALID.

Weitere Prüfungen können vorgenommen werden, nachdem das DRIVE-Programm wieder die Kontrolle hat (nach DISPLAY).

Folgende Attribute sollten zur Programmierung des Fehlerdialogs geprüft werden (siehe „DRIVE-Lexikon“ [3], Metavariable *attribut*, „FHS“ [20]). Hier müssen die Feldinhalte, d.h. das jeweilige Zeichen (z.B. 'V'), angegeben werden.

Globalattribute: Anzeige Editfunktion (FIELDS_VALID)

VALID (V) : Alle Felder des Formats sind von der Editroutine geprüft und fehlerfrei, und es sind alle Felder mit Eingabepflicht eingegeben.

INVALID (␣): Mindestens ein Feld des Formats ist von der Editroutine noch nicht geprüft der fehlerhaft, oder mindestens ein Feld mit Eingabepflicht ist noch nicht eingegeben.

Feldattribute: Editzustand (EDIT_STATE)

VALID (V): Feldinhalt geprüft und fehlerfrei, bzw. keine Editroutine verlangt.

INVALID (I): Feldinhalt geprüft und fehlerhaft;

MUST ERROR (M): Feld mit Eingabepflicht noch nicht eingegeben;

NOT CHECKED(␣): Feldinhalt ungeprüft

Eingabezustand (INPUT_STATE)
Mit diesem Feldattribut kann überprüft werden, ob das Feld modifiziert wurde oder nicht.

Kennzeichnen fehlerhaft eingegebener Bildschirmfelder

Allen Feldern eines FHS-Formats, deren Feldwert EDIT_STATE = 'I' oder 'M' ist (I ≅ INVALID ≅ Feld nicht fehlerfrei; M ≅ MUST_ERROR ≅ Feld mit Eingabepflicht wurde nicht versorgt), können Sie Attribute zuweisen. Verwenden Sie dazu den ERRORATTRIBUTE-Operanden der SET-Anweisung.

Die SET ... ERRORATTRIBUTE-Anweisung bezieht sich immer auf **ein** FHS-Format (genauer: auf eine SCREEN-Variable) und wird bei der nächsten Ausgabe des Formats wirksam, also beim nächsten DISPLAY.

Die Attribute, die Sie verwenden können, sind im „DRIVE-Lexikon“ [3] unter der Metavariablen *attribute* beschrieben.

Möchten Sie das Programm mit der ersten Anweisung nach dem fehlerhaften DISPLAY fortsetzen, so müssen Sie eine entsprechende WHENEVER-Anweisung formulieren (siehe folgendes Beispiel).

Beispiel

```
DCL SCREEN MASK;
WHENEVER &ERROR IN ('FORMAT ERROR') CONTINUE;
...
/* Prüfung auf fehlerhafte Eingaben */
CYCLE;
DISPLAY MASKE SCREENERROR CONTINUE;
/* vorheriges ERROR-Attribut zurücksetzen */
  SET MASKE WITH ATTR (DEFAULT)
```

```
IF &MASKE.FIELDS_VALID <>'␣'
  THEN BREAK CYCLE;
  ELSE ...               /* Eventuell weitere Prüfungen */
/* Kennzeichnen der Fehlerfelder */
  SET MASKE WITH ERRATTR (UNDERLINE);
     END IF;
END CYCLE;
...
```

5.3 Meldung am Bildschirm ausgeben

Mit der Anweisung SEND MESSAGE können Sie eine Meldung in der letzten Bildschirmzeile ausgeben. Das aktuelle Bildschirmformat bleibt erhalten. Die maximale Ausgabelänge ist 79 Zeichen (siehe „DRIVE-Lexikon“ [3], Anweisung SEND MESSAGE).

Die Meldung wird standardmäßig im Ausgabe-Modus WAIT ausgegeben. Das bedeutet, daß das Programm nach Ausgabe der Meldung erst dann fortgesetzt wird, nachdem der Programm-Anwender die Programm-Eingabe-Taste bzw. eine als Programm-Eingabe-Taste definierte K/F-Taste betätigt.

Beispiel

```
PROC A;
DECLARE VAR &M CHAR (32) INIT 'DER ARTIKELSATZ WURDE AUFGENOMMEN';
...
SEND MESSAGE &M;     /* WAIT ist standardmäßig voreingestellt */
...
END PROC;
```

Wollen Sie eine Meldung im Ausgabe-Modus WITHOUT WAIT, so geben Sie diesen Operanden mit an. In diesem Fall läuft das DRIVE-Programm nach Ausgabe der Meldung weiter. Dabei ist es möglich, daß die Meldung sehr schnell von einer nachfolgenden Meldung überschrieben wird. Die Angabe WITHOUT WAIT ist nur sinnvoll, wenn nach Ausgabe der Meldung nicht sofort wieder eine Ausgabe erfolgt.

5.4 NULL-Werte in Bildschirmformaten

5.4.1 NULL-Werte in DRIVE-Bildschirmformaten

Mit der Anweisung PARAMETER DYNAMIC NULL FORM legen Sie für ein alphanumerisches und/oder numerisches Datenfeld je ein Zeichen für die Darstellung des NULL-Werts am Bildschirm fest (siehe „DRIVE-Lexikon“ [3], Anweisung PARAMETER DYNAMIC).

Das NULL-Wertzeichen wird durch ein alphanumerisches Literal (charliteral) der Länge 1 bestimmt (siehe „DRIVE-Lexikon“ [3], Metavariable *nullwert*).

Für alphanumerische Datenfelder kann als NULL-Wertzeichen (charliteral1) jedes beliebige Zeichen verwendet werden. Für numerische Datenfelder ist als NULL-Wertzeichen (charliteral2) nur eine Ziffer oder eines der Sonderzeichen * + - , . zugelassen.

Das NULL-Wertzeichen für alphanumerische Datenfelder (charliteral1) ist auch gültig für Felder der Zeit-Datentypen. Das NULL-Wertzeichen für numerische Datenfelder (charliteral2) ist auch gültig für Felder des Datentyps INTERVAL.

Das mit PARAMETER DYNAMIC vereinbarte NULL-Wertzeichen kann für DRIVE-Bildschirmformate durch Angabe eines anderen NULL-Wertzeichens in der NULL-Klausel der DECLARE FORM-Anweisung überschrieben werden.
Ein NULL-Wertzeichen, das über DECLARE FORM NULL festgelegt wurde, gilt nur für das jeweilige mit DECLARE FORM definierte Bildschirmformat.

Wenn Sie kein NULL-Wertzeichen explizit festlegen, gilt für alphanumerische oder numerische Datenfelder das Sonderzeichen @ als Standardvorgabewert.

Ausgabe von NULL-Wertzeichen

NULL-Werte werden bei der Ausgabe durch Expandierung des NULL-Wertzeichens auf die definierte Länge des betreffenden Feldes dargestellt.

Beispiel

```
DCL VAR &nullwert CHAR(10);
PAR DYN NULL FORM CHARTYPE='#';
...
SET &nullwert=NULL;
DISPLAY FORM &nullwert;
```

Ausgabe: ##########

Eingabe von NULL-Wertzeichen

Sie können NULL-Wertzeichen in Datenfelder vom Datentyp CHARACTER, NUMERIC, DECIMAL, INTEGER, SMALLINT, DATE, TIME, TIME(3), TIMESTAMP(3) und INTERVAL eingeben.

Haben Sie kein NULL-Wertzeichen explizit vereinbart, können Sie das Standard-NULL-Wertzeichen (Sonderzeichen @) eingeben.

Bei Eingabe der entsprechend definierten NULL-Wertzeichen werden die Werte in eine "IS NULL"-Information umgesetzt. Es ist erforderlich, das komplette Eingabefeld mit dem entsprechenden NULL-Wertzeichen zu belegen.

5.4.2 NULL-Werte in FHS-Formaten

Zur Ein- und Ausgabe von NULL-Werten in FHS-Formaten stellt das Formatierungssystem IFG/FHS (ab Version 6.0) zwei Mechanismen zur Verfügung: „undefinierter Wert“ (≅ NULL-Wert) und „Ersatzzeichen“ (≅ NULL-Wertzeichen) (siehe „IFG“ [28], „FHS“ [29]). Diese Mechanismen werden von DRIVE/WINDOWS unterstützt.

Mit der Funktion „Format aendern und ergaenzen“ in IFG legen Sie Zeichen („Ersatzzeichen“) für die Darstellung des NULL-Wertes („undefinierter Wert“) in FHS-Formaten fest.

Sie können jeweils ein NULL-Wertzeichen für Felder von alphanumerischem und numerischem Datentyp und vom Datentyp DATE definieren.

Das NULL-Wertzeichen können Sie entweder als darstellbares Zeichen der Länge 1 oder als hexadezimalen Wert (insbesondere für nicht darstellbare Zeichen) angeben.

Als NULL-Wertzeichen für Felder von numerischem Datentyp ist entweder eine Ziffer von 0-9 oder eines der Sonderzeichen + - * / . , zu wählen.

Das definierte NULL-Wertzeichen können Sie in das Benutzerprofil als Standardwert übernehmen. Wollen Sie für ein FHS-Format das im Benutzerprofil enthaltene NULL-Wertzeichen nicht übernehmen, so definieren Sie ein neues formatspezifisches NULL-Wertzeichen, das nur für dieses Format gültig ist.

Wenn Sie kein NULL-Wertzeichen mit der Funktion „Format aendern und ergaenzen“ definieren, gilt das Sonderzeichen @ als Standardvorgabewert für die NULL-Wertdarstellung.Bei der Erstellung eines IFG-Formats mit IFG (Funktion „Format erstellen“) können Sie festlegen,

– ob NULL-Werte („undefinierte Werte“) von FHS verarbeitet werden sollen oder ob keine Verarbeitung von NULL-Werten erfolgen soll und

– ob ein Feld, für das die Verarbeitung von NULL-Werten zugelassen ist, vollständig oder nur die erste Position des Feldes mit dem vereinbarten NULL-Wertzeichen („Ersatzzeichen“) ausgefüllt werden muß. Soll die NULL-Werteingabe nur über ein einziges Zei-

chen erfolgen, wird in IFG/FHS-Handbüchern der Begriff „schnelle Überprüfung“ verwendet.
Wird nur die erste Position eines Feldes mit dem „Ersatzzeichen“ ausgefüllt und wurde bei der Formaterstellung die „schnelle Überprüfung“ gewählt, wird die Bearbeitung langer Felder wesentlich beschleunigt.

Ausgabe von NULL-Werten (mit Verarbeitung von „undefinierten Werten“)

NULL-Werte („undefinierten Werte“) werden bei der Ausgabe in FHS-Formaten, die mit IFG ab Version 6.0 erstellt wurden, durch Expandierung des im IFG definierten NULL-Wertzeichens („Ersatzzeichen“) auf die definierte Länge des betreffenden Feldes dargestellt.

Eingabe von NULL-Werten (mit Verarbeitung von „undefinierten Werten“)

Sie können NULL-Wertzeichen in Felder jedes Datentyps (alphanumerisch, numerisch und DATE) eingeben.

Bei Eingabe der entsprechend definierten NULL-Wertzeichen werden die Werte in eine "IS NULL"-Information umgesetzt. Je nach Festlegung bei der Erstellung eines IFG-Formats muß das Eingabefeld vollständig oder nur die erste Position des Feldes mit dem vereinbarten NULL-Wertzeichen („Ersatzzeichen“) ausgefüllt werden.

Enthält ein DRIVE-Programm zusätzlich zu FHS-Formaten, für die im IFG (ab Version 6.0) NULL-Werte definiert wurden, auch DRIVE-Bildschirmformate oder FHS-Formate, für die der NULL-Wert über die DRIVE-Anweisung PARAMETER DYNAMIC FORM NULL festgelegt wurde, müssen Sie folgendes beachten:

Haben Sie mit PARAMETER DYNAMIC FORM ein NULL-Wertzeichen festgelegt, das nicht dem Standardwert (Sonderzeichen @) entspricht und das sich von dem NULL-Wertzeichen unterscheidet, das bei der Erstellung von FHS-Formaten vereinbart wurde, gilt:
Wenn Sie die Verarbeitung von NULL-Werten („undefinierte Werte“) vereinbart haben, dann wird bei der Eingabe von NULL-Wertzeichen in ein FHS-Format (ab V6.0) nur das Zeichen („Ersatzzeichen“) als NULL-Wertzeichen verarbeitet, das bei der Erstellung des FHS-Formats festgelegt wurde.
Geben Sie (bei vereinbarter Verarbeitung von undefinierten Werten) das über PARAMETER DYNAMIC FORM NULL festgelegt Zeichen in das FHS-Format ein, so wird dieses Zeichen nicht als NULL-Wert, sondern als Zeichen mit echtem Wert ("NOT NULL") verarbeitet.
Um eine einheitliche NULL-Wertverarbeitung in DRIVE-, FHS-Bildschirmformaten (ohne Verarbeitung „undefinierter Werte“) und FHS-Bildschirmformaten (mit Verarbeitung „undefinierter Werte“) sicherzustellen, sollten Sie bei der DRIVE-Anweisung PARAMETER DYNAMIC FORM NULL und bei der Erstellung eines FHS-Formats (ab V6.0) das gleiche Zeichen als NULL-Wertzeichen wählen.

Aus-/Eingabe von NULL-Werten in FHS-Formate ohne „undefinierte Werten“

Wenn Sie bei der Erstellung eines FHS-Formats mit IFG die Verarbeitung von NULL-Werten („undefinierten Werten“) **ausgeschaltet** haben, gilt für die Ein- und Ausgabe von NULL-Werten das Verfahren für DRIVE-Bildschirmformate (siehe oben): Sie legen den NULL-Wert mit PARAMETER DYNAMIC FORM NULL fest. Beachten Sie folgende Besonderheiten:

Bei FHS-Formaten sind als NULL-Wertzeichen für Felder von numerischem Datentyp nur die Ziffern 0 bis 9 und das Zeichen NIL (für die Ausgabe) erlaubt. Die Sonderzeichen * + - , sind nicht zugelassen. Für Felder vom Datentyp DATE müssen NULL-Wertzeichen und Ein-/Ausgabefüllzeichen identisch sein.

Wurde als NULL-Wertzeichen das Zeichen @ vereinbart, so sollte als Ein-/Ausgabefüllzeichen für Maskenfeder auch das Zeichen @ gewählt werden. Andernfalls ist die Eingabe von NULL-Wertzeichen nicht möglich. NULL-Werte werden bei der Ausgabe durch das Ausgabefüllzeichen dargestellt.

5.5 Einschränkungen beim Verwenden von IBM-Terminals

Bei der Verwendung von IBM-Terminals ist von folgenden Einschränkungen auszugehen:

- Bei Bildschirm-Ausgaben über DISPLAY FORM bzw. SEND MESSAGE verlängern sich u.U. die Ausgaben. Dies führt zu Verschiebungen am Bildschirm. Der Anwender muß seine Ausgabedaten verkürzen, bis ein akzeptables Ausgabe-Layout erreicht ist.
- In der 23. Zeile sind die letzten Bytes gesperrt, d.h. es steht nicht der ganze Bildschirm zur Verfügung.
- Fehlermarkierungen führen zur Einstreuung von Leerzeichen. Diese werden jedoch bei einer Bildschirm-Eingabe ignoriert.
- Es gibt keine Endemarke. Deshalb müssen Dialog-Anweisungen mit „;“ abgeschlossen werden, wenn dahinter noch weitere Zeichen am Bildschirm stehen.
- Die Meldung (SEND MESSAGE) ist um 1 bzw. 2 Zeichen nach rechts verschoben.
- Bei der Arbeit mit IFG muß bei Eingabe der Datenstationstypen das IBM-Terminal 3270 immer mit angegeben werden.
- Ist der Einsatz der FHS-Formate auf IBM-Terminals geplant, so darf „Schnelle Formatierung“ nicht angegeben werden, wenn bei „Standardannahme beim Einsatz“ die Datensichtstation 3270 nicht angegeben wird. Bei „Gruppe von Datenstationstypen“ muß die Datensichtstation 3270 angegeben sein. Ansonsten müssen alle Formate nochmals erstellt werden.
- Weitere Einschränkungen siehe Handbücher „IFG“ [28] und „FHS“ [29].

6 DRIVE-Listenformate

DRIVE/WINDOWS bietet Funktionen, mit denen Sie Listenformate erstellen können. Drucklisten werden über DRIVE-Anweisungen programmiert.

In diesem Kapitel erfahren Sie, wie Sie:

– DRIVE-Listenformate erstellen und einsetzen (ab Seite 168)

– Kompakt-Listenformate erstellen und einsetzen (ab Seite 181)

– NULL-Werte in Listenformaten darstellen (ab Seite 182).

6.1 DRIVE-Listenformate erstellen und einsetzen

Das Erstellen und Einsetzen von DRIVE-Listenformaten geschieht analog zum Erstellen und Einsetzen von DRIVE-Bildschirmformaten. Deshalb wird das Thema Listenformate nicht mehr mit der Ausführlichkeit behandelt, wie es im Kapitel „DRIVE-Bildschirmformate“ auf Seite 125 der Fall ist.

DRIVE-Listenformate werden innerhalb eines DRIVE-Programms erstellt. Die Listenformate können Sie somit - ohne DRIVE/WINDOWS zu verlassen - leicht programmieren und, falls erforderlich, ohne großen Aufwand veränderten Anforderungen anpassen.

Die Anweisungen zur Programmierung eines DRIVE-Listenformats sind:

DECLARE LIST listname

Die Anweisung legt einen Speicherbereich für ein Listenformat an, definiert dafür einen Rahmen und legt ein Layout (z.B. Kopfzeilen) fest. Sie muß im Deklarationsteil des Programms stehen.

FILL listname

Die Anweisung füllt den FILL-Bereich (siehe Seite 174) mit Inhalt. Sie kann für beliebige, mit DECLARE LIST definierte Listenformate angegeben werden. Sie muß im Ablaufteil des Programms stehen.

DISPLAY listname

Die Anweisung schließt eine Ausgabeformatierung ab und stößt eine Listenausgabe an. Konkret geschieht folgendes: Die Kopf- und Fußzeilen (siehe Seite 174) werden erst jetzt mit Inhalt gefüllt; dann wird das komplette Listenformat nach SYSLST (im TIAM-Betrieb) bzw. in die zentrale Druckdatei (im UTM-Betrieb) ausgegeben. Die Anweisung muß im Ablaufteil des Programms stehen.

DRIVE/WINDOWS bietet zusätzlich die Möglichkeit, ein Listenformat ad hoc mitten im Ablaufteil eines Programms einzusetzen. Dieses Listenformat wird im Ablaufteil des Programms mit **einer** Anweisung definiert, aufbereitet und ausgegeben:

DISPLAY LIST

In den DRIVE/WINDOWS-Handbüchern wird diese Anweisung auch als Kompaktanweisung und das entsprechende Format als Kompakt-Listenformat bezeichnet, falls es aus Gründen der Eindeutigkeit notwendig ist.

Die folgende Darstellung zeigt die DRIVE-Listenformate und die zur Programmierung dafür notwendigen Anweisungen.

Erstellen und Einsetzen eines DRIVE-Listenformats mit den Anweisungen	
DECLARE LIST listname FILL listname DISPLAY listname	DISPLAY LIST
↓	↓
DRIVE-Listenformat	**Kompakt-Listenformat** (Sonderform des DRIVE-Listenformats)

6.1.1 DRIVE-Listenformate definieren

Mit der Anweisung DECLARE LIST listname (Syntax und Regeln siehe DRIVE-Lexikon [3], Anweisung DECLARE LIST) können Sie ein Listenformat aufbauen.

Festlegen der Listendimension

Sie können mit den Angaben COLUMNS und LINES den Umfang der Listenseite bestimmen. Der Standardwert für COLUMNS ist 132, für LINES 60.

Beispiel

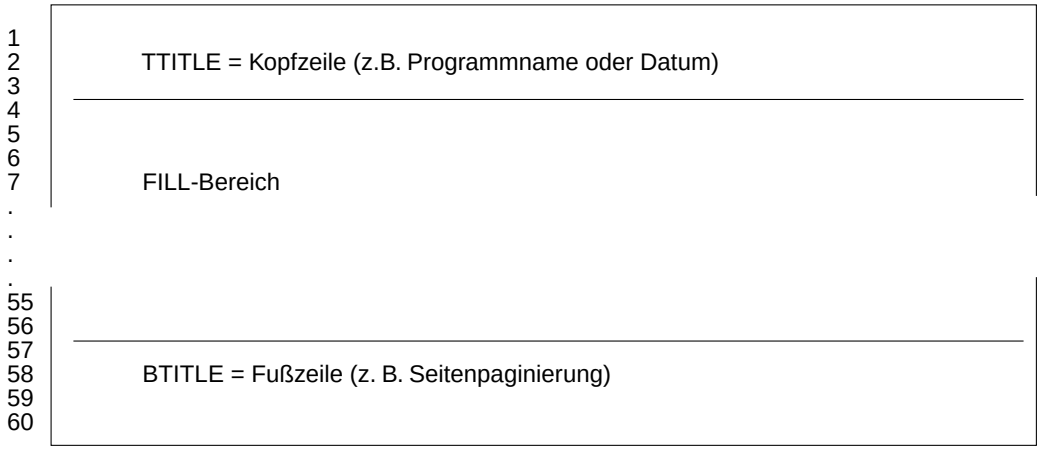
```
DECLARE LIST MITARBEITERAUSGABE COLUMNS 90, LINES 50;
```

Eine Listenseite wird auf 90 Spalten und 50 Zeilen begrenzt.

Festlegen der Kopf- und Fußzeilen

Dazu benötigen Sie die Operanden TTITLE bzw. BTITLE. Pro Seite können maximal n_1 TTITLE + n_2 BTITLE + n_3 FILL = 999 Zeilen definiert werden. Es gilt: $1 \leq n_3 \leq 999$ und $0 \leq n_1, n_2 \leq 998$. Mindestens eine Zeile ist für den FILL-Bereich vorgesehen (siehe Abschnitt „DRIVE-Listenformat ausgeben“ auf Seite 179).

Das folgende Bild zeigt, wie ein Listenformat in Kopf- und Fußzeilen aufgeteilt werden könnte. Der Bereich zwischen Kopf- und Fußzeilen ist der FILL-Bereich.



Definieren von Listenfeldern

Sie können in die Kopf- und Fußzeilen Ausgabe- bzw. Listenfelder setzen. Einschränkungen und weitergehende Regeln lesen Sie bitte nach im DRIVE-Lexikon [3], Anweisung DECLARE LIST.

Ein Auszug aus der Syntaxdarstellung zu DECLARE LIST soll die unten erwähnten Möglichkeiten für das Definieren von Listenfeldern in Kopfzeilen in übersichtlicher Form zusammenfassen (für Fußzeilen analog mit BTITLE):

```
...
[ TTITLE [ format ] { ausdruck [ mask ] |
                      NEWLINE n |
                      TABULATOR n |
                      BLANK n }, ... ]
...
```

Beim Definieren von Listenfeldern stehen folgende Möglichkeiten zur Verfügung:

– Definieren der Listenfelder durch Angabe von *ausdruck* (siehe DRIVE-Lexikon [3], Metavariable *ausdruck*). *ausdruck* kann u.a. sein: Variable, Literal, arithmetischer Ausdruck, Stringfunktion. Bei mehreren Listenfeldern muß ein Komma zwischen den Felddefinitionen stehen, z.B.

```
DCL LIST MITARBEITERAUSGABE TTITLE &DATUM, &ZEIT, 'SEITE: ', &PAGES ...
```

– Einsetzen des Wiederholungsfaktors

  Sie können sich Schreibaufwand ersparen, wenn Sie den Wiederholungsfaktor verwenden. Der Wiederholungsfaktor darf nur bei Literalen verwendet werden. Form: 'literal'(n) (siehe „DRIVE-Lexikon“ [3], Metavariable *literal*). Der Wiederholungsfaktor *n* ist eine ganze Zahl, die in Klammern unmittelbar hinter dem Literal anzugeben ist, z.B.

```
... '␣'(18), ...    → ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣(18 Leerzeichen)
```

– Definieren der Darstellungsmöglichkeit für Listenfelder

  Sie können für Listenfelder mit Hilfe des Operanden MASK unterschiedliche Darstellungsarten definieren. Z.B. das Maskensteuerzeichen „X“ für alphanumerische Datentypen, oder die Unterdrückung der Ausgabe von führenden Nullen mit „Z“ (siehe Abschnitt „MASK-Klausel (Ausgabe aufbereiten)“ auf Seite 56 und „DRIVE-Lexikon“ [3], Metavariable *mask*),z.B.

```
... &DATUM DATE MASK 'Q(10)'', DER ''ZD''. ''R(10)''.''YYYY'
→ MONTAG    , DER 9. JANUAR    1996
```

– Positionieren der Listenfelder

  Sie können die Listenfelder innerhalb der Kopf- bzw. Fußzeilen exakt positionieren mit Hilfe von Angaben wie Leerzeichen, Tabulator oder Zeilenvorschub. Mehrere Positionsangaben hintereinander sind möglich. Sie müssen durch ein Komma getrennt sein, z.B.

```
... BLANK 37 ...    →  in der gleichen Zeile folgen 37 Leerspalten
... NL 2 ...        →  es folgen 2 neue Zeilen (NEWLINE)
... TAB 20 ...      →  Fortsetzung der Ausgabe ab Spalte 20
```

  Es bestehen Abhängigkeiten bezüglich Angaben zur Positionierung der Listenfelder und Angaben zur Datenanordnung über *format* (siehe unten Festlegen der Datenanordnung).

– Ausgeben von Datenbank-Inhalten siehe folgenden Abschnitt.

Festlegen der Datenanordnung

Mit DECLARE LIST *format* können Sie für die Daten der Kopf- und Fußzeilen bestimmen, wie diese auf der Listenseite anzuordnen sind, z.B. eine tabellenförmige Ausgabe. DRIVE/WINDOWS bietet die Wahl zwischen Standard-Listenformaten und freier Layoutsteuerung (siehe DRIVE-Lexikon [3], Metavariable *format*).

Die Standardformate können Sie folgendermaßen definieren:

– Ausgabe erfolgt tabellenförmig (Operand TABLE)
– Ausgabe erfolgt zeilenweise (Operand LINE)
– Ausgabe erfolgt hintereinander angeordnet (Operand SEQUENCE)

Eine freie Layoutsteuerung definieren Sie mit dem Operanden FREE.

Bei Standardformaten und bei freier Layoutsteuerung können Sie zusätzlich beeinflussen, ob folgendes ausgegeben werden soll:

– Variablennamen (Operand NAMES)
– Dateninhalte bzw. -werte der auszugebenden Elemente (Operand VALUES)
– beides (NAMES und VALUES)

Durch Kombinieren dieser Operanden können Sie sich Ihr gewünschtes Listenlayout gestalten. Beispiele zu diesen Operanden finden Sie im nächsten Abschnitt „DRIVE-Listenformate mit Inhalt füllen“ auf Seite 174.

Die freie Layouterstellung erfolgt mit dem Operanden FREE, der standardmäßig vorbelegt ist.

Die Dateninhalte bzw. -werte der auszugebenden Elemente werden lückenlos hintereinander ausgegeben, ohne Rücksicht auf das Zeilenende, d.h. bei Zeilenende erfolgt Umbruch. Die Namen der Dateninhalte werden nicht ausgegeben.

Es bestehen folgende Abhängigkeiten bezüglich Angaben zur Positionierung der Listenfelder und Angaben zur Datenanordnung über *format*:

Wird für *format* TABLE angegeben, sind nachfolgend die Angaben NEWLINE und NEWPAGE nicht erlaubt.

Wird für *format* LINE angegeben, erfolgt bei den Angaben TABULATOR und BLANK ein Zeilenvorschub mit Leerzeile.

Festlegen eines Zeichens für die NULL-Wertdarstellung

Sie können durch Angabe von NULL *nullwert* definieren, wie der NULL-Wert im Listenformat dargestellt werden soll. Eine in der PARAMETER-Anweisung vereinbarte NULL-Wertdarstellung wird hiermit überschrieben. Sie können für ein alphanumerisches und/oder

numerisches Datenfeld je ein NULL-Wertzeichen festlegen. Das NULL-Wertzeichen wird als alphanumerisches Literal der Länge 1 bestimmt (siehe DRIVE-Lexikon [3], Metavariable *nullwert*), z.B.

```
DECLARE LIST MITARBEITERAUSGABE NULL CHARTYPE='␣' NUMTYPE='0'...
```

Festlegen der Lebensdauer eines DRIVE-Listenformats

In einem Unterprogramm (mit CALL aufgerufen) wird ein DRIVE-Listenformat definiert:

– Der Inhalt eines mit PERMANENT definierten Listenformats bleibt über Unterprogrammende hinaus erhalten.

– Der Inhalt eines mit TEMPORARY definierten Listenformats bleibt bei Unterprogrammende nicht erhalten.

Beispiel: Definieren des Listenformats MITARBEITERAUSGABE

```
PROC MITDRUCK;
...
DCL LIST MITARBEITERAUSGABE PERMANENT
 TTITLE                                        /* Beginn der Kopfzeilen    */
 &DATUM,TAB 37,&ZEIT,TAB 110,'SEITE:',&PAGES, /* 1. Zeile                 */
 NL 3,                                         /* 3 neue Zeilen (New Line) */
 TAB 60,'MITARBEITERLISTE',                    /* ab Spalte 60 das Literal */
 NL 1,                                         /* 1 neue Zeile             */
 TAB 59,'-'(18),                               /* ab Spalte 59 Unterstrich */
 NL 2                                          /* 2 neue Zeilen            */
                                               /* Ende der Kopfzeilen      */
 BTITLE                                        /* Beginn der Fußzeilen     */
 '='(132), NL 2;                               /* Doppelstrich in Zeile 58 */
                                               /* dann 2 Leerzeilen        */
...
END PROC;
```

Die Definition des DRIVE-Listenformats mit dem Namen MITARBEITERAUSGABE legt den Speicherbereich für ein Listenformat an, wie im folgenden dargestellt. Die Zeilenlänge reicht hier im Handbuch nicht aus, um die komplette Listenbreite abzubilden.

Bild 1: Auszug aus dem Listenformat Mitarbeiterausgabe

Für die Liste MITARBEITERAUSGABE wurden 6 Kopfzeilen und ein Fußzeilenbereich definiert, der von Zeile 58 bis einschließlich 60 reicht.

6.1.2 DRIVE-Listenformate mit Inhalt füllen

Voraussetzung für das Füllen eines DRIVE-Listenformats mit Inhalt ist die Definition eines Listenformats (siehe Abschnitt „DRIVE-Listenformate definieren“ auf Seite 169).

FILL listname füllt den FILL-Bereich des Speicherbereichs mit Inhalt (Syntax und Regeln siehe DRIVE-Lexikon [3], Anweisung FILL listname). Diese Anweisung können Sie für beliebig viele mit DECLARE LIST definierte Listenformate angeben. Jedes FILL leitet eine neue Zeile ein.

Definieren von Listenfeldern

Die Syntaxdarstellung zu FILL listname soll die Möglichkeiten für das Definieren von Listenfeldern im FILL-Bereich in übersichtlicher Form zusammenfassen.

```
FILL listname [ format ]

     { ausdruck [ mask ] |
       NEWLINE n |
       NEWPAGE n |
       TABULATOR n |
       BLANK n }, ...
```

Beim Definieren von Listenfeldern im FILL-Bereich stehen die gleichen Möglichkeiten zur Verfügung wie beim Definieren von Listenfeldern in den Kopf- und Fußzeilen (siehe Abschnitt „DRIVE-Listenformate definieren“ auf Seite 169). Deshalb wird an dieser Stelle nicht noch einmal darauf eingegangen, mit Ausnahme des nun folgenden Aspekts:

Ausgeben von Datenbank-Inhalten

Datenbank-Inhalte, also Datenbankfelder, können nicht über ihren Namen in einem Format ausgegeben werden. Es müssen Variablen definiert werden, die die gewünschten Werte der Datenbankfelder enthalten. Dies geschieht folgendermaßen:
Eine Variable bekommt den Inhalt eines Cursors zugewiesen. (Eine strukturierte Variable kann ja wie ein Cursor aufgebaut sein.)

Beispiel

Im Unterprogramm MITDRUCK wird ein CURSOR definiert zur Aufnahme der Datenbankinhalte. Zusätzlich muß für DRIVE/WINDOWS eine strukturierte Variable bereitgestellt werden zur Aufnahme des Cursorinhalts.

```
DCL DRUCKCURSOR CURSOR FOR S ABT_MIT_NR,LFD_NR,NACHNAME,VORNAME,
                GEHALT,ABT_LEITER,PROJ_MIT FROM MITARBEITER;
DCL VAR 1 &DRUCKSATZ,
            2 ABT_MIT_NR_ CHAR(4) REDEFINES &AABT_MIT_NR,
            2 LFD_NR_ INT,
            2 NACHNAME_ CHAR(20) ,
            2 VORNAME_ CHAR(20) ,
            2 GEHALT_ NUM(7,2) MASK 'ZZZZ9P99'' DM ''',
            2 ABT_LEITER_  CHAR(8) ,
            2 PROJ_MIT_ CHAR(6);
DCL LIST MITARBEITERAUSGABE;
```

Nun zur Datenausgabe auf Liste: Zuerst sollen die Namen der Satzelemente - ähnlich einer Überschriftenzeile - ausgegeben werden (siehe nachfolgende Beschreibung zur Festlegung der Datenanordnung (*format*)):

```
FILL MITARBEITERAUSGABE TABLE NAMES &DRUCKSATZ;
```

Unter diese Zeile sollen die Datenwerte (VALUES) tabellenförmig (TABLE) angegeben sein:

```
CYCLE DRUCKCURSOR INTO &DRUCKSATZ;
   FILL MITARBEITERAUSGABE TABLE VALUES &DRUCKSATZ.*;
END CYCLE;
...
DISPLAY MITARBEITERAUSGABE;
```

Und so wird - nach erfolgtem DISPLAY - die Anordnung der Dateninhalte auf Liste aussehen:

ABT_MIT_NR_LFD_NR_	NACHNAME_	VORNAME_	GEHALT_	ABT_LEITER_	PROJ_MIT
0108 1	Winterberg	Hannelore	3500.00 DM	01080008	————
0108 2	Mitscherlich	Hermann	2700.00 DM	01080008	————
0108 3	Grenzer	Wilhelm	2300.00 DM	01080008	.....
0108 4	Paarungen	Morten	3200.00 DM	01080008	.....

. . .

Festlegen der Datenanordnung (*format*)

Mit FILL listname *format* können Sie die Anordnung der Daten im FILL-Bereich (nicht in den Kopf- und Fußzeilen) definieren und gleichzeitig füllen. DRIVE/WINDOWS bietet die Wahl zwischen Standardformaten und freier Layoutsteuerung (siehe DRIVE-Lexikon [3], Metavariable *format*).

Die Standardformate können Sie folgendermaßen definieren:

– Ausgabe erfolgt tabellenförmig (Operand TABLE)
– Ausgabe erfolgt zeilenweise (Operand LINE)
– Ausgabe erfolgt hintereinander angeordnet (Operand SEQUENCE)

Eine freie Layoutsteuerung definieren Sie mit dem Operanden FREE.

Bei Standardformaten und bei freier Layoutsteuerung können Sie zusätzlich beeinflussen, ob folgendes ausgegeben werden soll:

– Variablennamen (Operand NAMES)
– Dateninhalte der auszugebenden Elemente, also der Variablen (Operand VALUES)
– beides (NAMES und VALUES)

Durch Kombinieren dieser Operanden können Sie Ihr gewünschtes Layout gestalten. Die Voreinstellung ist FREE.

Hat die FILL-Anweisung einen Pufferüberlauf zur Folge, so wird implizit eine DISPLAY-Anweisung ausgeführt (siehe Abschnitt „DRIVE-Listenformat ausgeben“ auf Seite 179) ausgeführt.

Beispiele

Sowohl Variablennamen als auch Datenwerte sollen in Tabellenform ausgegeben werden (eine Programmschleife sorgt dafür, daß mehrere Sätze gelesen werden):

```
CYCLE ...
FILL L1 TABLE NAMES VALUES &ADRESSE ...
END CYCLE;
```

```
NAME_____________VORNAME_________STRASSE_________
HUBER            MAX             AUERSTR.
NAME_____________VORNAME_________STRASSE_________
MEIER            GERHARD         APESSARTSTR.
NAME_____________VORNAME_________STRASSE_________
SEILER           FRANZ-GEORG     OBERE_MUEHLRADS

.    ...
.    ...
.    ...                                                                .
```

Diese Darstellungsform erzeugt zu den Datenwerten jeweils auch die Variablennamen (≅ Überschriftenzeile). Wollen Sie mehrere Werte unter einer einzigen Überschriftenzeile, müssen Sie NAMES und VALUES durch eine Schleife „trennen“ (siehe Beispiel zum Definieren von Listenfeldern):

```
FILL L1 TABLE NAMES &ADRESSE ...
CYCLE ...
FILL L1 TABLE VALUES &ADRESSE ...
END CYCLE;
```

```
NAME_____________VORNAME_________STRASSE_________
HUBER            MAX             AUERSTR.
MEIER            GERHARD         APESSARTSTR.
SEILER           FRANZ-GEORG     OBERE_MUEHLRADS

.   ...
.   ...
.   ...                                                                 .
```

Nur Variablennamen sollen zeilenweise ausgegeben werden:

```
FILL L1 LINE NAMES &ADRESSE ...
```

```
NAME␣:
VORNAME␣:
STRASSE␣:
.   ...
.   ...
.   ...
```

Sowohl Variablennamen als auch Datenwerte sollen hintereinander (horizontal) ausgegeben werden:

```
FILL L1 SEQUENCE NAMES VALUES &ADRESSE ...
```

NAME␣:	␣HUBER	␣VORNAME␣:	␣MAX	␣STRASSE␣:	␣AUENSTR.
NAME␣:	␣MEIER	␣VORNAME␣:	␣GERHARD	␣STRASSE␣:	␣SPESSARTSTR.
NAME␣:	␣SEILER	␣VORNAME␣:	␣FRANZ-GEORG	␣STRASSE␣:	␣OBERE_MUEHLRADS

```
.    ...
.    ...
.    ...
```

Die freie Layouterstellung erfolgt mit dem Operanden FREE, der standardmäßig vorbelegt ist:

Die Dateninhalte bzw. -werte der auszugebenden Elemente (≅ Variablen) werden lückenlos hintereinander ausgegeben, ohne Rücksicht auf das Zeilenende. Die Variablennamen werden nicht ausgegeben:

```
FILL L1 FREE ...
```

```
HUBER          MAX           AUENSTR.          MEIER           GERHARD
SPE
SSARTATR.     SEILER             FRANZ-GEORG          OBERE_MUEHLRADS
.   ...
.   ...
.   ...
```

6.1.3 DRIVE-Listenformat ausgeben

Mit der Anweisung DISPLAY listname wird das DRIVE-Listenformat im TIAM-Betrieb nach SYSLST, im UTM-Betrieb in die zentrale Druckdatei ausgegeben.
Nach erfolgter Ausgabe bewirkt die Anweisung DISPLAY listname kein Löschen des Speichers für das ausgegebene DRIVE-Listenformat. Ein Listenformat kann somit mehrmals hintereinander ausgegeben werden, wenn zwischen den DISPLAY-Anweisungen kein neues FILL-Kommando gegeben wurde.

```
DISPLAY listname;
```

Nach dieser Anweisung erfolgt die tatsächliche Ausgabe des mit DECLARE LIST definierten und in der Regel mit FILL listname gefüllten DRIVE-Listenformats nach SYSLST (TIAM) bzw. in die zentrale Druckdatei (UTM).

Beispiel: Programm zur Erstellung und Ausgabe einer Liste

Hauptprogramm MITARBEITER.HAUPT

```
PROCEDURE MITARBEITER;
COPY MITVAR;                              /* Einfügen des COPY-Elements MITVAR */
DECLARE VARIABLE &ENDE2 CHAR(1);
DECLARE SCREEN FHSBILD;
CYCLE;
  DISPLAY FHSBILD;
  IF &WAHL = 'O' THEN BREAK CYCLE;
  END IF;
  ...
  IF &WAHL = '5' THEN CALL MITDRUCK; /* Aufruf des Unterprogrammes MITDRUCK */
  SET &WAHL = ' ';
  END IF;
END CYCLE;
COMMIT WORK;
END PROCEDURE;
```

COPY-Element MITVAR

```
DECLARE VARIABLE &FRAGE PERMANENT CHAR(1) INIT 'N';
DECLARE MITARBEITER CURSOR FOR S ALL * FROM MITARBEITER;
DECLARE DRUCKCURSOR CURSOR FOR S ABT_MIT_NR,LFD_NR,NACHNAME,VORNAME,
                    GEHALT,ABT_LEITER,PROJ_MIT FROM MITARBEITER;
DECLARE VARIABLE &MITARBEITERSATZ LIKE CURSOR MITARBEITER,
                 &INDEX INT INIT 1,
                 &NUMMER INT,
                 &VGL CHAR(4),
                 &DATUM DATE
                   MASK 'Q(10)'',DER ''ZD''.''R(10)''.''YYYY',
```

```
                    &ZEIT TIME
                      MASK 'ZH'' UHR ''ZI',
                    1 &DRUCKSATZ,
                        2 ABT_MIT_NR_ CHAR(4) REDEFINES &AABT_MIT_NR,
                        2 LFD_NR_ INT,
                        2 NACHNAME_ CHAR(20) ,
                        2 VORNAME_ CHAR(20) ,
                        2 GEHALT_ NUM(7,2) MASK 'ZZZZ9P99'' DM ''',
                        2 ABT_LEITER_  CHAR(8) ,
                        2 PROJ_MIT_ CHAR(6) ,
                    &ABFRAGEBEFEHL CHAR(15) INIT 'DISPLAY FORM ',
                    &ABFRAGETEXT1 CHAR(74) INIT
'NL 14,TAB 5,''GEBEN SIE DIE ABT_MIT_NR EIN : '',RETURN &VGL,',
                    &ABFRAGETEXT2 CHAR(74) INIT
'NL  2,TAB 5,''GEBEN SIE DIE LAUFENDE NUMMER EIN : '',RETURN &NUMMER;',
                    &FEHLERMELDUNG CHAR(74) INIT
'SEND MESSAGE ''  DER DATENSATZ IST NICHT VORHANDEN. DUE-TASTE'';',
                    &DRUCKTEXT CHAR(74) INIT
'NL 15,TAB 15,''SOLL JETZT GEDRUCKT WERDEN ? (J/N) : '',RETURN &FRAGE';
```

Unterprogramm MITDRUCK:

```
PROCEDURE MITARBEITERDRUCK;
COPY MITVAR;
DECLARE LIST MITARBEITERAUSGABE
    TTITLE &DATUM,TAB 37,&ZEIT,TAB 110,'SEITE:',&PAGES,NL 3,
           TAB 60,'MITARBEITERLISTE',NL 1,
           TAB 59,'-'(18),NL 2
    BTITLE NL 2,'='(123);
  FILL MITARBEITERAUSGABE TABLE NAMES &DRUCKSATZ;
  CYCLE DRUCKCURSOR INTO &DRUCKSATZ.*;
     FILL MITARBEITERAUSGABE TABLE VALUES &DRUCKSATZ;
  END CYCLE;
  DISPLAY MITARBEITERAUSGABE;
  EXEC CONCAT(&ABFRAGEBEFEHL,&DRUCKTEXT);
  IF &FRAGE = 'J' THEN
    SYSTEM 'PRINT *SYSLST';
    SEND MESSAGE 'DIE MITARBEITERLISTE WIRD AUSGEDRUCKT. DUE-TASTE';
    SET &FRAGE = 'N';
  ELSE
    SEND MESSAGE
    'DIE MITARBEITERLISTE WIRD AM ENDE DER BS2000-SITZUNG GEDRUCKT.';
  END IF;
END PROCEDURE;
```

Und so sieht die erzeugte DRIVE-Liste aus (Die ganze Listenbreite kann im Handbuch nicht dargestellt werden.):

```
MONTAG   , DER   9. JANUAR   1996     12 UHR 30
SEITE 1
                          MITARBEITERLISTE
                          ________________

ABT_MIT_NR_LFD_NR_  NACHNAME_     VORNAME_   GEHALT_      ABT_LEITER_    PROJ_MIT
0108              1  Winterberg    Hannelore  3500.00 DM  01080008       ________
0108              2  Mitscherlich  Hermann    2700.00 DM  01080008       ________
0108              3  Grenzer       Wilhelm    2300.00 DM  01080008       ........
0108              4  Paarungen     Morten     3200.00 DM  01080008       ........
.  ...             .  ...          ...        ...         ...            ...        .
.  ...             .  ...          ...        ...         ...            ...        .
.  ...             .  ...          ...        ...         ...            ...        .


==============================================================================
```

6.2 Kompakt-Listenformat erstellen und einsetzen

Mit der Anweisung DISPLAY LIST können Sie Listenformate im Ablaufteil eines Programms ad hoc definieren, füllen und ausgeben (Syntax und Regeln siehe DRIVE-Lexikon [3], Anweisung DISPLAY LIST).

Diese Anweisung enthält implizit die DECLARE LIST-, die FILL- und die DISPLAY listname-Anweisungen. Statt drei Anweisungen genügt somit eine. Das Kompakt-Listenformat ist eine Sonderform des DRIVE-Listenformats.

DISPLAY LIST ist vor allem dann sinnvoll einzusetzen, wenn einmalige Listenausgaben erwünscht sind. Ein einmal ausgegebenes Kompakt-Listenformat kann nicht noch einmal eingesetzt werden, außer es wird nochmals mit DISPLAY LIST definiert und ausgegeben. Es existiert ja kein Listenformatname.

Der Aufbau eines Kompakt-Listenformats erfolgt nach den gleichen Regeln wie der beim DRIVE-Listenformat. Beachten Sie, daß die Kompaktanweisung DISPLAY LIST implizit ein DECLARE und FILL enthält.

6.3 NULL-Werte in Listenformaten

Mit der Anweisung PARAMETER DYNAMIC NULL LIST legen Sie für ein alphanumerisches und/oder numerisches Datenfeld je ein Zeichen für die Darstellung des NULL-Werts bei Listenformaten fest (siehe DRIVE-Lexikon [3], Anweisung PARAMETER DYNAMIC).

Das NULL-Wertzeichen wird durch ein alphanumerisches Literal (charliteral) der Länge 1 bestimmt (siehe DRIVE-Lexikon [3], Metavariable *nullwert*).

Für alphanumerische Datenfelder kann als NULL-Wertzeichen (charliteral1) jedes beliebige Zeichen verwendet werden. Für numerische Datenfelder ist als NULL-Wertzeichen (charliteral2) nur eine Ziffer oder eines der Sonderzeichen * + - , . zugelassen.

Das NULL-Wertzeichen für alphanumerische Datenfelder (charliteral1) ist auch gültig für Felder der Zeit-Datentypen. Das NULL-Wertzeichen für numerische Datenfelder (charliteral2) ist auch gültig für Felder des Datentyps INTERVAL.

Das mit PARAMETER DYNAMIC vereinbarte NULL-Wertzeichen kann für DRIVE-Listenformate durch Angabe eines anderen NULL-Wertzeichens in der NULL-Klausel der DECLARE LIST-Anweisung überschrieben werden.
Ein NULL-Wertzeichen, das über DECLARE LIST NULL festgelegt wurde, gilt nur für das jeweilige mit DECLARE LIST definierte Listenformat.

Wenn Sie kein NULL-Wertzeichen explizit festlegen, gilt für alphanumerische oder numerische Datenfelder der Punkt „.“ als Standardvorgabewert.

7 Report-Generator

Der Report-Generator unterstützt das Erzeugen von Listen, die auf einem Ausgabegerät ausgegeben werden. Diese Listen können z.B. Verkaufsstatistiken in immer wiederkehrendem Format, Lagerbestände oder Fahrtenberichte enthalten. Solche Listen heißen Reports. Das Ausgabegerät für einen Report kann ein Drucker oder eine Datei sein.

Der Report-Generator bietet Ihnen Möglichkeiten zur Gestaltung von

– Standard-Reports

– individuellen Reports

Standard-Reports sind einfach und schnell zu generieren, aber durch Standard-Vorgaben in der Gestaltung eingeschränkt. Individuelle Reports können Sie vollständig nach Ihren Wünschen gestalten.

In diesem Kapitel ist beschrieben:

– wie das Generieren eines Reports abläuft (ab Seite 183)

– wie Sie Standard-Reports generieren (ab Seite 196)

– welche Arten von individuellen Reports möglich sind (ab Seite 202)

– welche Strukturen beim Gestalten der Seite eines individuellen Reports zu berücksichtigen sind (ab Seite 205)

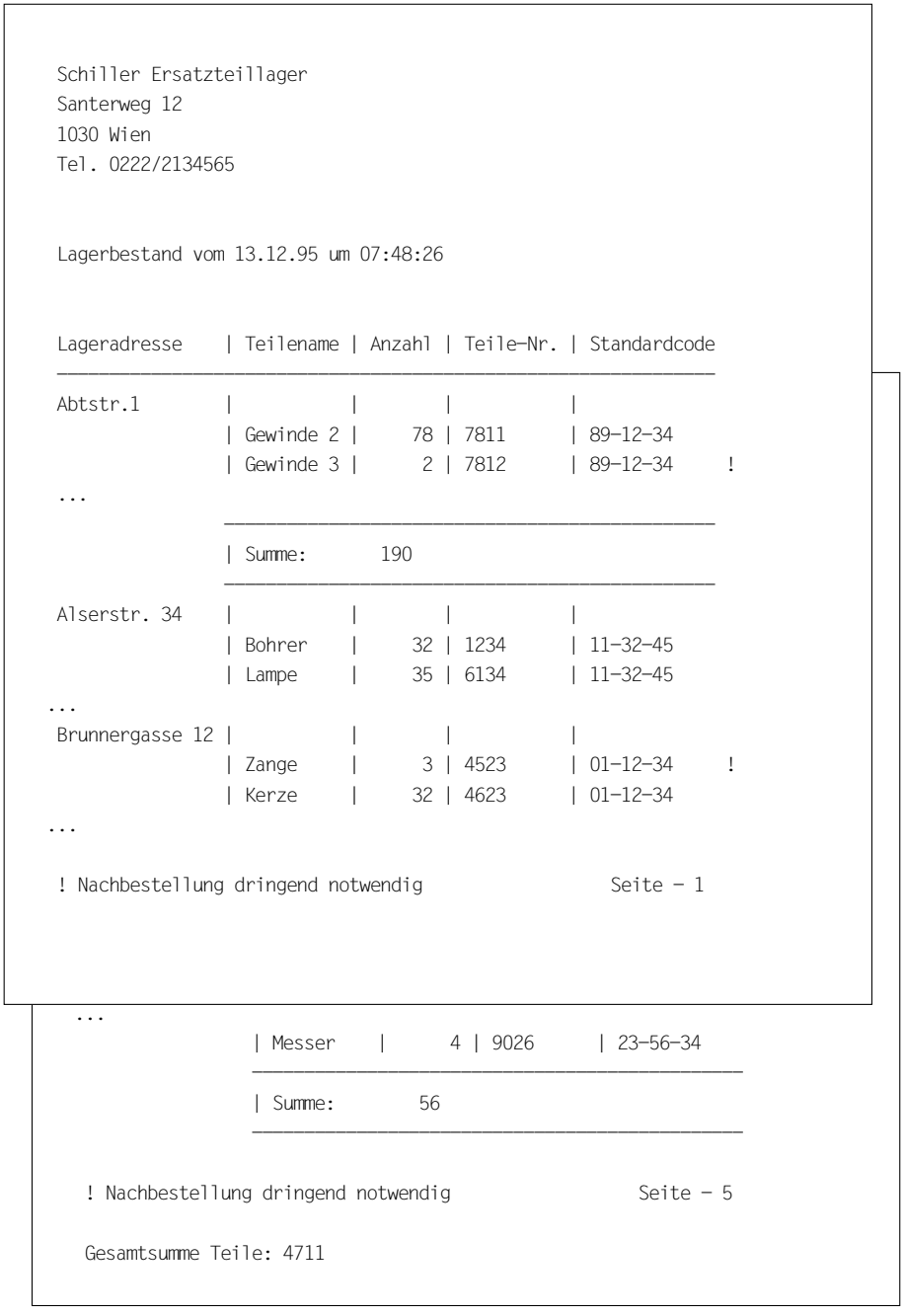
– wie Sie einen individuellen Report generieren (ab Seite 225)

– welche Systemumgebung Voraussetzung für den Report-Generator ist (ab Seite 236)

7.1 Ablauf einer Report-Generierung

In diesem Abschnitt erfahren Sie, welche Daten zum Generieren eines Reports verwendet werden können, wie die Report-Generierung in ein DRIVE-Programm eingebettet wird und wie Sie beim Ausführen eines Reports vorgehen. Außerdem sind die Report- und DRIVE-Anweisungen aufgelistet, die Sie bei der Report-Generierung verwenden dürfen.

Die folgende Abbildung zeigt ein Beispiel für einen individuellen Report:



```
Schiller Ersatzteillager
Santerweg 12
1030 Wien
Tel. 0222/2134565


Lagerbestand vom 13.12.95 um 07:48:26


Lageradresse    | Teilename | Anzahl | Teile-Nr. | Standardcode
________________________________________________________________
Abtstr.1        |           |        |           |
                | Gewinde 2 |     78 | 7811      | 89-12-34
                | Gewinde 3 |      2 | 7812      | 89-12-34     !
...
                ________________________________________________
                | Summe:       190
                ________________________________________________
Alserstr. 34    |           |        |           |
                | Bohrer    |     32 | 1234      | 11-32-45
                | Lampe     |     35 | 6134      | 11-32-45
...
Brunnergasse 12 |           |        |           |
                | Zange     |      3 | 4523      | 01-12-34     !
                | Kerze     |     32 | 4623      | 01-12-34
...

! Nachbestellung dringend notwendig                 Seite - 1
```

```
  ...
                | Messer    |      4 | 9026      | 23-56-34
                ________________________________________________
                | Summe:        56
                ________________________________________________

  ! Nachbestellung dringend notwendig                 Seite - 5

  Gesamtsumme Teile: 4711
```

Wenn Sie einen Report generieren wollen, sind zunächst folgende Überlegungen notwendig:

– welche Daten sollen in einem Report verarbeitet werden
– welche Daten sollen ausgegeben werden
– welche Satzarten liegen vor
– wie soll die Ausgabe aussehen

Wenn Sie wissen, welche Daten in einem Report verarbeitet und ausgegeben werden sollen, und das Layout entworfen ist, müssen Sie den Entwurf umsetzen. Zum Umsetzen sind folgende Schritte notwendig:

– Definieren des Reports
– Ausführen des Reports

7.1.1 Report-Daten

Report-Daten sind die Daten, die gemeinsam mit der Beschreibung des Reports zur Generierung der Ausgabeliste dienen. Das können sein:

– Nettodaten, mit denen Sie den Report bei der Ausführung versorgen
– lokale Daten, die Sie innerhalb der Report-Definition benutzen
– Layout-Daten, die Sie zur Gestaltung des Reports verwenden

Nettodaten

Die Nettodaten sind die Daten, mit denen Sie den Report bei der Ausführung durch die Anweisungen OPEN REPORT und/oder FILL REPORT versorgen. Die Nettodaten beschreiben Sie in der Anweisung DECLARE REPORT, um zu definieren wie diese Daten verarbeitet oder ausgegeben werden sollen. Mit der LIKE-Klausel können Sie Datenbeschreibungen aus dem DRIVE-Programm in die Report-Definition übernehmen.

Außerdem können Sie in der Anweisung DECLARE REPORT als Startparameter ein oder mehrere einfache Variablen definieren, die einmalig beim Öffnen des entsprechenden Report-Puffers mit Werten versorgt werden. Diese Variablen stehen Ihnen dann auch innerhalb der Report-Definition zur Verfügung.

Zur Beschreibung der Nettodaten haben Sie zwei Möglichkeiten:

1. Sie übernehmen die Beschreibung der Daten aus dem DRIVE-Programm z.B. mit:

```
DECLARE REPORT name USING &rep_var LIKE &drive_var;
```

2. Sie beschreiben die Daten in der Anweisung DECLARE REPORT

```
DECLARE REPORT name USING &rep_var1 datentyp_a,
                          &rep_var2 datentyp_b,
                          &rep_var3 datentyp_c;
```

Sie müssen bei der Beschreibung der Daten darauf achten, daß die in der FILL REPORT-Anweisung verwendeten Nettodaten in die mit der Anweisung DECLARE REPORT beschriebenen Typen konvertierbar sind.

Folgende Daten wurden aus einem DRIVE-Programm in einen Report übernommen und für den Beispiel-Report auf Seite 184 aufbereitet:

```
Haringasse 11       Becher              4 6255 11-21-33
Alserstr. 34        Bohrer             32 1234 11-32-45
Dorfplatz 2         Gabel              44 4711 11-21-34
Abtstr. 1           Gewinde 2          78 7811 89-12-34
Abtstr. 1           Gewinde 3           2 7812 89-12-34
Alserstr. 34        Hacke               2 1134 11-32-45
Haringasse 11       Hammer              3 5655 11-01-33
Haringasse 11       Kabel              23 5155 11-91-33
Brunnergasse 12     Kerze              32 4623 01-12-34
Abtstr. 1           Kette               2 7162 89-12-34
Abtstr. 1           Kette 4             2 7163 89-12-34
Rahlgasse 12        Kleber              2 8806 23-56-34
Alserstr. 34        Lampe              35 6134 11-32-45
Rahlgasse 12        Messer              4 9026 23-56-34
Rahlgasse 12        Niete               5 8906 23-56-34
Abtstr. 1           Reifen             99 7112 89-12-34
Dorfplatz 2         Schaufel           11 3181 11-21-34
Rahlgasse 12        Schlauch           45 3111 23-56-34
Dorfplatz 2         Schlauch           66 3111 11-21-34
Abtstr. 1           Sitz                7 8122 89-12-34
Brunnergasse 12     Speiche            31 5213 01-12-34
Brunnergasse 12     Teer                3 4213 01-12-34
Haringasse 11       Zahnrad            23 5555 12-01-33
Brunnergasse 12     Zange               3 4523 01-12-34
```

Lokale Daten

Lokale Daten sind die Daten, die Sie nur innerhalb der Report-Definition verwenden. Das können sein:

– Nettodaten und Startparameter, die Sie in der Anweisung DECLARE REPORT beschrieben haben

– Die Inhalte der Systemvariablen &PAGES und &LINES

– Die Inhalte von lokalen Variablen

  Lokale Variablen müssen Sie innerhalb der Report-Definition definieren (siehe DRIVE-Lexikon [3], Anweisung DECLARE VARIABLE für Reports). Solche Variablen dürfen nicht außerhalb der Report-Definition verwendet werden. Lokalen Variablen können Sie beispielsweise Anfangswerte für Schleifenzähler, Ergebnisse von Report-Mengenfunktionen oder Rückgabewerte von C-Modul- Aufrufen zuweisen.

  Lokale Variablen müssen von einfachem Typ sein. Der Typ INTERVAL ist nicht erlaubt. Auf lokale Variablen dürfen Sie keine INIT-, CHECK-, LIKE- oder REDEFINES-Klauseln anwenden.

– Die Ergebnisse von Report-Mengenfunktionen

  Folgende Funktionen werden unterstützt: COUNT, MIN, MAX, AVG und SUM. Die Beschreibung der Report-Mengenfunktionen finden Sie im „DRIVE-Lexikon“ [3], Kapitel Report-Anweisungen.

– Logische und arithmetische Ausdrücke

  Um logische oder arithmetische Ausdrücke zu bilden, stehen folgende Operatoren zur Verfügung:

  Vergleichsoperatoren: = <> < > <= >=

  Logische Operatoren: AND, OR

  Arithmetische Operatoren: + – * /

– Ergebnisse von C-Modul-Aufrufen

  Innerhalb einer Report-Definition können Sie C-Module aufrufen. Wenn z.B. ein Modul verfügbar ist, der Maßeinheiten umrechnet, kann das Ergebnis einer lokalen Variablen zugewiesen und für die weitere Bearbeitung verwendet werden.

– Die Inhalte von Textdateien

  Inhalte von Textdateien können Sie unverändert an beliebiger Stelle im Report ausgeben lassen. Dazu müssen Sie in der Report-Definition den Namen der Textdatei bekannt geben (siehe „DRIVE-Lexikon“ [3], Anweisung SOURCE).

Layout-Daten

Layout-Daten können ganzzahlige Werte oder Zeichenketten-Konstanten sein (z.B. Fixtexte oder Semigrafiken). Diese Daten spezifizieren Sie ebenfalls innerhalb der Report-Definition. Sie dienen zur Gestaltung einer Ausgabeseite und werden direkt oder mit einem symbolischen Feldnamen bei der entsprechenden Anweisung angegeben. Im Beispiel-Report auf Seite 184 wurde z.B. die Zeichenkette

„! Nachbestellung dringend notwendig“

in der Anweisung PAGE TRAILER für den Seitenfuß angegeben (siehe Abschnitt „Kontrollstruktur“ auf Seite 208).

Einen symbolischen Feldnamen geben Sie dann an, wenn sprachspezifische Fixtexte in einer MIP-Datei unter dem Namen hinterlegt sind.

7.1.2 Ausgabedaten

Ausgabedaten sind die Daten, die Sie tatsächlich in einem Report ausgeben. Das können ebenfalls sein:

– Nettodaten
– lokale Daten
– Layout-Daten

Mit der PRINT-Anweisung können Sie für jedes Datenfeld festlegen:

– die Ausgabeposition
– das Ausgabeformat
– die Darstellungsattribute

Ausgabeposition

Bei einem Standard-Report sind die Ausgabepositionen der Daten abhängig von dem Format, das Sie gewählt haben (siehe Abschnitt „Standard-Reports“ auf Seite 196).

Bei einem individuellen Report wird normalerweise an der aktuellen Position weiter ausgegeben. Sie können die Ausgabeposition aber auch für jedes Feld gesondert bestimmen. Dazu geben Sie die Position innerhalb einer Druckzeile in cm oder inch oder in einer Spaltenanzahl an. Außerdem haben Sie die Möglichkeit, eine Ausgabeposition relativ zum Ende des letzten Ausgabefeldes zu bestimmen.

Ausgabeformat

Bei einem Standard-Report wird die Darstellung der auszugebenden Daten vom Datentyp der einzelnen Ausgabefelder bestimmt (siehe Abschnitt „Standard-Reports“ auf Seite 196).

Bei einem individuellen Report sind alle Ausgabeformate zulässig, außer den Maskensteuerzeichen, die im „DRIVE-Lexikon“ [3], Kapitel Report-Anweisungen, Abschnitt Einschränkungen für Report-Parameter beschrieben sind. Wenn Sie kein Ausgabeformat spezifizieren, dann wird die Datendarstellung ebenfalls vom Typ der Daten bestimmt.

Darstellungsattribute

Bei individuellen Reports können Sie Darstellungsattribute für den gesamten Report voreinstellen oder für jedes Ausgabefeld gesondert festlegen (siehe „DRIVE-Lexikon“ [3], Anweisung PRINT, format-klausel). Folgende Darstellungsattribute stehen zur Verfügung:

– Proportionalschrift
– Schriftart
– Schriftgröße
– Zeichendichte
– Zeichenabstand
– Dehnung (horizontal, vertikal)
– Nationaler Zeichensatz
– Fettdruck
– Kursivdruck
– Unterstreichen
– Tiefstellen (halbe Zeile) *
– Hochstellen (halbe Zeile) *
– Inversdarstellung
– Zeilenabstand
– Blinken **
– Hell bzw. Halbhell **
– Vordergrundfarbe
– Hintergrundfarbe

* nur PRINT-Anweisung

** dient nur der Bildschirmausgabe und wird derzeit nicht ausgewertet

7.1.3 Satzarten

Liegen die Nettodaten, die Sie in einem Report verarbeiten wollen, in unterschiedlichen Satzarten vor, so müssen Sie dies in der Anweisung DECLARE REPORT bekannt geben.

Unterschiedliche Satzarten identifizieren Sie durch den Inhalt eines Kennfeldes, das in allen Satzarten an der gleichen Position beginnt und vom gleichen Datentyp sein muß. Als Kennfeld voreingestellt ist das erste Feld eines Satzes. Sie können aber auch ein anderes Feld wählen. Unterscheidet sich beispielsweise das jeweils erste Feld von zwei Satzarten durch die Inhalte „satz-a“ und „satz-b“ geben Sie dies bekannt mit:

```
DECLARE REPORT name RECORD TYPE 'satz-a' USING &rep_var_a LIKE &drive_a,
                   RECORD TYPE 'satz-b' USING &rep_var_b LIKE &drive_b;
```

Damit sind die unterschiedlichen Satzarten im Report definiert.

Beim Bearbeiten der Daten unterschiedlicher Satzarten und beim Vorbereiten der Ausgabe müssen Sie innerhalb der Report-Definition die Bezeichnung angeben, die Sie der Satzart in der DECLARE REPORT-Anweisung zugeordnet haben und ggf. den Inhalt des Kennfeldes (siehe „DRIVE-Lexikon“ [3], Anweisung DETAIL). Stimmt bei der Bearbeitung der Daten beispielsweise der Inhalt des Kennfeldes mit der Bezeichnung satz-a bzw. satz-b überein, werden die Anweisungen ausgeführt, die Sie der jeweiligen Satzart zugeordnet haben:

```
DETAIL 'satz-a' IN &rep_var_a.kennfeld-a
       PRINT TAB 15, 'Satz von Satzart A ', &rep_var_a;

DETAIL 'satz-b' IN &rep_var_b.kennfeld-b
       PRINT TAB 45, 'Satz von Satzart B ', &rep_var_b;
```

Sie können bei der Bearbeitung einer Satzart auf den letzten Satz der anderen Satzarten zugreifen durch Angabe des Inhalts des Kennfelds. Sie müssen dafür Sorge tragen, daß bei diesem Zugriff die Sätze dem Report bekannt sind.

Aus folgenden Gründen ist es nicht sinnvoll, Satzarten zu gruppieren:

- Es wird über alle Satzarten sortiert, nicht innerhalb jeder Satzart.
- Durch die automatische Sortierung beim Gruppieren sollte bei der Bearbeitung einer Satzart nicht auf eine andere Satzart zugegriffen werden.

Beim Ausführen des Reports müssen Sie dann ebenfalls angeben, daß der Report mit Daten aus unterschiedlichen Satzarten versorgt wird (siehe „DRIVE-Lexikon“ [3], Anweisung FILL REPORT).

```
FILL REPORT name RECORD TYPE 'satz-a' USING &drive_a,
                RECORD TYPE 'satz-b' USING &drive_b;
```

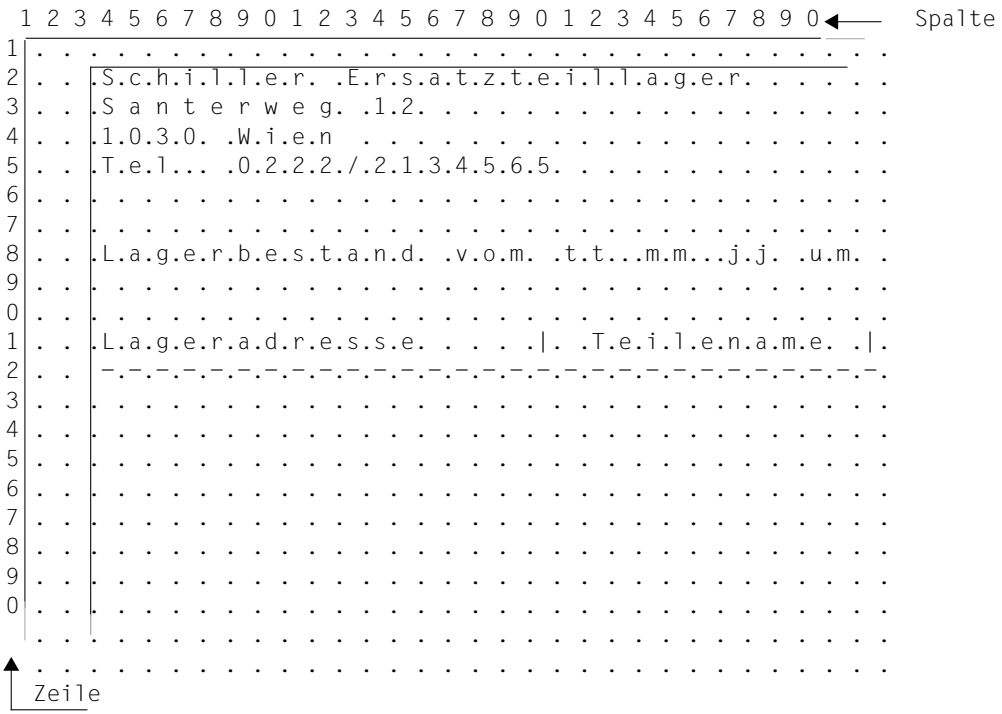
Geben Sie für &drive einzelne Felder an, müssen Sie darauf achten, daß die Reihenfolge und die Datentypen den Vereinbarungen entsprechen, die Sie in der Report-Definition festgelegt haben (&rep_var).

7.1.4 Ausgabe gestalten

Der Platzbedarf für die Ausgabe der Daten wird in Anzahl Zeilen und Spalten bemessen. Zur Berechnung des zur Verfügung stehenden Platzes verwenden Sie eine Standard-Seite. Die Größe einer Standard-Seite für den Drucker, den Sie verwenden wollen, ist in einem Geräteprofil hinterlegt (siehe Abschnitt „Geräteprofil“ auf Seite 237). Die Anzahl der möglichen Spalten ist von der im Geräteprofil angegebenen Spaltenbreite und die Anzahl der möglichen Zeilen vom Zeilenabstand abhängig.

Bei einem Standard-Report ist die Breite der Ränder fest vorgegeben. Die Nutzung des verbleibenden Platzes wird durch den Typ der Daten bestimmt, die Sie ausgeben. Dabei haben Sie Einfluß darauf, ob die Daten im Tabellenformat, im vertikalen Format oder im horizontalen Format ausgegeben werden (siehe Abschnitt „Standard-Reports“ auf Seite 196).

Die folgende Abbildung zeigt, wie Sie mithilfe eines Rasters eine Ausgabeseite des Reports entwerfen.



Bei einem individuellen Report können Sie die Gestaltung einer Ausgabeseite vollständig beeinflussen. Sie haben beispielsweise die Möglichkeit, Randbreiten, Überschriften und Druckpositionen für bestimmte Daten festzulegen (siehe Abschnitt „Struktur individueller Reports“ auf Seite 205).

Für den oben dargestellten Beispiel-Report wurde für den oberen und unteren Rand eine Breite von einer Zeile und für den linken und rechten Rand eine Breite von drei Spalten festgelegt. Die Ausgabe der Adresse beginnt in Zeile 1 Druckposition 1 der tatsächlich zur Verfügung stehenden Ausgabeseite.

7.1.5 Report definieren

Die Datenstruktur und das Layout eines Reports definieren Sie mit Anweisungen, die mit DECLARE REPORT beginnen und mit END REPORT enden. Eine Report-Definition steht im Deklarationsteil eines DRIVE-Programms.

In einer Report-Definition:

– legen Sie einen Namen für die Report-Definition fest (name)
– beschreiben Sie die Nettodaten (&report_var und &drive_var)
– definieren Sie lokal zu bearbeitende Daten (daten-bereich)
– beschreiben Sie das Layout (layout-bereich)
– veranlassen Sie die Bearbeitung der Report-Daten

Report-Name festlegen

Mit der Anweisung DECLARE REPORT erhält ein Report seinen Namen (name). Aus dem Report-Namen und dem Namen des DRIVE-Programms wird zur Übersetzungs- bzw. Laufzeit der Name der Report-Definitionsdatei gebildet. Die Report-Definitionsdatei wird in der

DRIVE-Bibliothek als Element vom Typ X abgelegt. Der Name dieser Datei leitet sich aus dem Programmnamen (maximal 5 Zeichen) und dem Report-Namen (7 Zeichen) mit der Namenserweiterung .r in folgender Form ab:

programmname.report-name.r

Ist der Programmname kürzer als 5 Zeichen, werden fehlende Zeichen durch X ergänzt. Enthält der Name Sonderzeichen werden diese ebenfalls durch X ersetzt. Der Name, den Sie der Report-Definition geben, muß den BS2000-Konventionen für Dateinamen entsprechen (siehe „BS2000 Benutzerkommandos“ [35]). Für die Eindeutigkeit des Namens der Report-Definitionsdatei müssen Sie sorgen, da sie von DRIVE/WINDOWS nicht gewährleistet wird.

Nettodaten beschreiben

Mit der Angabe FOR START USING der Anweisung DECLARE REPORT definieren Sie Startparameter, die nur einmal am Beginn der Report-Ausführung mit Werten versorgt werden (siehe „DRIVE-Lexikon“ [3], Anweisung OPEN REPORT). Mit der USING-Klausel der Anweisung DECLARE REPORT definieren Sie Report-Parameter (&report_var) oder übernehmen die Beschreibung der Report-Parameter aus dem DRIVE-Programm (&drive_var).

Lokale Daten definieren

Innerhalb einer Report-Definition ist für die Beschreibung von Daten, die lokale Bedeutung haben (daten-bereich), folgende Anweisung erlaubt:

DECLARE VARIABLE zur Definition von Variablen innerhalb eines Reports (z.B. für Schleifenzähler oder zur Aufnahme von Ergebnissen von Report-Mengenfunktionen).

Layout beschreiben

Bei der Beschreibung des Layout-Bereichs müssen Sie sich zunächst entscheiden, ob Sie einen Standard-Report oder einen individuellen Report generieren wollen.

Bei der Definition eines Standard-Reports beschreiben Sie das Layout mit der Report-Anweisung STANDARD LAYOUT (siehe Abschnitt „Standard-Reports“ auf Seite 196).

Bei der Definition eines individuellen Reports haben Sie verschiedene Möglichkeiten: Entweder Sie legen mit der der Anweisung GLOBAL LAYOUT Einstellungen für den gesamten Report fest, wie z.B. Seitenränder oder die Schriftart (siehe Abschnitt „Struktur individueller Reports“ auf Seite 205).

Oder Sie verzichten auf Einstellungen für den gesamten Report. In diesem Fall müssen Sie jedoch mindestens eine der folgenden Report-Anweisungen verwenden:

REPORT	zur Beschreibung des Listenkopfes und -fußes
PAGE	zur Beschreibung des Seitenkopfes und -fußes
GROUP	zur Beschreibung eines Gruppenkopfes und -fußes
DETAIL	zur Beschreibung von Detailzeile

Report-Daten bearbeiten

Neben den beschreibenden Anweisungen für den Daten- und den Layout-Bereich, können Sie in einer Report-Definition mit der Anweisung:

SOURCE	eine Textdatei an einer bestimmten Stelle einfügen
PRINT	festlegen, welche Daten oder Layout-Elemente an welcher Position, in welcher Form mit welchen Attributen ausgegeben werden

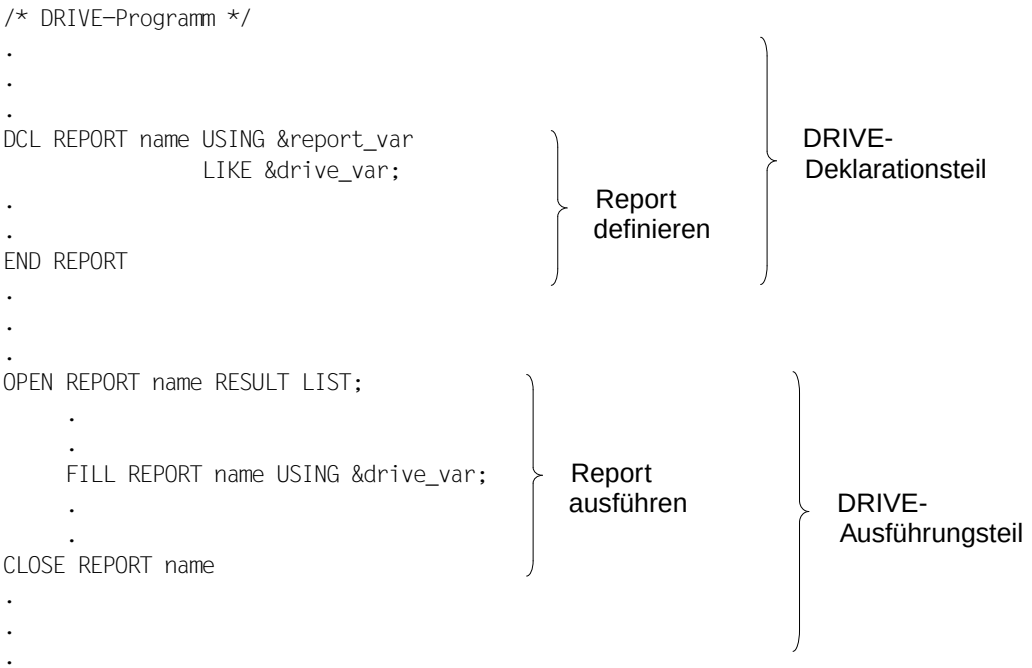
Zusätzlich zu den oben aufgezählten Anweisungen dürfen Sie in einer Report-Definition noch folgende DRIVE-Anweisungen verwenden:

BREAK CYCLE	um eine Schleife abzubrechen
CALL C MODULE	um in C geschriebene Module einzubinden
CYCLE	um Schleifen zu programmieren
IF	um Bedingungen zu programmieren
SET	um Werte zuzuweisen

Bei der Report-Generierung gelten für einige dieser Anweisungen Einschränkungen. Diese Einschränkungen sind im DRIVE-Lexikon [3], Kapitel Report-Anweisungen.

7.1.6 Report ausführen

Das Ausführen eines oder mehrerer Reports beginnt mit der Anweisung OPEN REPORT und endet mit der Anweisung CLOSE REPORT. Ein solcher Anweisungsblock muß im ausführbaren Teil eines DRIVE-Programms stehen.



Mit der Anweisung OPEN REPORT veranlassen Sie das Öffnen eines Report-Puffers und spezifizieren ggf. das Ausgabegerät und den Gerätetyp. In den geöffneten Report-Puffer werden mit der Anweisung FILL REPORT die auszugebenden Daten (&drive_var) gemäß der verwendeten Report-Definition (name) eingetragen.

Zur Laufzeit des DRIVE-Programms wird durch das Schließen eines Report-Puffers die Ausgabe des Reports ausgelöst. Der Report-Puffer wird nach der Ausführung des Programms gelöscht.

7.1.7 Ausgabegeräte

Einen Report können Sie auf einem Drucker oder in eine Datei schreiben.

Bei der Ausgabe auf einem Drucker werden gerätespezifische Steuerzeichen aus einem entsprechenden Geräteprofil gelesen (siehe Abschnitt „Geräteprofil“ auf Seite 237).

Bei der Ausgabe in eine Datei werden gerätespezifische Steuerzeichen mit in die Datei geschrieben. Drucken Sie die Datei mit dem BS2000-Kommando PRINT-DOCUMENT aus, damit diese gerätespezifischen Steuerzeichen ausgewertet werden:

```
/PRINT-DOCUMENT FROM-FILE=...,DOCUMENT-FORMAT=*TEXT(LINE-SPACING=*BY-EBCDIC-CONTROL)
```

Das Ausgabegerät oder die Ausgabe in eine Datei fordern Sie in der Anweisung OPEN REPORT an.

7.2 Standard-Reports

In diesem Abschnitt werden Ihnen die drei Formate vorgestellt, die zur Generierung eines Standard-Reports zur Verfügung stehen:

– das Tabellenformat

– das vertikale Format

– das horizontale Format

Standard-Reports bieten Ihnen z.B. die Möglichkeit, sich die auszugebenden Daten anzuschauen, um später mit diesen Daten einen individuellen Report zu gestalten.

Einen Standard-Report fordern Sie mit der Anweisung STANDARD LAYOUT an. Das ist die einzige Anweisung, die zwischen DECLARE REPORT und END REPORT notwendig ist.

Wenn Sie einen Standard-Report nur mit der Anweisung STANDARD LAYOUT generieren, sind Randbreiten und Darstellungsattribute (z.B. Schrifttyp) von den Voreinstellungen des gewählten Ausgabegeräts abhängig. Standardmäßig erzeugt der Report-Generator einen Kopfbereich, mit einer Seitenüberschrift. Dieser Kopfbereich ist drei Zeilen breit und enthält das aktuelle Datum. Im Tabellenformat wird im Kopfbereich auch noch die Tabellenüberschrift ausgegeben. Die übrige Ausgabeseite steht Ihnen in dem Umfang zur Verfügung, wie er im Geräteprofil des gewählten Ausgabegeräts für eine Standard-Seite definiert ist (siehe Abschnitt „Geräteprofil“ auf Seite 237).

Das Definieren einer Seitenstruktur oder von Kontrollblöcken für einen späteren individuellen Report, z.B. für Seitenkopf und Seitenfuß oder Gruppenkopf und Gruppenfuß, ist in einer Report-Definition für einen Standard-Report nicht erlaubt (siehe Abschnitt „Seitenstruktur“ auf Seite 205). Solche Definitionen führen zu Fehlermeldungen.

Beim Generieren eines Standard-Reports werden die Definition der Ausgabepositionen aus der Satzbeschreibung der auszugebenden Daten und vom gewählten Standard-Format bestimmt.

Für die Beispiele in den nächsten Abschnitten werden folgende Daten verwendet:

abt_mit_nr	lfd_nr	nachname	vorname	gehalt	sprachen(4)			
0106	1	Sennert	Gustl	5800.00	ENG			
0106	2	Bergmann	Susanne	6900.00	ENG			
0107	1	Olsen	Matthes	4000.00	ENG			
0107	2	Mattsen	Gunter	4800.00	ENG			
0107	3	Nonnen	Paul	4500.00				
0107	4	Dormagen	Siegfried	5500.00	FRA	SPA		
0108	1	Winterberg	Hannelore	3500.00	ENG			
0109	1	Mitscher	Fred	4900.00	FRA			
0109	2	Jansen	Jutta	4700.00				
0109	3	Zimmermann	Peter Johannes	5800.00	ENG	SPA		

7.2.1 Tabellenformat

Im Tabellenformat werden die Daten eines Ausgabesatzes nebeneinander in eine Zeile geschrieben. Das bedeutet, daß das Tabellenformat nur dann sinnvoll ist, wenn alle Daten eines Ausgabesatzes in eine Zeile passen.

Bei einem Zeilenüberlauf erfolgt Programmabbruch.

Die Breite eines Feldes in einem Ausgabesatz richtet sich entweder nach dem Feldnamen oder dem längst möglichen Feldinhalt. Das längere von beiden wird verwendet. Zwischen den einzelnen Feldern werden drei Leerzeichen ausgegeben. Das ist eine Voreinstellung, die Sie durch Angabe einer FILL-Klausel überschreiben können (siehe „DRIVE-Lexikon“ [3], Anweisung STANDARD LAYOUT).

Bei eindimensionalen Datenstrukturen (Vektoren und Wiederholungsgruppen) werden die Feldnamen mit dem Wiederholungsfaktor ausgegeben.

Matrizen und Datenstrukturen mit mehr als einer Dimension (z.B. Vektoren in Wiederhlungsgruppen) werden nicht ausgegeben. Sie werden durch die DRIVE-Meldung `DRI7121 Datenstruktur im Standard-Layout ausgelassen`darüber informiert.

Beispiel

Sie generieren mit den oben beschriebenen Daten einen Standard-Report im Tabellenformat mit dem Namen gliste. Die Felder eines Ausgabesatzes sollen mit der Zeichenfolge Leerzeichen, Längsstrich (|), Leerzeichen voneinander getrennt werden.

```
...
DECLARE c1 CURSOR FOR SELECT abt_mit_nr, lfd_nr, nachname, vorname, gehalt
                          FROM tab1;
DECLARE VARIABLE &satz LIKE CURSOR c1;

DECLARE REPORT gliste USING &rsatz LIKE CURSOR c1;
   STANDARD LAYOUT TABLE FILL ' | ';
END REPORT;

OPEN REPORT gliste RESULT LIST;
CYCLE c1 INTO &satz.*;
   FILL REPORT gliste USING &satz;
END CYCLE;
CLOSE REPORT gliste;
...
```

Ausgabe:

```
abt_mit_nr   lfd_nr        nachname          vorname          gehalt
_____________________________________________________________________
0106       |          1 | Sennert         | Gustl          |   5800.00
0106       |          2 | Bergmann        | Susanne        |   6900.00
0107       |          1 | Olsen           | Matthes        |   4000.00
0107       |          2 | Mattsen         | Gunter         |   4800.00
0107       |          3 | Nonnen          | Paul           |   4500.00
0107       |          4 | Dormagen        | Siegfried      |   5500.00
0108       |          1 | Winterberg      | Hannelore      |   3500.00
0109       |          1 | Mitscher        | Fred           |   4900.00
0109       |          2 | Jansen          | Jutta          |   4700.00
0109       |          3 | Zimmermann      | Peter Johannes |   5800.00
```

Sie verwenden in dem gleichen Programm den Vektor „spachen(4)“, um einen Cursor zu definieren.

```
...
DECLARE c1 CURSOR FOR SELECT nachname, vorname, sprachen(1-3) FROM tab1;
...
```

Ausgabe:

```
nachname          vorname          sprachen(1)   sprachen(2)   sprachen(3)
──────────────────────────────────────────────────────────────────────────
Sennert         | Gustl          | eng         |             |
...
Nonnen          | Paul           |             |             |
Dormagen        | Siegfried      | fra         | spa         |
Winterberg      | Hannelore      | eng         |             |
Mitscher        | Fred           | fra         |             |
Jansen          | Jutta          |             |             |
Zimmermann      | Peter Johannes | eng         | spa         | fra
```

7.2.2 Vertikales Format

Im vertikalen Format werden alle Feldinhalte untereinander aufgelistet. Eine Zeile enthält die Hierarchiestufe, den Feldnamen und den jeweiligen Feldinhalt. Feldname und Feldinhalt sind durch einen Doppelpunkt und ein Leerzeichen voneinander getrennt.

Ist ein Feldname und der aktuelle Feldinhalt so lang, daß der Platz in einer Zeile nicht ausreicht, werden die Daten am Ende der Ausgabezeile abgeschnitten.

Bei eindimensionalen Datenstrukturen (Vektoren und Wiederholungsgruppen) werden die Feldinhalte, getrennt durch ein Leerzeichen, hintereinander in einer Zeile ausgegeben.

Matrizen und Datenstrukturen mit zwei Dimensionen (z.B. Vektoren in Wiederholungsgruppen) werden in mehreren Zeilen im Matrixformat ausgegeben.

Datenstrukturen mit mehr als zwei Dimensionen werden nicht ausgegeben. Sie werden durch die DRIVE-Meldung `DRI7121 Datenstruktur im Standard-Layout ausgelassen`darüber informiert.

Im vertikalen Format werden Matrizen und Wiederholungsgruppen ausgegeben. Matrizen erscheinen im Matrixformat.

Beispiel

Sie generieren mit den oben beschriebenen Daten einen Standard-Report im vertikalen Format mit dem Namen gliste.

```
...
DECLARE c1 CURSOR FOR SELECT abt_mit_nr, lfd_nr, nachname, vorname, gehalt
                          FROM tab1;
DECLARE VARIABLE &satz LIKE CURSOR c1;

DECLARE REPORT gliste USING &rsatz LIKE CURSOR c1;
   STANDARD LAYOUT LINE;
END REPORT;
```

```
OPEN REPORT gliste RESULT LIST;
CYCLE c1 INTO &satz.*;
   FILL REPORT gliste USING &satz;
END CYCLE;
CLOSE REPORT gliste;
...
```

Ausgabe:

```
1 rsatz
2 abt_mit_nr: 0106
2 lfd_nr:           1
2 nachname: Sennert
2 vorname: Gustl
2 gehalt:   5800.00
1 rsatz
2 abt_mit_nr: 0106
2 lfd_nr:           2
2 nachname: Bergmann
2 vorname: Susanne
2 gehalt:   6900.00
1 rsatz
2 abt_mit_nr: 0107
...
```

Sie verwenden in dem gleichen Programm den Vektor „spachen(4)“, um einen Cursor zu definieren.

```
...
DECLARE c1 CURSOR FOR SELECT nachname, vorname, sprachen(1-3) FROM tab1;
...
```

Ausgabe:

```
1 rsatz
2 nachname: Sennert
2 vorname: Gustl
2 sprachen: eng
...
1 rsatz
2 nachname: Nonnen
2 vorname: Paul
2 sprachen:
1 rsatz
2 nachname: Dormagen
2 vorname: Siegfried
2 sprachen: fra spa
1 rsatz
...
```

7.2.3 Horizontales Format

Im horizontalen Format werden die Feldnamen und die aktuellen Feldinhalte der Ausgabesätze fortlaufend gedruckt. Zwischen Feldname und Feldinhalt steht ein Doppelpunkt und ein Leerzeichen, hinter dem Feldinhalt steht ein Leerzeichen.

Bei einem Zeilenüberlauf erfolgt Programmabbruch.

Bei eindimensionalen Datenstrukturen (Vektoren und Wiederholungsgruppen) werden die Feldnamen mit dem Wiederholungsfaktor ausgegeben.

Matrizen und Datenstrukturen mit mehr als einer Dimension (z.B. Vektoren in Wiederhlungsgruppen) werden nicht ausgegeben. Sie werden durch die DRIVE-Meldung `DRI7121 Datenstruktur im Standard-Layout ausgelassen` darüber informiert.

Beispiel

Sie generieren mit den oben beschriebenen Daten einen Standard-Report im horizontalen Format mit dem Namen gliste.

```
...
DECLARE c1 CURSOR FOR SELECT abt_mit_nr, lfd_nr, nachname, vorname, gehalt
                           FROM tab1;
DECLARE VARIABLE &satz LIKE CURSOR c1;

DECLARE REPORT gliste USING &rsatz LIKE CURSOR c1;
   STANDARD LAYOUT SEQUENCE;
END REPORT;

OPEN REPORT gliste RESULT LIST;
CYCLE c1 INTO &satz.*;
   FILL REPORT gliste USING &satz;
END CYCLE;
CLOSE REPORT gliste;
...
```

Ausgabe:

```
abt_mit_nr: 0106 lfd_nr:          1 nachname: Sennert         vorname: Gustl      ...
abt_mit_nr: 0106 lfd_nr:          2 nachname: Bergmann        vorname: Susanne    ...
abt_mit_nr: 0107 lfd_nr:          1 nachname: Olsen           vorname: Matthes    ...
abt_mit_nr: 0107 lfd_nr:          2 nachname: Mattsen         vorname: Gunter     ...
abt_mit_nr: 0107 lfd_nr:          3 nachname: Nonnen          vorname: Paul       ...
abt_mit_nr: 0107 lfd_nr:          4 nachname: Dormagen        vorname: Siegfried  ...
abt_mit_nr: 0108 lfd_nr:          1 nachname: Winterberg      vorname: Hannelore  ...
...
```

Sie verwenden in dem gleichen Programm den Vektor „spachen(4)“, um einen Cursor zu definieren.

```
...
DECLARE c1 CURSOR FOR SELECT nachname, vorname, sprachen(1–3) FROM tab1;
...
```

Ausgabe:

```
nachname: Sennert         vorname: Gustl           sprachen(1): eng sprachen(2):   sprachen( ...
...
nachname: Nonnen          vorname: Paul            sprachen(1):   sprachen(2):   sprachen(3) ...
nachname: Dormagen        vorname: Siegfried       sprachen(1): fra sprachen(2): spa sprache ...
nachname: Winterberg      vorname: Hannelore       sprachen(1): eng sprachen(2):   sprachen( ...
nachname: Mitscher        vorname: Fred            sprachen(1): fra sprachen(2):   sprachen( ...
nachname: Jansen          vorname: Jutta           sprachen(1):   sprachen(2):   sprachen(3) ...
nachname: Zimmermann      vorname: Peter Johannes  sprachen(1): eng sprachen(2): spa sprache ...
```

7.3 Arten individueller Reports

In diesem Abschnitt werden die beiden Arten individueller Reports vorgestellt, die der Report-Generator unterscheidet. Das sind:

– Hierarchisch gegliederte Reports

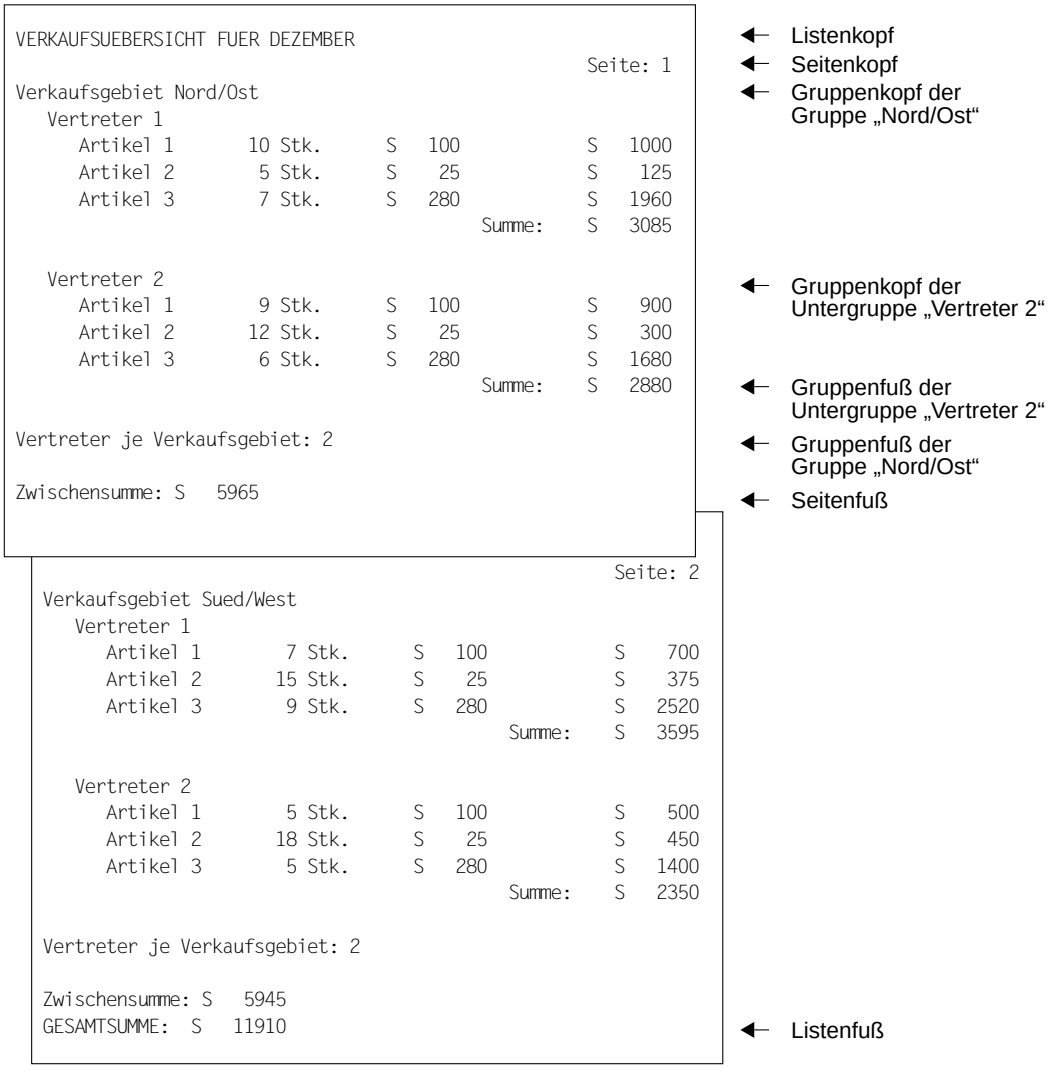
– Formulare

Individuelle Reports können Sie frei gestalten. Sie bestimmen das Aussehen einer Seite und können z.B. für jeden Report ein Deckblatt und eine Abschlußseite generieren.

7.3.1 Hierarchisch gegliederte Reports

Ein hierarchisch gegliederter Report zeichnet sich durch eine hierarchische Gruppenstruktur aus und umfaßt ein oder mehrere Ausgabeseiten. Eine hierarchische Gruppenstruktur liegt dann vor, wenn Sie für die Ausgabe Datengruppen bilden können, die voneinander abhängig sind.

Die folgende Abbildung zeigt beispielhaft einen hierarchisch gegliederten Report, der zwei Seiten umfaßt.



Der Report enthält am Beginn, als Überschrift für die gesamte Liste, einen Listenkopf. Am Ende der Liste ist als Zusammenfassung des Listeninhalts ein Listenfuß ausgegeben. Dem Report ist eine Seitenstruktur überlagert. Reservierte Zeilen am Seitenanfang und am Seitenende enthalten einen Seitenkopf bzw. einen Seitenfuß.

In diesem Report sind hierarchisch gegliederte Gruppen erkennbar:

– Die Gruppen mit den Inhalten Nord/Ost und Sued/West (Feld „Verkaufsgebiet“).

– Die Untergruppen mit den Inhalten Vertreter 1 und Vertreter 2 (Feld „Vertreter), die je Verkaufsgebiet arbeiten.

7.3.2 Formulare

Reports ohne hierarchische Gruppenstruktur heißen Formulare. Ein Report kann aus einer Folge von Formularen bestehen, wobei Sie auf einer Seite ein oder mehrere Formulare generieren können. Die hier gemeinten Formulare sind vorzugsweise eine Folge von unabhängigen Einzelblättern (z.B. Zahlscheine, Bestellscheine oder, wie in der Abbildung unten, Haushaltslisten).

In der folgenden Abbildung ist beispielhaft ein Formular dargestellt.

```
HAUSHALTSLISTE                              Stand: 1/96

 Lfd. Nummer: 5814-280753
 Fam.Name:    Bergermaier              Geschl.: m
 Vorname(n):  Gustav Adolf        Familienstand: v

 Ehegatte:    Sylvia             Geb.Datum: 160457
 Geb.Name:    Koppensteiner            Geschl.: w

 Wohnort:     Wien
 PLZ: 1010  Strasse: Kaerntnergasse 11/3/7

 Kinder: 2
   1: Elisabeth   Geb.Datum: 170186
   2: Michael     Geb.Datum: 281187


Formular FA-OE-4711/80-N-1/007
```

Das Layout eines Formulars ist in keiner Hinsicht eingeschränkt. Auch für ein Formular kann ein Listenkopf und -fuß sowie Seitenkopf und -fuß definiert werden. Das kann bei mehrseitigen Formularen, wie z.B. bei Standard-Verträgen, nützlich sein.

Die Ausgabedaten für Formulare, die aus einer Folge von Einzelblättern bestehen, werden jeweils aus einem Datensatz bezogen. Zwischen einzelnen Formularen besteht kein inhaltlicher Zusammenhang.

7.4 Struktur individueller Reports

In diesem Abschnitt erfahren Sie, welche Strukturierungshilfen der Report-Generator zur Gestaltung individueller Reports zu Verfügung stellt und wie Sie diese Strukturen verwenden. Strukturierungshilfen sind Anweisungen

– zur Strukturierung einer Ausgabeseite
– zur kontrollierten Zuordnung von Daten

Wenn Sie einen individuellen Report generieren wollen, teilen Sie zunächst eine Seite in Bereiche auf und definieren eine Seitenstruktur. Mit diesen Bereichen sind Kontrollblöcke verknüpft, denen Sie anschließend die zu bearbeitenden Daten und die auszugebenden Daten zuordnen. Dadurch definieren Sie eine Kontrollstruktur für die Ausgabe der Daten.

Für die Beispiele in diesem Abschnitt wurden die Daten aus dem Beispiel-Report auf Seite 184 und Report-Daten, Seite 186 verwendet. Die Datendeklaration im DRIVE-Programm sieht wie folgt aus:

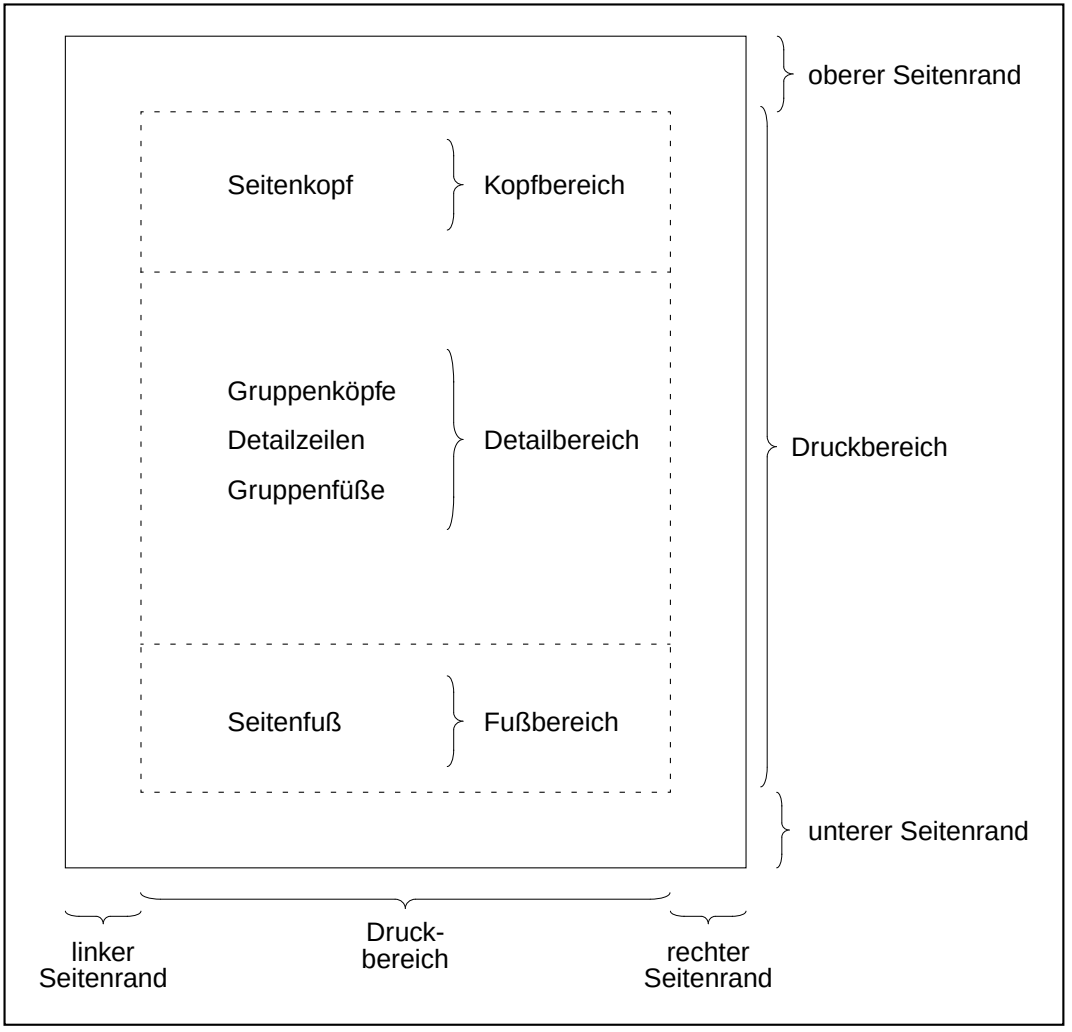
```
DECLARE VARIABLE 1 &drive_var,
                  2 teilenummer    INT,
                  2 teilename      VARCHAR (20),
                  2 anzahl         SMALLINT,
                  2 standardcode   CHAR (8),
                  2 lagername      VARCHAR (20);
```

7.4.1 Seitenstruktur

Die Strukturelemente, mit denen Sie eine Seite in Bereiche aufteilen, sind:

– die Seitenränder
– der Kopfbereich für die Ausgabe des Seitenkopfes
– der Detailbereich für die Ausgabe von Gruppenköpfen, Detailzeilen, Gruppenfüßen
– der Fußbereich für die Ausgabe des Seitenfußes

Die folgende Abbildung zeigt die Seitenstruktur eines individuellen Reports:



Der Listenkopf überlagert den Kopfbereich auf der ersten Seite. Passen Listenkopf und Seitenkopf in den Kopfbereich, werden auf Seite 1 beide ausgegeben. Ist dies nicht der Fall, wird der Seitenkopf auf Seite 1 unterdrückt. Für den Listenfuß gilt bezüglich des Fußbereichs auf der letzten Seite dasselbe.

Enthält eine Seite außer dem Listenfuß/-kopf keinen Detailbereich, hat sie weder Seitenkopf noch Seitenfuß. Nimmt der Listenkopf eine ganze Seite ein, wird auf der ersten Seite weder Seitenkopf noch Seitenfuß ausgegeben. Dasselbe gilt auch für den Listenfuß auf der letzten Seite des Reports.

Die Breite der Seitenränder und die Breiten für den Kopf- und Fußbereich definieren Sie mit der Anweisung GLOBAL LAYOUT. Die Breite des Detailbereichs wird bei der Report-Generierung intern berechnet aus:

```
   Anzahl der Zeilen einer Ausgabeseite
-  Anzahl der Zeilen für den oberen Rand
-  Anzahl der Zeilen für den unteren Rand
-  Anzahl der Zeilen für den Kopfbereich
-  Anzahl der Zeilen für den Fußbereich
_____________________________________________________
Anzahl der verbleibenden Zeilen für den Detailbereich
```

Die Anzahl der verbleibenden Zeilen müssen Sie nicht angeben. Mit der Anweisung GLOBAL LAYOUT können Sie aber festlegen, wieviel Zeilen mindestens für den Detailbereich auf einer Ausgabeseite zur Verfügung stehen müssen. Durch diese Angabe verhindern Sie z.B. bei einem Wechsel des Papierformats, daß auf einer Seite nur noch Seitenköpfe und -füße ausgegeben werden.

Die Größen für die Bereiche geben Sie wieder, ausgehend von einer Standard-Seite, mit der Anzahl Zeilen bzw. Spalten an (siehe Abschnitt „Ablauf einer Report-Generierung“ auf Seite 183).

Beispiel

Sie wollen einen Report auf einer DIN A4-Seite im Hochformat ausgeben. Der rechte und der linke Rand soll jeweils 5 Spalten, der obere und untere Rand 1 Zeile breit sein. Für den Kopfbereich reservieren Sie 11 Zeilen und für den Fußbereich 4. Für den Detailbereich sollen mindestens 30 Zeilen zur Verfügung stehen.

```
/* DRIVE-Programm */
.
.
.
DCL REPORT lager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT
      TOP MARGIN 2                          /* oberer Rand          */
      BOTTOM MARGIN 1                       /* unterer Rand         */
      LEFT MARGIN 5                         /* linker Rand          */
      RIGHT MARGIN 5                        /* rechter Rand         */
      HEADER LINES 11                       /* Kopfbereich          */
```

```
    TRAILER LINES 4                          /* Fußbereich           */
    MINIMUM LINES 30                         /* Minimum Detailbereich */

    ORDER BY &report_var.lagername;          /* Sortieren            */
.
.
.
```

In der Anweisung GLOBAL LAYOUT können Sie auch angeben, ob und nach welchen Kriterien Daten sortiert werden sollen. Hier wurden die Datensätze nach den Lageradressen in aufsteigender Folge sortiert.

7.4.2 Kontrollstruktur

Wenn Sie für einen individuellen Report eine Seitenstruktur definiert haben, bestimmen Sie, welche Daten in den einzelnen Bereichen der Seitenstruktur ausgegeben werden sollen. Damit die Daten auch in dem gewünschten Bereich ausgegeben werden, z.B. Seitennumerierung im Fußbereich, legen Sie eine Kontrollstruktur an. Die Kontrollstruktur beschreibt, welche Informationen auszugeben sind:

– auf der ersten Seite eines Reports
– an jedem Seitenbeginn im Kopfbereich
– im Detailbereich
– an jedem Seitenende im Fußbereich
– auf der letzten Seite eines Reports

Die Verknüpfung zwischen Seitenstruktur und Kontrollstruktur stellen Sie mit Kontrollblökken her. Folgende Kontrollblöcke stehen zur Verfügung:

Listenkopf	für Daten, die nur einmal am Report-Anfang ausgegeben werden
Listenfuß	für Daten, die nur einmal am Report-Ende ausgegeben werden
Seitenkopf	für Daten, die an jedem Seitenanfang ausgegeben werden
Seitenfuß	für Daten, die an jedem Seitenende ausgegeben werden
Gruppenkopf	für Daten, die an jedem Gruppenanfang ausgegeben werden
Gruppenfuß	für Daten, die an jedem Gruppenende ausgegeben werden
Detailzeilen	für gruppenspezifische Ausgabedaten

Außer dem Kontrollblock für Detailzeilen sind die Kontrollblöcke optional. Für die Ausgabe eines individuellen Reports muß für jede Satzart ein Detailkontrollblock definiert sein. Für die Ausgabe eines Standard-Reports müssen Sie eines der Formate des Standard-Layouts

definieren (siehe Abschnitt „Standard-Reports“ auf Seite 196). Für den Report-Generator impliziert jeder Kontrollblock den Beginn einer neuen Zeile. Jeder definierte Kontrollblock muß mindestens eine Anweisung enthalten.

Innerhalb der Kontrollblöcke weisen Sie Daten zur Bearbeitung zu oder Sie definieren, was mit den Daten passieren soll. Sie können beispielsweise die Ausgabe über Bedingungen steuern oder Schleifen definieren, in denen eine Reihe von Anweisungen mehrmals ausgeführt werden.

Formulare werden im Detailbereich oder in einem der Standard-Layouts definiert.

7.4.3 Listenkopf

Einen Listenkopf definieren Sie mit der Anweisung REPORT HEADER (siehe „DRIVE-Lexikon“ [3], Anweisung REPORT). Dort können Sie beispielsweise eine Überschrift für die gesamte Liste festlegen, eine Adresse angeben oder die Ausgabe des aktuellen Datums veranlassen. Ein Listenkopf darf enthalten:

– Literale und mit der Anweisung SOURCE eingefügte Texte
– Systemvariablen
– Startparameter
– arithmetische Ausdrücke
– Daten des ersten Nettodatensatzes
– lokale Variablen

(Siehe Abschnitt „Report-Daten“ auf Seite 185.)

Ein Listenkopf wird genau einmal am Beginn des Reports ausgegeben. Ein Listenkopf kann ein- oder mehrzeilig sein.

Beispiel

Der Listenkopf des Reports soll auf der linken Seite die Adresse und die Telefonnummer enthalten. Zwei Zeilen darunter soll die Bezeichnung der Liste mit dem aktuellen Datum und der aktuellen Uhrzeit stehen:

```
/* DRIVE-Programm */
.
.
.
DCL REPORT lager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT ...;          /* Seitenstruktur festlegen */
   .
   .
   .
```

```
REPORT HEADER
       PRINT TAB 1, 'Schiller Ersatzteillager',
             NEWLINE 1,
             TAB 1, 'Santerweg 12',
             NEWLINE 1,
             TAB 1, '1180 Wien',
             NEWLINE 1,
             TAB 1, 'Tel. 0222/213456',
             NEWLINE 2,
             TAB 1, 'Lagerbestand vom ',
                    CURRENT DATE MASK 'DD''.''MO''.''YYYY',
                    ' um ',
                    CURRENT TIME MASK 'HH'':''MI'':''SS';
.
.
.
```

Listenkopf

Mit der PRINT-Anweisung wurde für den Listenkopf festgelegt:

– die Ausgabepositionen der nachfolgenden Zeichenketten (z.B. TAB 1, '1180 Wien')

– Zeilenvorschübe (z.B. NEWLINE 1)

– das aktuelle Datum mit Ausgabeformat (CURRENT DATE MASK 'DD"."MO"."YYYY'

– die aktuelle Uhrzeit mit Ausgabeformat (CURRENT TIME MASK 'HH":"MI":"SS')

Ausgabe:

```
Schiller Ersatzteillager
Santerweg 12
1030 Wien
Tel. 0222/2134565


Lagerbestand vom 13.12.95 um 07:48:26
```

7.4.4 Listenfuß

Einen Listenfuß definieren Sie mit der Anweisung REPORT TRAILER (siehe „DRIVE-Lexikon“ [3], Anweisung REPORT). Dort können Sie sich beispielsweise Summeninformationen über die gesamte Druckliste ausgeben lassen. Ein Listenfuß darf enthalten:

- Literale und mit SOURCE eingefügte Texte
- Systemvariablen
- Startparameter
- Report-Mengenwerte, die aus den Gesamtdaten errechnet werden
- arithmetische Ausdrücke
- Daten des letzten Nettodatensatzes
- lokale Variablen

(Siehe Abschnitt „Report-Daten“ auf Seite 185.)

Ein Listenfuß wird genau einmal pro Report am Ende des Reports ausgegeben. Ein Listenfuß kann ein- oder mehrzeilig sein.

Beispiel

Auf der letzten Seite des Reports soll die Anzahl aller Teile ausgegeben werden, die in den verschiedenen Lagern noch vorhanden sind.

```
/* DRIVE-Programm */
.
.
.
DCL REPORT lager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT ... ;          /* Seitenstruktur festlegen */
   REPORT HEADER  ... ;         /* Listenkopf */
   .
   .
   REPORT TRAILER                                  ⎫
      PRINT NEWLINE 1,                             ⎪
      TAB 1, 'Gesamtsumme Teile: ' ,               ⎬ Listenfuß
             SUM (&report_var.anzahl);             ⎭
   .
   .
   .
```

Für den Listenfuß wurde mit der PRINT-Anweisung die Report-Mengenfunktion SUM angefordert und auf die Report-Variable &report_var.anzahl angewendet.

Ausgabe:

```
Gesamtsumme Teile: 4711
```

7.4.5 Seitenkopf

Einen Seitenkopf definieren Sie mit der Anweisung PAGE HEADER (siehe „DRIVE-Lexikon“ [3], Anweisung PAGE). Dort können Sie sich z.B. eine Überschrift oder eine Seitennumerierung ausgeben lassen. Voraussetzung dafür ist, daß Sie mit der Anweisung GLOBAL LAYOUT auch einen Kopfbereich festlegen. Ein Seitenkopf darf enthalten:

– Literale und mit der Anweisung SOURCE eingefügte Texte
– Systemvariablen
– Startparameter
– arithmetische Ausdrücke
– Report-Mengenwerte
– Daten des aktuellen Nettodatensatzes
– lokale Variablen

(Siehe Abschnitt Report-Daten, Seite 185).

Ein Seitenkopf wird auf jeder Seite im definierten Kopfbereich ausgegeben. Ausnahmen davon gelten, wenn:

– der Listenkopf allein auf der ersten Ausgabeseite erscheint
– der Listenkopf mehr Platz beansprucht, als für den Kopfbereich definiert ist
– innerhalb eines Listenfußes die Seite wechselt

In diesen drei Fällen wird der Seitenkopf unterdrückt. Ein Seitenkopf kann ein- oder mehrzeilig sein. Er kann maximal jedoch nur die Anzahl der Zeilen beanspruchen, die Sie für den Kopfbereich vorgesehen haben.

Der Report-Generator erkennt den Anfang einer Seite, wenn:

– die nächste freie Zeile mit der ersten Zeile des Fußbereichs identisch ist
– die Anzahl der möglichen Zeilen innerhalb der Seitenstruktur überschritten ist

Beispiel

Die Seitenköpfe des Reports sollen als Überschrift die Bezeichnungen enthalten, die für die aufzulistenden Teile ausgegeben werden.

```
/ * DRIVE-Programm */
.
.
.
DCL REPORT lager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT ... ;         /* Seitenstruktur festlegen */
   REPORT HEADER ... ;         /* Listenkopf */
   REPORT TRAILER ... ;        /* Listenfuß  */
   .
   .
   PAGE HEADER                                          )
     PRINT NEWLINE 2,                                   )
           TAB 1,  'Lageradresse',                      )
           TAB 17, ' Teilename ',                       )
           TAB 29, '| Anzahl ',                         )
           TAB 38, '| Teile-Nr. ',                      ) Seitenkopf
           TAB 50, '| Standardcode ',                   )
           NEWLINE 1,                                   )
           TAB 1,  '--------------------',              )
                   '--------------------',              )
                   '-------------------------';         )
   .
   .
   .
```

Ausgabe:

```
Schiller Ersatzteillager
Santerweg 12
1030 Wien
Tel. 0222/2134565


Lagerbestand vom 13.12.95 um 07:48:26


Lageradresse    | Teilename | Anzahl | Teile-Nr. | Standardcode
-------------------------------------------------------------
```

7.4.6 Seitenfuß

Einen Seitenfuß definieren Sie mit der Anweisung PAGE TRAILER (siehe „DRIVE-Lexikon“ [3], Anweisung PAGE). Dort können Sie sich z.B. eine Seitennumerierung oder eine Zwischensumme ausgeben lassen. Voraussetzung dafür ist, daß Sie mit der Anweisung GLOBAL LAYOUT auch einen Fußbereich festgelegt haben. Ein Seitenfuß darf enthalten:

– Literale und mit der Anweisung SOURCE eingefügte Texte
– Systemvariablen
– Startparameter
– arithmetische Ausdrücke
– Report-Mengenwerte
– Daten des aktuellen Datensatzes
– lokale Variablen

Ein Seitenfuß wird auf jeder Seite im definierten Fußbereich ausgegeben. Eine Ausnahme davon gilt, wenn ein Listenfuß den Report abschließt. Dann wird auf der letzten Report-Seite kein Seitenfuß ausgegeben. Ein Seitenfuß kann ein- oder mehrzeilig sein. Er kann maximal jedoch nur die Anzahl der Zeilen beanspruchen, die Sie für den Fußbereich vorgesehen haben.

Der Report-Generator erkennt das Ende einer Seite, wenn

– die nächste freie Zeile mit der ersten Zeile des Kopfbereichs identisch ist
– die Anzahl der möglichen Zeilen innerhalb der Seitenstruktur erreicht ist

Beispiel

Die Seitenfüße des Reports sollen den Hinweis „! Nachbestellung dringend notwendig“ und die Seitennummer enthalten.

```
/* DRIVE-Programm */
.
.
.
DCL REPORTlager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT ... ;          /* Seitenstruktur festlegen */
   REPORT HEADER ... ;          /* Listenkopf */
   REPORT TRAILER ... ;         /* Listenfuß  */
   PAGE  HEADER ...;            /* Seitenkopf */
   ...
```

```
PAGE TRAILER
      PRINT NEWLINE 1,                                  }
            TAB 1,  '! Nachbestellung ',                }
                    'dringend notwendig',               }  Seitenfuß
            NEWLINE 1,                                  }
            TAB 54, 'Seite - ', &PAGES;                 }
        .
        .
        .
```

Für die Ausgabe der Seitenzahl wurde in der PRINT-Anweisung die Systemvariable &PAGES verwendet.

Ausgabe:

```
! Nachbestellung dringend notwendig                Seite – 1
```

7.4.7 Gruppen

Im Detailbereich sind die Zuordnungen von Daten zu Datengruppen und die Anweisungen zusammengefaßt, mit denen Sie die Bearbeitung eines Datensatzes steuern können.

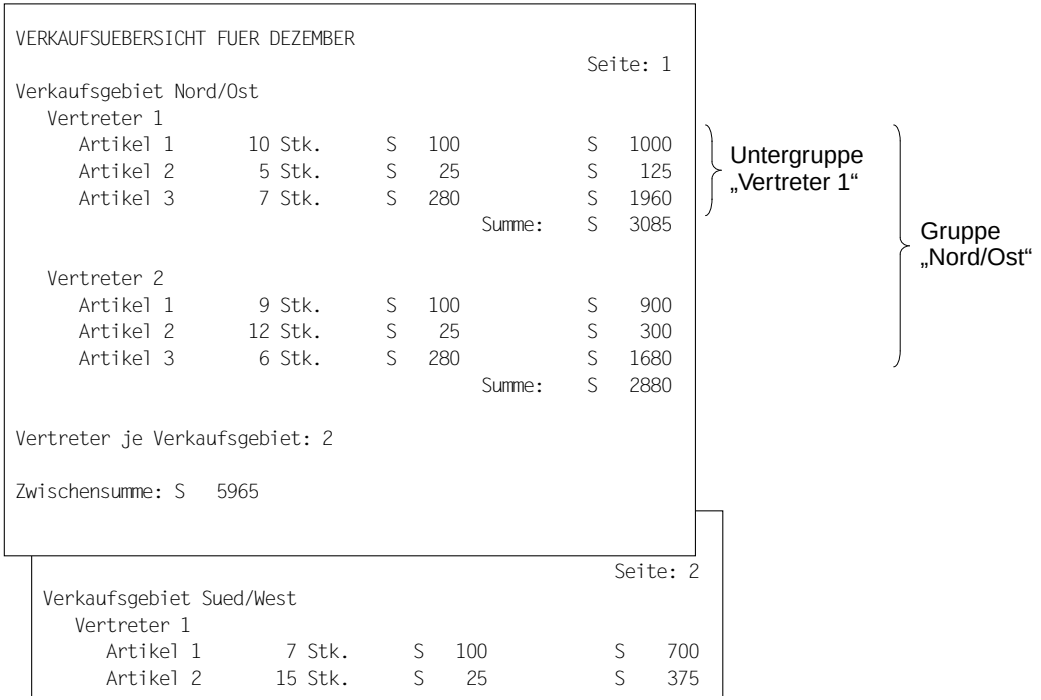
Liegen in mehreren Sätzen an jeweils gleicher Stelle in einem Datenfeld gleiche Inhalte vor, können diese Felder zu Gruppen zusammengefaßt werden. In der folgenden Tabelle stehen in der ersten Spalte die Gruppen bezüglich des Felds „Verkaufsgebiet“, in der zweiten Spalte die Gruppen bezüglich des Felds „Vertreter“.

Verkaufsgebiet	**Vertreter**					← Feldname
Nord/Ost	Vertreter 1	1	10	100	1000	← Feldinhalte
Nord/Ost	Vertreter 1	2	5	25	125	
Nord/Ost	Vertreter 1	3	7	280	1960	
Nord/Ost	Vertreter 2	1	9	100	900	← Gruppenwechsel im Feld „Vertreter“
Nord/Ost	Vertreter 2	2	12	25	300	
Nord/Ost	Vertreter 2	3	6	280	1680	
Sued/West	Vertreter 1	1	7	100	700	← Gruppenwechsel in den Feldern „Verkaufsgebiet“ und „Vertreter“
Sued/West	Vertreter 1	2	15	25	375	
Sued/West	Vertreter 1	3	9	280	2520	
Sued/West	Vertreter 2	1	5	100	500	← Gruppenwechsel im Feld „Vertreter“
Sued/West	Vertreter 2	2	18	25	450	
Sued/West	Vertreter 2	3	5	280	1400	

Wechselt der Inhalt eines Feldes, so wird ein Gruppenwechsel ausgelöst. Bei einem Gruppenwechsel können z.B. Zwischenergebnisse oder graphische Markierungen ausgegeben werden. Ein Gruppenwechselfeld wird durch den Namen des entsprechenden Feldes im Nettodatensatz bestimmt.

Enthält jeder der Datensätze mehrere Felder, die als Gruppenwechselfelder bestimmt werden können, lassen sich Hierarchien bilden.

Im unten dargestellten Report sind die Datenfelder mit dem Namen „Verkaufsgebiet“ entsprechend ihrem Inhalt zu den Gruppen „Nord/Ost“ und „Sued/West“ zusammengefaßt. Die Datenfelder mit dem Namen „Vertreter“ sind entsprechend ihren Inhalten zu den Gruppen „Vertreter 1“ und „Vertreter 2“ zusammengefaßt.



Die Gruppen im Feld „Vertreter“ werden in Abhängigkeit von den Gruppen im Feld „Verkaufsgebiet“ ausgegeben und bilden somit Untergruppen.

Sie können auch mehrere Felder (Feldkombination) zu einer Gruppe zusammenfassen. Beispielsweise sind im unten abgebildeten Report die Felder mit den Namen Verkaufsgebiet und Vertreter zu einer Gruppe zusammengefaßt. Dann wird ein Gruppenwechsel ausgelöst, wenn sich der Inhalt eines der Nettodatenfelder der Feldkombination ändert.

Gruppe						
Verkaufsgebiet	**Vertreter**					← Feldname
Nord/Ost	Vertreter 1	1	10	100	1000	← Feldinhalte
Nord/Ost	Vertreter 1	2	5	25	125	
Nord/Ost	Vertreter 1	3	7	280	1960	
Nord/Ost	Vertreter 2	1	9	100	900	← Gruppenwechsel durch Änderung des Inhalts im Feld „Vertreter“
Nord/Ost	Vertreter 2	2	12	25	300	
Nord/Ost	Vertreter 2	3	6	280	1680	

Die Daten werden entweder bereits sortiert vom Report-Generator übernommen oder sie werden vor der Ausgabeaufbereitung durch den Report-Generator sortiert. Diese implizite Sortierung ist immer möglich. Ein explizites Sortieren können Sie in der Anweisung GLOBAL LAYOUT veranlassen.

Sie definieren eine Gruppe, indem Sie die entsprechenden Daten je einem Kontrollblock für den Gruppenkopf und für den Gruppenfuß zuordnen. Wenn Sie mehrere Gruppen definieren, gibt die Reihenfolge der Definitionen eine Hierarchie an. Unter der Voraussetzung, daß die Felder der Gruppen sortiert sind, werden Gruppenhierarchien in der folgenden Art ausgeführt:

```
Gruppenkopf von a
   Gruppenkopf von b
      Gruppenkopf von c
         Detailzeilen
      Gruppenfuß von c
   Gruppenfuß von b
Gruppenfuß von a
```

Sind mehrere Gruppen definiert und findet im nächsten zu bearbeitenden Nettodatensatz ein Gruppenwechsel satt, dann werden die Kontrollblöcke in der Reihenfolge bearbeitet:

– alle Gruppenfüße von der niedrigsten bis zu der Kontrollhierarchie, in welcher der Gruppenwechsel stattfindet

– alle Gruppenköpfe von der aktuellen bis zur niedrigsten Kontrollhierarchie

– alle Anweisungen für die Detailzeilen, mit den Daten des nächsten Nettodatensatzes

7.4.8 Gruppenkopf

Einen Gruppenkopf für eine Gruppe definieren Sie mit der Anweisung GROUP HEADER (siehe „DRIVE-Lexikon“ [3], Anweisung GROUP). Dabei können Sie z.B. eine Überschrift festlegen, die bei jedem Gruppenwechsel innerhalb dieser Gruppe ausgegeben wird (siehe Abbildung auf Seite 216, Vertreter 1 bzw. Vertreter 2). Ein Gruppenkopf kann ein- oder mehrzeilig sein. In einem Gruppenkopf können Sie sich ausgeben lassen:

- Literale und mit der Anweisung SOURCE eingefügte Texte
- Systemvariablen
- Startparameter
- Report-Mengenwerte
- arithmetische Ausdrücke
- Daten des aktuellen Nettodatensatzes
- lokale Variablen

Beispiel

Es sollen alle Teile einer Lageradresse zusammen ausgegeben werden. Dazu wird die Gruppe Lageradresse definiert. Die Eingabedaten wurden bereits in der Anweisung GLOBAL LAYOUT nach Lageradressen sortiert.

```
/ * DRIVE-Programm */
.
.
.
DCL REPORT lager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT ... ;          /* Seitenstruktur festlegen */
   REPORT HEADER ... ;          /* Listenkopf */
   REPORT TRAILER ... ;         /* Listenfuß  */
   PAGE  HEADER ... ;           /* Seitenkopf */
   PAGE TRAILER ... ;           /* Seitenfuß  */
   .
   .
   GROUP  HEADER &lagername          )
      PRINT NEWLINE 1,               |
            TAB 1,  &lagername,      |
            TAB 17, '| ',           } Gruppenkopf
            TAB 29, '| ',           |
            TAB 38, '| ',           |
            TAB 50, '| ';           )
   .
   .
   .
```

Ausgabe:

```
Schiller Ersatzteillager
Santerweg 12
1030 Wien
Tel. 0222/2134565


Lagerbestand vom 13.12.95 um 07:48:26


Lageradresse    | Teilename | Anzahl | Teile-Nr. | Standardcode
_____________________________________________________________
Abtstr.1        |           |        |           |
```

7.4.9 Detailzeilen

Den Beginn von Detailzeilen kennzeichen Sie mit der Anweisung DETAIL (siehe „DRIVE-Lexikon“ [3], Anweisung DETAIL). Daran schließen sich Anweisungen an, die das Bearbeiten der Report-Daten steuern. Detailzeilen dürfen enthalten:

- Literale und mit der Anweisung SOURCE eingefügte Texte
- Systemvariablen
- Startparameter
- Report-Mengenwerte
- Nettodaten des aktuellen Datensatzes und anderer Satzarten
- arithmetische Ausdrücke
- lokale Variable

Bei Reports in Formularform definieren Sie normalerweise nur den Kontrollblock für Detailzeilen (siehe Abschnitt „Formulare“ auf Seite 204).

Beispiel

Als Detailzeilen sollen alle Teile aufgelistet werden, die sich in einem Lager befinden. Sind von einem Teil weniger als fünf Stück vorhanden, soll dies durch ein Ausrufezeichen gekennzeichnet werden.

```
/* DRIVE-Programm */
.
.
.
DCL REPORT lager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT ... ;           /* Seitenstruktur festlegen   */
   REPORT HEADER ... ;           /* Listenkopf  */
   REPORT TRAILER ... ;          /* Listenfuß   */
   PAGE  HEADER ... ;            /* Seitenkopf  */
   PAGE TRAILER ... ;            /* Seitenfuß   */
   GROUP HEADER &lagername ... ; /* Gruppenkopf */
   .
   .
     DETAIL                                                    )
        PRINT TAB 17, '| ',  &report_var.teilename ,           |
              TAB 29, '| ',  &report_var.anzahl MASK 'Z9',     |
              TAB 38, '| ',  &report_var.teilenummer,          |
              TAB 50, '| ',  &report_var.standardcode;         |  Detailzeilen
                                                              |
        IF &report_var.anzahl < 5                             |
           THEN PRINT TAB 65, '!';                            |
         END IF;                                              |
                                                              |
        PRINT NEWLINE 1;                                      )
   .
   .
   .
```

Mit der IF-Anweisung wird geprüft, ob die Anzahl der vorhanden Teile weniger als fünf ist. Ist das der Fall, wird ein Ausrufezeichen ausgegeben.

Ausgabe:

```
Schiller Ersatzteillager
Santerweg 12
1030 Wien
Tel. 0222/2134565


Lagerbestand vom 13.12.95 um 07:48:26


Lageradresse    | Teilename | Anzahl | Teile-Nr. | Standardcode
________________________________________________________________
Abtstr.1        |           |        |           |
                | Gewinde 2 |     78 | 7811      | 89-12-34
                | Gewinde 3 |      2 | 7812      | 89-12-34     !
                | Kette 4   |      2 | 7163      | 89-12-34     !
                | Kette     |      2 | 7162      | 89-12-34     !
                | Reifen    |     99 | 7112      | 89-12-34
                | Sitz      |      7 | 8122      | 89-12-34
```

7.4.10 Gruppenfuß

Einen Gruppenfuß für eine Gruppe definieren Sie mit der Anweisung GROUP TRAILER. Im Gruppenfuß können Sie sich z.B die Summe der Einzelwerte dieser Gruppe ausgeben lassen (siehe Abbildung auf Seite 216, Summe: S 3085 bzw. 2880). Ein Gruppenfuß kann enthalten:

- Literale und mit der Anweisung SOURCE eingefügte Texte
- Systemvariablen
- Startparameter
- Report-Mengenwerte
- arithmetische Ausdrücke
- Daten des aktuellen Nettodatensatzes
- lokale Variablen

Beispiel

Bei einem Gruppenwechsel soll die Summe aller Teile ausgegeben werden, die in diesem Lager vorhanden sind.

```
/* DRIVE-Programm */
.
.
.
DCL REPORT lager USING &report_var LIKE &drive_var;

   GLOBAL LAYOUT ... ;            /* Seitenstruktur festlegen */
   REPORT HEADER ... ;            /* Listenkopf   */
   REPORT TRAILER ... ;           /* Listenfuß    */
   PAGE  HEADER ... ;             /* Seitenkopf   */
   PAGE TRAILER ... ;             /* Seitenfuß    */
   GROUP HEADER &lagername ... ;  /* Gruppenkopf  */
      DETAIL ... ;                /* Dateilzeilen */
      .
      .
   GROUP TRAILER &lagername
      PRINT TAB 17, '-' (47),
      NEWLINE 1,
      TAB 17, '|  Summe:       ',
      SUM (&report_var.anzahl),
      NEWLINE 1,
      TAB 17, '-' (47),
      NEWLINE 1;
    .
    .
    .
```

Gruppenfuß

In diesem Beispiel wird in der Anweisung PRINT der Wiederholungsfaktor (47) für die Ausgabe des Querstrichs benutzt.

Ausgabe:

```
Schiller Ersatzteillager
Santerweg 12
1030 Wien
Tel. 0222/2134565


Lagerbestand vom 13.12.95 um 07:48:26


Lageradresse    | Teilename | Anzahl | Teile-Nr. | Standardcode
________________|___________|________|___________|_____________
Abtstr.1        |           |        |           |
                | Gewinde 2 |     78 | 7811      | 89-12-34
                | Gewinde 3 |      2 | 7812      | 89-12-34     !
                | Kette 4   |      2 | 7163      | 89-12-34     !
                | Kette     |      2 | 7162      | 89-12-34     !
                | Reifen    |     99 | 7112      | 89-12-34
                | Sitz      |      7 | 8122      | 89-12-34
                |___________|________|___________|_____________
                | Summe:       190
                |___________|________|___________|_____________
Alserstr. 34    |           |        |           |
                | Bohrer    |        |           |
...
```

7.4.11 Seitenhintergrundmuster

Eine Seite versehen Sie mit einem Hintergrundmuster, indem Sie zunächst das Hintergrundmuster mit der Anweisung GLOBAL PAGE BASE definieren. In der Anweisung GLOBAL PAGE BASE beschreiben Sie auch mit der Anweisung PAGE PRINT das Hintergrundmuster, das aus Texten, Linien und Rechtecken (mit Füllmustern) zusammengesetzt werden kann.

Mit der Anweisung OVERLAY PAGE BASE aktivieren Sie dann das zuvor definierte Hintergrundmuster.

Beispiel

Für einen Ausweis wird mit der Anweisung GLOBAL PAGE BASE ein Hintergrundmuster mit Text, Linien und Rechtecken definiert. In der aktuellen Systemumgebung ist „TYPE6“ ein hellgraues Füllmuster.
Die DETAIL-Anweisung aktiviert das Hintergrundmuster und bestimmt Position und Darstellungattribute der Ausgabefelder. In diesem Beispiel werden die Felder kursiv ausgegeben.
Die Anweisung OPEN REPORT ... RESULT FILE erzeugt eine Postscript-Datei „ausweis.ps“, die auf einem entsprechenden Drucker ausgedruckt werden kann.

```
/* DRIVE-Programm */
.
.
.
DECLARE REPORT ausweis USING ...

GLOBAL PAGE BASE h_ausweis
PAGE PRINT
   SET (FONT 'HELVETICA', PROPORTIONAL),
   SET (SIZE 60, BOLD 1),
      'projekte + vorhaben gmbh' POSITION 1.0 1.0 CM,
   LINE 1.0 1.2 8.9 1.2 CM LWIDTH 4 UNITS,
   BOX  1.0 2.0 5.5 2.0 CM BTYPE 'TYPE6' BWIDTH 0.75
        LINE LWIDTH 1 UNITS,
   BOX  5.8 2.0 8.9 2.0 CM BTYPE 'TYPE6' BWIDTH 0.75
        LINE LWIDTH 1 UNITS,
   SET (SIZE 30, BOLD 0),
   'Name' POSITION 1.0 3.1 CM,
   'Personalnummer' POSITION 5.8 3.1 CM,
   SET (SIZE 40),
   'Projekt:' POSITION 2.5 5.0 CM,
   BOX  0.6 0.2 9.4 0.2 CM BTYPE EMPTY BWIDTH 5.4
        LINE LWIDTH 2 UNITS;

DETAIL
   OVERLAY PAGE BASE h_ausweis ON;
   PRINT
   SET (SIZE 50, ITALIC 1, BOLD 1),
   PAGE POSITION 1.2 2.6 CM,
      &nachname,
   PAGE POSITION 6.0 2.6 CM,
      &nachname, &abt_mit_nr CLIPPED, ' ', &lfd_nr,
   SET (SIZE 70),
   PAGE POSITION 4.3 5.0 CM,
      &p_name,
```

```
   SET (BOLD 0,ITALIC 0);
END REPORT;

OPEN REPORT ausweis RESULT FILE 'ausweis.ps' DEVICETABLE 'PS';
FILL REPORT ausweis USING ...
CLOSE REPORT ausweis;
```

Ausgabe:

Aus drucktechnischen Gründen entspricht diese Darstellung nicht völlig der Druckausgabe.

7.5 Individuellen Report generieren

In diesem Abschnitt wird aus den folgenden Daten schrittweise ein Report generiert. In diesem Beispiel erfahren Sie, wie Sie

- Daten für eine Report-Definition beschreiben
- lokale Daten definieren
- Daten sortieren und Bereiche bestimmen
- Inhalte der Kontrollblöcke festlegen
- den Report mit Daten versorgen

Für das Beispiel in diesem Abschnitt werden folgende Daten verwendet:

Nachname	Ort	Gehalt	Projekt	P_Name	Budget
Mitscher	HANNOVER	0490000	000000109	Athene	000050000000
Manson	MUENCHEN	1200000	000000111	Prometheus	000002000000
Winterberg	HAMBURG	0350000	000000108	Poseidon	000061000000
Mitscherlich	HAMBURG	0270000	000000108	Poseidon	000061000000
Lorenz	MUENCHEN	0330000	000000110	Aphrodite	000008000000
Grenzer	HEIST	0230000	000000108	Poseidon	000061000000
van Claeren	BERN	0590000	000000111	Prometheus	000002000000
Paarungen	HAMBURG	0320000	000000108	Poseidon	000061000000
Miller	LOS ALAMOS	0960000	000000111	Prometheus	000002000000
Jansen	HANNOVER-MUENDEN	0470000	000000109	Athene	000050000000
Ohlsen	KIEL	0400000	000000107	Hera	000015000000
Mattsen	KIEL	0480000	000000107	Hera	000015000000
Nonnen	KIEL	0450000	000000107	Hera	000015000000
Andermatt	HAMBURG	0510000	000000108	Poseidon	000061000000
Zimmermann	HANNOVER	0580000	000000109	Athene	000050000000
Sennelaub	HANNOVER	0650000	000000109	Athene	000050000000
Sennert	MUENCHEN	0580000	000000106	Zeus	000035000000
Schmidt	HAMBURG	0620000	000000108	Poseidon	000061000000
Ammer	MUENCHEN	0570000	000000110	Aphrodite	000008000000
Dormagen	HELMSTORF	0550000	000000107	Hera	000015000000
Dollinger	MUENCHEN	0280000	000000110	Aphrodite	000008000000
von Haalen	NEW YORK	1000000	000000111	Prometheus	000002000000
Pollinger	HAMBURG	0400000	000000108	Poseidon	000061000000
Bergmann	MUENCHEN	0690000	000000106	Zeus	000035000000
Berghoff	MUENCHEN	0300000	000000106	Zeus	000035000000
Bergner	MUENCHEN	0450000	000000106	Zeus	000035000000
Maier	HANNOVER	0350000	000000109	Athene	000050000000
Maier	HANNOVER	0290000	000000109	Athene	000050000000
Mauer	HAMBURG	0650000	000000108	Poseidon	000061000000

In dem Report soll sortiert aufgelistet werden, an welchem Projekt, in welchen Städten, welche Mitarbeiter beschäftigt sind. Der Report soll aussehen, wie in der folgenden Abbildung.

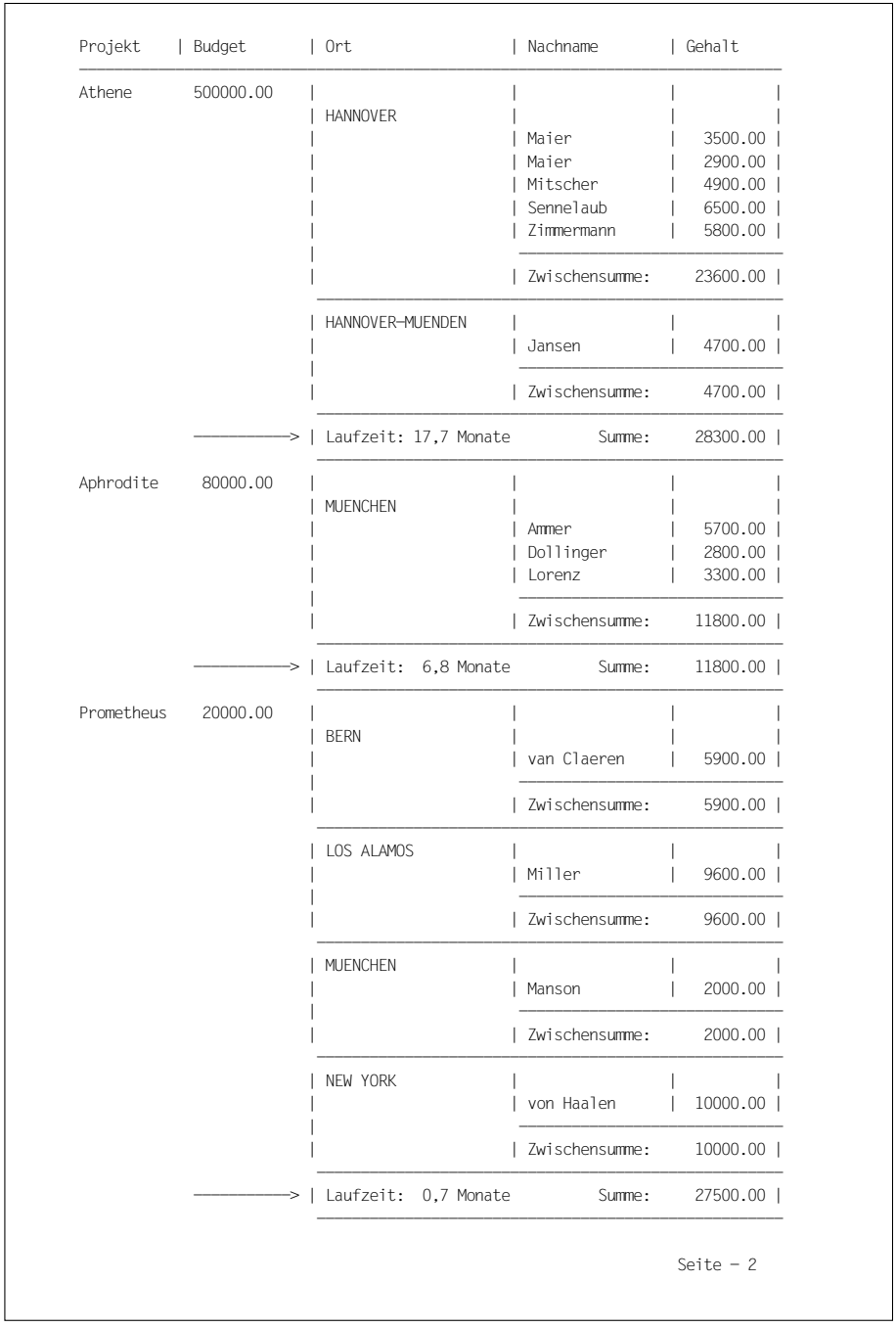
Beispiel

Gehaltaufkommen für laufende Projekte: Übersicht

Stand 23.01.96

Projekt	Budget	Ort	Nachname	Gehalt
Zeus	350000.00			
		MUENCHEN		
			Berghoff	3000.00
			Bergmann	6900.00
			Bergner	4500.00
			Sennert	5800.00
			Zwischensumme:	20200.00
	——————>	Laufzeit: 17,3 Monate	Summe:	20200.00
Hera	150000.00			
		HELMSTORF		
			Dormagen	5500.00
			Zwischensumme:	5500.00
		KIEL		
			Mattsen	4800.00
			Nonnen	4500.00
			Ohlsen	4000.00
			Zwischensumme:	13300.00
	——————>	Laufzeit: 8,0 Monate	Summe:	18800.00
Poseidon	610000.00			
		HAMBURG		
			Andermatt	5100.00
			Mauer	6500.00
			Mitscherlich	2700.00
			Paarungen	3200.00
			Pollinger	4000.00
			Schmidt	6200.00
			Winterberg	3500.00
			Zwischensumme:	31500.00
		HEIST		
			Grenzer	2300.00
			Zwischensumme:	2300.00
	——————>	Laufzeit: 18,2 Monate	Summe:	33500.00

Seite – 1



```
Projekt    | Budget       | Ort                  | Nachname        | Gehalt
-----------------------------------------------------------------------------
Athene       500000.00  |                      |                 |
                        | HANNOVER             |                 |
                        |                      | Maier           |   3500.00 |
                        |                      | Maier           |   2900.00 |
                        |                      | Mitscher        |   4900.00 |
                        |                      | Sennelaub       |   6500.00 |
                        |                      | Zimmermann      |   5800.00 |
                        |                      |                 ------------
                        |                      | Zwischensumme:  |  23600.00 |
                        -------------------------------------------------------
                        | HANNOVER-MUENDEN     |                 |
                        |                      | Jansen          |   4700.00 |
                        |                      |                 ------------
                        |                      | Zwischensumme:  |   4700.00 |
                        -------------------------------------------------------
              ----------> | Laufzeit: 17,7 Monate       Summe:     |  28300.00 |
                        -------------------------------------------------------
Aphrodite     80000.00  |                      |                 |
                        | MUENCHEN             |                 |
                        |                      | Ammer           |   5700.00 |
                        |                      | Dollinger       |   2800.00 |
                        |                      | Lorenz          |   3300.00 |
                        |                      |                 ------------
                        |                      | Zwischensumme:  |  11800.00 |
                        -------------------------------------------------------
              ----------> | Laufzeit:  6,8 Monate       Summe:     |  11800.00 |
                        -------------------------------------------------------
Prometheus    20000.00  |                      |                 |
                        | BERN                 |                 |
                        |                      | van Claeren     |   5900.00 |
                        |                      |                 ------------
                        |                      | Zwischensumme:  |   5900.00 |
                        -------------------------------------------------------
                        | LOS ALAMOS           |                 |
                        |                      | Miller          |   9600.00 |
                        |                      |                 ------------
                        |                      | Zwischensumme:  |   9600.00 |
                        -------------------------------------------------------
                        | MUENCHEN             |                 |
                        |                      | Manson          |   2000.00 |
                        |                      |                 ------------
                        |                      | Zwischensumme:  |   2000.00 |
                        -------------------------------------------------------
                        | NEW YORK             |                 |
                        |                      | von Haalen      |  10000.00 |
                        |                      |                 ------------
                        |                      | Zwischensumme:  |  10000.00 |
                        -------------------------------------------------------
              ----------> | Laufzeit:  0,7 Monate       Summe:     |  27500.00 |
                        -------------------------------------------------------

                                                                Seite - 2
```

7.5.1 Daten für die Report-Definition beschreiben

Zunächst müssen Sie die Daten, die bearbeitet werden sollen, für die Report-Definition beschreiben. Im DRIVE-Programm sind diese Daten wie folgt deklariert:

```
DCL VAR 1  &mitarb,
           2 nachname  CHAR (12),
           2 ort       CHAR (16),
           2 gehalt    NUM  (8,2),
           2 projekt   INTEGER,
           2 p_name    CHAR (10),
           2 budget    NUM  (12,2);
```

Zur Beschreibung der Daten verwenden Sie die Anweisung DECLARE REPORT.

```
.
.
.
DCL REPORT ueber USING &daten LIKE &mitarb; ------------------------------------------------   (1)
.
.
.
```

(1) Der Report erhält den Namen ueber. Die Daten &mitarb werden in der Variablen &daten beschrieben.

7.5.2 Lokale Daten definieren

Für die Berechnung der Gehaltssummen und der Projektlaufzeit definieren Sie mit der Anweisung DECLARE VARIABLE lokale Report-Variablen.

```
.
.
.
   DCL VAR &sum_ko   NUM(8,2); ---------------------------------------------------------------   (1)
   DCL VAR &lau_zt   NUM(5,1);
.
.
.
```

(1) Die Variable &sum_ko nimmt die Gehaltssummen auf und &lau_zt die Laufzeitwerte. Beide Variablen müssen von einfachem Typ sein.

7.5.3 Daten sortieren und Bereiche festlegen

Die Daten sollen im Report nach Projektnummern, nach Städten und innerhalb der Städte nach Namen sortiert verarbeitet werden.

Anschließend definieren Sie die Breiten der Seitenränder und den Kopf- und Fußbereich einer Seite. Wieviele Zeilen und Spalten standardmäßig auf eine Ausgabeseite passen, ist im Druckerprofil des Druckers hinterlegt, den Sie verwenden wollen (siehe Abschnitt „Geräteprofil“ auf Seite 237).

Denken Sie daran, daß der Kopfbereich der ersten Seite und der Fußbereich der letzten Seite ggf. nicht ausgegeben wird, wenn Listenkopf bzw. -fuß diesen Bereich beanspruchen.

```
.
.
.
   GLOBAL LAYOUT
     ORDER BY &daten.projekt ASCENDING,-------------------------------------------   (1)
              &daten.ort ASCENDING,
              &daten.nachname ASCENDING
      TOP MARGIN 5 ----------------------------------------------------------------   (2)
      BOTTOM MARGIN 5
      LEFT MARGIN 10
      RIGHT MARGIN
      HEADER LINES 6---------------------------------------------------------------   (3)
      TRAILER LINES 6
      MINIMUM LINES 48;------------------------------------------------------------   (4)
.
.
.
```

(1) Sie weisen das Sortieren der Daten &daten mit den Schlüsselworten ORDER BY an. Das Sortieren in aufsteigender Reihenfolge kennzeichnen Sie mit ASCENDING.

(2) Für die Seitenränder sehen Sie für den oberen und unteren Rand je 5 Zeilen vor. Der linke Rand soll 10 Spalten und der rechte Rand 5 Spalten breit sein.

(3) Die Breiten von Kopf- und Fußbereich legen Sie mit je 6 Zeilen fest.

(4) Auf einer Ausgabeseite sollen mindestens 48 Zeilen zur Verfügung stehen.

7.5.4 Inhalt des Listenkopfs festlegen

Der Listenkopf soll darüber informieren, was die gesamte Liste zu welchem Zeitpunkt enthält.

```
.
.
.
 REPORT HEADER
     PRINT SET (BOLD 1, FONT TIMES ROMAN, CHARACTER DENSITY 12),————————————————   (1)
               'Gehaltsaufkommen für laufende Projekte: Übersicht',
           RESET (BOLD),————————————————————————————————————————————————————   (2)
           NEWLINE 2,———————————————————————————————————————————————————————   (3)
           SET (ITALIC 1),——————————————————————————————————————————————————   (4)
               'Stand ', CURRENT DATE MASK 'DD''.''MO''.''YY',——————————————————   (5)
           RESET (ITALIC);——————————————————————————————————————————————————   (6)
.
.
.
```

(1) Für den Listenkopf setzen Sie mit der PRINT-Anweisung die folgenden Darstellungsattribute: Fettschrift, Schriftart Times Roman, Zeichendichte 12.

(2) Sie schalten die Fettschrift wieder aus.

(3) Der Listenüberschrift soll eine Leerzeile folgen. Da der Druckerkopf nach der letzten Ausgabe stehen bleibt, müssen Sie einen Zeilenvorschub von 2 Zeilen angeben.

(4) Sie setzen das Darstellungsattribut für kursive Schrift.

(5) Das aktuelle Datum erhalten Sie mit der Angabe CURRENT DATE und das Ausgabeformat mit den Maskensteuerzeichen 'DD''."MO''."YY'.

(6) Sie schalten die kursive Schrift wieder aus.

7.5.5 Inhalt von Seitenkopf und -fuß festlegen

Jede Seite soll mit den Spaltenüberschriften für die Auflistung beginnen. Der Seitenfuß soll eine Seitennumerierung enthalten.

```
.
.
.
 PAGE HEADER

     PRINT SET (CHARACTER DENSITY 10),                        (1)
           NEWLINE 2,
           TAB 1,  'Projekt',                                 (2)
           TAB 12, '| Budget',                                (2)
           TAB 27, '| Ort',                                   (2)
           TAB 50, '| Nachname',                              (2)
           TAB 68, '| Gehalt',                                (2)

           NEWLINE 1,
           TAB 1, '-' (80);                                   (3)

 PAGE TRAILER

     PRINT NEWLINE 2,
           TAB 69, 'Seite - ', &PAGES;                        (4)
.
.
.
```

(1) Sie setzen die Schreibdichte auf 10.

(2) Nach zwei Zeilenvorschüben geben Sie die Tabulatorpositionen an, an denen eine Spaltenüberschrift beginnt.

(3) Den Seitenkopf beenden Sie mit einer durchgezogenen Line unter den Spaltenüberschriften. Die durchgezogene Line können Sie mit einem Wiederholungsfaktor in der Form '-' (80) erzeugen.

(4) Im Seitenfuß verwenden Sie die Systemvariable &PAGES, um die Seitennummer zu erhalten.

7.5.6 Inhalt des Detailbereichs festlegen

Bevor Sie den Inhalt für Gruppenköpfe, Detailzeilen und Gruppenfüße festlegen, müssen Sie zunächst Gruppen definieren.

Sie haben die Gruppe Projekt, auf die sich alle anderen Daten beziehen. Innerhalb der Gruppe Projekt gibt es die Gruppe Städte, denn die Zwischensummen für die Gehälter beziehen sich auf die Städtenamen. In diesem Beispiel wird das mit zwei Gruppenköpfen bzw. -füße beschrieben.

Gruppenköpfe

Der Gruppenkopf der Gruppe Projekt soll jeweils den Namen des Projekts und das zur Verfügung stehende Budget enthalten.

```
.
.
.
 GROUP HEADER &daten.projekt GROUPNUM 1 ──────────────────────────────────────   (1)
     PRINT TAB 1,  &daten.p_name,
           TAB 14, &daten.budget MASK 'ZZZZZZZZZZP99',
           TAB 27, '|',
           TAB 50, '|',
           TAB 68, '|',
           TAB 79, '|';
   GROUP HEADER &daten.ort GROUPNUM 2 ────────────────────────────────────────   (2)
     PRINT TAB 27, '| ', &daten.ort,
           TAB 50, '|',
           TAB 68, '|',
           TAB 79, '|';
.
.
.
```

(1) In der Anweisung GROUP HEADER geben Sie für die Gruppe Projekt als Gruppenwechselfeld &daten.projekt für die höchste Hierarchiestufe an. Außerdem erhält diese Gruppe die Gruppennummer 1. Im Gruppenkopf geben Sie den Namen des Projekts und das noch zur Verfügung stehende Budget aus.

(2) Für die von der Gruppe Projekt abhängige Gruppe Städte definieren Sie als Gruppenwechselfeld &daten.ort. Diese Gruppe erhält die Gruppennummer 2. Im Gruppenkopf geben Sie den Städtenamen aus.

Detailzeilen

In den Detailzeilen werden die Namen der Mitarbeiter und deren Monatsgehalt ausgegeben. Die Monatsgehälter werden je Städtegruppe summiert.

```
...
 DETAIL

     PRINT TAB 27, '|',
           TAB 50, '| ', &daten.nachname,
           TAB 68, '| ', &daten.gehalt MASK 'ZZZZZZP99', ' |',
           NEWLINE 1;
...
```

Gruppenfüße

Wechselt bei der Bearbeitung der Gruppe Städte der Inhalt des Gruppenwechselfeldes &daten.ort, soll im Gruppenfuß die Summe der Gehälter ausgegeben werden, die in dieser Stadt für das Projekt pro Monat gezahlt werden.

Im Gruppenfuß der Gruppe Projekt soll die Summe aller Gehälter für das Projekt ausgegeben werden. Außerdem wird errechnet, welche Laufzeit das Projekt anhand des vorhandenen Budgets und der Summe der Gehälter noch haben kann.

```
.
.
 GROUP TRAILER &daten.ort GROUPNUM 2
     SET &sum_ko = SUM (GROUP GROUPNUM 2 &daten.gehalt); ————————————————————   (1)
     PRINT NEED LINES 10,————————————————————————————————————————————————   (2)
           TAB 27, '|',
           TAB 51, '-' (30),
           NEWLINE 1,
           TAB 27, '|',
           TAB 50, '| Zwischensumme:',
           TAB 71, &sum_ko MASK 'ZZZZZP99', ' |', ——————————————————————————   (3)
           NEWLINE 1,
           TAB 28, '-' (53);
GROUP TRAILER &daten.projekt  GROUPNUM 1
     SET &sum_ko = SUM (GROUP GROUPNUM 1 &daten.gehalt);————————————————————   (4)
     SET &lau_zt = &daten.budget/&sum_ko; ———————————————————————————————   (5)
     PRINT TAB 14, '———————————>',
           TAB 27, '| Laufzeit: ', &lau_zt MASK 'ZZP9', ' Monate',
           TAB 60, 'Summe:   ', &sum_ko MASK 'ZZZZZZP99', ' |',
           NEWLINE 1,
           TAB 28, '-' (53);
END REPORT;
.
.
```

(1) Sie weisen der Report-Variablen &sum_ko die Summe der Gehälter je Stadt zu.

(2) Mit der PRINT-Anweisung legen Sie fest, daß auf einer Seite für die Ausgabe dieser Detailzeilen mindestens 10 Zeilen zur Verfügung stehen müssen. Sind keine 10 Zeilen mehr vorhanden, erfolgt ein Seitenumbruch.

(3) Die errechnete Summe geben Sie in dem Format aus, das Sie mit dem Schlüsselwort MASK definieren.

(4) Den letzten Inhalt der Variablen &sum_ko können Sie bei einem Gruppenwechsel in der Gruppe Projekt überschreiben und jetzt die Gehaltskosten für das gesamte Projekt zuweisen.

(5) Aus dem zur Verfügung stehenden Budget und der Gesamtsumme der Gehälter errechnen Sie die noch verbleibende Laufzeit &lau_zt.

7.5.7 Report mit Daten versorgen

Nachdem der Report definiert ist, wird er gemäß der Definition mit Daten versorgt. Beim Öffnen des Report-Puffers bestimmen Sie auch den Drucker, auf dem der Report ausgegeben wird.

```
.
.
.
OPEN REPORT ueber RESULT LIST 'G203' DEVICETABLE '9001';

   CYCLE c1 INTO &mitarb.*;
      FILL REPORT ueber USING &mitarb;
   END CYCLE;

 CLOSE REPORT ueber;
```

7.5.8 Report ausgeben

Wenn der Report-Puffer gemäß der Report-Definition ueber mit Daten versorgt und geschlossen ist, beginnt die Ausgabe des Reports. Der Report wird auf dem Gerät ausgegeben, daß Sie in der Anweisung OPEN REPORT gewählt haben. In diesem Beispiel wurde mit den Schlüsselworten RESULT LIST eine Druckerausgabe angefordert und der Typ des Druckers mit „9001“ spezifiziert.

7.6 Systemumgebung

Um mit dem Report-Generator arbeiten zu können, müssen bestimmte report-spezifische

- Benutzerprofile vorliegen
- Geräteprofile vorhanden sein
- Dateien installiert sein

7.6.1 Benutzerprofil

Das Benutzerprofil ist eine editierbare Datei mit dem Namen PROFILE.USER. Diese Datei wird als Element vom Typ S in der DRIVE-Bibliothek oder in der PLAM-Bibliothek mit dem Linknamen RSOML abgelegt. Ist die Datei PROFILE.USER nicht vorhanden, verwendet DRIVE/WINDOWS den vorbelegten Wert.

Die Datei PROFILE.USER enthält folgende Vorbelegungen, die zum Teil benutzerspezifisch verändert werde können:

- Globale Formateinstellung für die Werte der Datentypen DATE und TIME:

  ```
  DATEFORMAT=YYYY''-''MM''-''DD

  TIMEFORMAT=hh'':''mm'':''ss
  ```

  Diese Angaben dürfen vom Anwender verändert werden.

- Wertevorbelegung und Länge des Datentyps BOOLEAN:

  ```
  BOOLSET=1,0x00
  ```

  Die Länge ist mit der Anzahl der Bytes angegeben und der symbolische Wert FALSE als Hexadezimalwert. Für TRUE wird implizit (!FALSE) angenommen.

  Diese Angabe darf vom Anwender **nicht** verändert werden.

- Voreinstellung für das Indikatorfeld für NULL-VALUES:

  ```
  NULLSET='B'        Darf vom Anwender nicht verändert werden.
  NULLPOS='0'                          "

  NULLNUM='-'        Darf vom Anwender verändert werden.
  NULLCHAR=' '                         "
  ```

– Voreinstellungen für Trennzeichen

  `DELSTRUC='.'` für Komponenten einer Struktur,

  `DELINDON='('` für die Indizierungen bei den Datentypen
  `DELINDOFF=')'` matrix und datengruppe,

  `DELRANGE=','` für Bereichsangaben beim Datentyp matrix,

  `DELRECORD=':'` für die Satzartkennung.

  Diese Angaben dürfen vom Anwender **nicht** verändert werden.

– Voreinstellung für die Darstellung des Dezimalpunkts:

  `DECPOINT='.'`

  Diese Angabe darf vom Anwender verändert werden.

– Voreinstellung für PRINT-DOCUMENT-Kommando:

  `SPOOLDOPT=LAYOUT-CONTR=PAR(CONTR-CHAR=PHYS)`

  Werden in einem Report mehrere Schriftarten verwendet, müssen diese Schriftarten für ND- und HP-Drucker außerdem in der Option CHAR-SET angegeben werden (siehe DRIVE-Lexikon [3], Report-Anweisung OPEN REPORT).

7.6.2 Geräteprofil

Alle gerätespezifischen Voreinstellungen sind in einem Geräteprofil zusammengefaßt. Für jeden unterstützten Drucker gibt es ein Geräteprofil in einer lesbaren Datei mit dem Namen PROFILE.geraete-tabelle. geraete-tabelle ist der Name des Geräts.

Die Datei PROFILE.geraete-tabelle ist als Element vom Typ S in der PLAM-Bibliothek mit dem Linknamen RSOML abgelegt. Diese Datei darf nur vom Systemverwalter verändert werden.

Unterstützte Drucker sind:

9001-1, 9001-11, 9001-2, 9001-3, 9001-31, 9001-32, 9001-891, 9001-892, 9001-893, 9001-8931, 9001-8939,
9011-18, 9011-19, 9011-28, 9011-29,
9012-2,
9013, 9013-300,
9021,

9047,
HPLJ (beliebige HPLaserJet-Drucker (ab PCL4))
PS (beliebige PostScript-Drucker)

Im Geräteprofil sind Anweisungszeilen hinterlegt, die auf die Metasteuerzeichen für das geräteunabhängige Format verweisen und die tatsächlichen Steuerzeichen für das jeweilige Gerät enthalten.

Wird ein Report ausgedruckt, setzt der Report-Generator die Metasteuerzeichen mit Hilfe eines Konverters in die tatsächlichen Steuerzeichen für den Drucker um.

Für jedes Metasteuerzeichen kann eine Anweisungszeile mit dem tatsächlichen Steuerzeichen definiert werden, wenn der Drucker dies unterstützt. Die tatsächlichen Steuerzeichen sind im Druckerhandbuch zu finden.

Definitionen für Metasteuerzeichen sind in /# und #/ eingeschlossen. Zwischen den Zeichen #: und :# stehen Informationen für den Interpreter. Kommentare sind mit /* und */ gekennzeichnet.

Anweisungen für die Steuerzeichen beginnen jeweils auf einer neuen Zeile. Kommentare können mehrere Zeilen lang sein. Sie dürfen auch in Anweisungszeilen stehen.

Im folgenden sind nur die Anweisungen aufgelistet, die Festlegungen für eine Druckseite betreffen. Diese Angaben ermöglichen dem Report-Generator, alle Aufteilungen und Berechnungen für eine Druckseite vorzunehmen.

Änderungen in den Geräteprofilen oder das Anlegen neuer Geräteprofile sollte nur vom Systemverwalter vorgenommen werden.

RDINAME: #:RN=rdiname:#

rdiname ist der Name des Konverters, der Metasteuerzeichen in tatsächliche Steuerzeichen umsetzt.

Seitenlänge: #:PL=seitenlänge1,seitenlänge2:#

seitenlänge1 ist die Anzahl der Druckzeilen im Hochformat. seitenlänge2 ist die Anzahl der Druckzeilen im Querformat.

Die Anzahl der möglichen Zeilen läßt sich aus dem Zeilenabstand errechnen (siehe unten).

Gibt es eines der beiden Formate nicht, muß die entsprechende Angabe 0 sein. Die Angabe #:PL=0,0:# ist nicht zulässig.

Seitenbreite: #:PW=druckspalten1,druckspalten2:#

druckspalten1 ist die Anzahl der Druckspalten je Zeile im Hochformat. druckspalten2 ist die Anzahl der Druckspalten je Zeile im Querformat. Die Anzahl der möglichen Druckspalten läßt sich aus der Spaltenbreite errechnen (siehe unten).

Gibt es eines der beiden Formate nicht, muß die entsprechende Angabe 0 sein. Die Angabe #:PW=0,0:# ist nicht zulässig.

Zeilenhöhe: #:LE=aktzeilenhöhe,minzeilenhöhe:#

aktzeilenhöhe bezeichnet die voreingestellte Zeilenhöhe einer Druckzeile des Druckers. minzeilenhöhe bezeichnet die kleinste Zeilenhöhe einer Druckzeile des Druckers. Angegeben werden die Zähler der Ergebnisse aus der Umrechnung in 1/7200 Zoll.

Beispiel

```
1/6 Zoll       ist der tatsächliche Wert
1200/7200 Zoll ist der umgerechnete Wert
1200           ist als Wert anzugeben
```

Spaltenbreite: #:CW=spaltenbreite,0:#

spaltenbreite bezeichnet die voreingestellte Spaltenbreite des Druckers. Angegeben wird der Zähler des Ergebnisses aus der Umrechnung in 1/7200 Zoll (siehe Beispiel oben). Der zweite Wert ist hier immer 0.

Breitschriftenfaktor: #:EH=faktor1[,faktor2]...:#

faktor1 bezeichnet den Vervielfachungsfaktor der Schriftbreite beim Aktivieren der Breitschrift. Angegeben wird der Zähler des Ergebnisses aus der Umrechnung in 1/7200 Zoll (siehe Beispiel oben).

Gibt es mehrere Breitschriften, sind deren umgerechnete Faktoren ebenfalls anzugeben. Es können maximal sechs Faktoren, durch Kommas getrennt, angegeben werden.

Beispiel

```
Bei Verdoppelung des Schreibschritts ist der Faktor 2.
Umgerechnet sind das 14400/7200 Zoll.
Angegeben wird 14400.
```

Kennt der Drucker keine Breitschrift, sollte diese Anweisungszeile entfallen.

Zeichenbreite: #:CI=werteliste:#

werteliste enthält die für den Drucker gültigen Werte für die Zeichenbreite in Zeichen pro Zoll (CPI). Handelt es sich um diskrete Werte, müssen sie durch Kommas getrennt angegeben werden (z.B. #:CI=10,12,14,16.67,18.8:#). Handelt es sich bei den zulässigen Werten um Intervalle, wird zuerst der Wert für die kleinstmögliche Zeichenbreite, dann die Zeichenkombination und zuletzt die größtmögliche Zeichenbreite angegeben (z.B. #:CI=8,..,18:#).

Fehlt diese Anweisungszeile, wird jede in der Report-Definition angegebene Zeichenbreite als gültig angesehen.

Zeichenabstand: #:CD=werteliste:#

werteliste enthält die für den Drucker gültigen Werte für die Zeichenabstände in 1/300 Zoll-Einheiten. Diskrete Werte oder Intervalle werden, wie bei der Zeichenbreite beschrieben, angegeben.

Fehlt diese Anweisungszeile, wird jeder in der Report-Definition angegebene Zeichenabstand als gültig angesehen.

Zeilenabstand: #:LD=werteliste:#

werteliste enthält die für den Drucker gültigen Werte für die Zeilenabstände in 1/300 Zoll-Einheiten. Diskrete Werte oder Intervalle werden, wie bei der Zeichenbreite beschrieben, angegeben.

Fehlt diese Anweisungszeile wird jeder in der Report-Definition angegebene Zeilenabstand als gültig angesehen.

7.6.3 Report-Dateien

Bei der Übersetzung von Report-Definitionen entstehen Report-Dateien. Diese Dateien werden in der DRIVE-Bibliothek als Elemente vom Typ X abgelegt.

Die Namen dieser Dateien werden aus den ersten fünf Zeichen des Programmnamens, dem Report-Namen und einer Namenserweiterung gebildet. Die Namenserweiterung ist

– .x während der Übersetzung,
– .r während der Ausführung und bei der Ablage von Zwischencode,

Die so gebildeten Dateinamen müssen den BS2000-Konventionen für Dateinamen entsprechen (siehe BS2000 Benutzerkommandos [35]). Für die Eindeutigkeit der Namen müssen Sie bei der Vergabe des Report-Namens sorgen.

8 Datenbanken bearbeiten

Eine Einführung in die Struktur einer relationalen Datenbank sowie in die Begriffe und Konzepte von SQL finden Sie in den datenbank-spezifischen SQL-Sprachbeschreibungen:

– SQL für SESAM/SQL V1.0 [17]

– SQL für SESAM/SQL V2 [18 und 20]

– SQL für UDS/SQL [25]

Dieses Kapitel weist stichwortartig auf die wichtigsten Bearbeitungsmöglichkeiten hin.

Eine Datenbank ist die 'physische' Datenbasis, die abgefragt und verändert werden kann. Sie umfaßt die Datensätze sowie die Beschreibung der Sätze und ihrer logischen Organisation (Metadaten). Aus relationaler Sicht ist jede Datenbank logisch in Basistabellen und Systemtabellen organisiert.

Basistabellen sind im relationalen Schema der Datenbank definiert, entsprechen also den UDS-Satzarten. Set-Beziehungen zwischen Satzarten und die Arten der Set-Mitgliedschaft werden auf Spaltenconstraints abgebildet.

Die Datensätze einer Datenbank können gelesen (Selektions-Operation) oder neu eingefügt, gelöscht oder in ihren Werten geändert werden (Manipulations-Operationen).

Beim **Selektieren** kann mit der SELECT-Anweisung ein einzelner Satz aus der Datenbank gelesen werden. Gibt es mehrere Treffer, gibt DRIVE/WINDOWS eine Fehlermeldung aus. Sollen mehrere Sätze gelesen werden, muß eine Ergebnistabelle (Cursortabelle) spezifiziert werden, die die ausgewählten Sätze enthält. Diese Ergebnistabelle wird durch das Deklarieren eines Cursor (Satzzeiger) mit der Metavariablen *cursorbeschreibung* erzeugt (siehe „DRIVE-SQL-Lexika“ [4], [5], [6], Metavariable *cursorbeschreibung*).

Beim **Manipulieren** können mit **einer** SQL-Anweisung mehrere Sätze, die bestimmte Bedingungen erfüllen in **gleicher** Weise geändert werden (**mengenorientierte Änderung**, siehe „DRIVE-SQL-Lexika“ [4], [5], [6], Metavariable *abfrageausdruck*).

Sollen hingegen **mehrere** Sätze auf **verschiedene** Weise geändert werden, so müssen sie in einer Ergebnistabelle (Cursortabelle) ausgewählt und mit dem Cursor einzeln angesprochen werden (**satzweise Änderung**). Geändert wird die Basistabelle, die dem Cursor bei der Deklaration zugrunde liegt.

Eine Cursortabelle brauchen Sie auch, wenn Sie einen oder mehrere Sätze verändern oder löschen und die Bedingungen dafür erst zum Ablaufzeitpunkt festlegen wollen.

Beim **Einfügen** von Sätzen in die Datenbank bewirkt die RETURN-Klausel der INSERT-Anweisung, daß sich der DRIVE-Anwender die vom Datenbank-System vergebene Zahl zur Identifizierung eines Satzes in eine DRIVE-Variable ausgeben lassen kann:

– für UDS den Primärschlüssel

– für SESAM V1 und V2 den Inhalt des Zählfeldes beim Compound Key (*-Angabe in der Klausel VALUES).

Sätze werden aus einer Tabelle ausgewählt durch die Angabe von Bedingungen in der WHERE-Klausel der SELECT-Anweisung oder der Metavariablen *select-ausdruck*. Mehrere Bedingungen können durch die logischen Operatoren AND oder OR (siehe „SQL-Lexika“ [4], [5], [6], Metavariable *bedingung*) verbunden werden. Zusätzlich können Ergebnistabellen (Views, Cursortabellen) Datensätze aus mehreren Tabellen enthalten. Die Tabellen werden verknüpft (Join), indem:

1. in der FROM-Klausel der SELECT-Anweisung oder der Metavariablen *select-ausdruck* alle betroffenen Tabellen angegeben werden und
2. entweder in der WHERE-Klausel ein Satzelement (SESAM V1, SESAM V2 und UDS) einer Tabelle mit dem Satzelement einer anderen Tabelle durch den Vergleichsoperator „=“ verbunden wird (dabei müssen die Satzelemente, die verglichen werden, den gleichen Datentyp haben)

   oder innerhalb der FROM-Klausel eine Joinbedingung durch eine ON-Klausel referenziert wird (nur SESAM V2).

Um beim Selektieren und Manipulieren auf die Sätze einer Cursortabelle zugreifen zu können, muß die Cursortabelle mit OPEN geöffnet und der Cursor mit FETCH auf die einzelnen Sätze positioniert werden.

Bei SESAM V1, SESAM V2 und UDS kann die aktuelle Cursorposition mit STORE über das Transaktionsende hinaus gespeichert und in einer späteren Transaktion mit RESTORE wiederhergestellt werden.

Für das Arbeiten mit einer Cursortabelle ergibt sich folgende Reihenfolge von Anweisungen für SESAM V1, SESAM V2 und UDS:

DECLARE *cursor*....

OPEN *cursor*...

FETCH *cursor*...

Bearbeiten der Cursortabelle mit DELETE, UPDATE, falls der Cursor änderbar ist

STORE *cursor*

RESTORE *cursor*

CLOSE *cursor*

DROP CURSOR *cursor (*ggf. *)*

Views (Sichten) sind virtuelle Tabellen, die einen Ausschnitt aus einer oder mehreren Basistabellen definieren.

SESAM V1 und UDS unterstützen nur temporäre Views, d.h. die Views werden am Ende einer Datenbank-Sitzung automatisch gelöscht. Bei SESAM V2 können Sie zwischen temporären und permanenten Views wählen.

In der Deklaration des View wird ein Name vereinbart und eine Datenbank-Abfrage mit einem *abfrageausdruck* angegeben. Der View umfaßt die Ergebnistabelle dieser Datenbank-Abfrage. Der Inhalt der Ergebnistabelle wird erst bestimmt, wenn sich eine Anweisung auf den deklarierten View bezieht. Der temporäre View muß im DRIVE-Programm vor allen Anweisungen deklariert werden, die den View verwenden. Wenn der View änderbar ist, können über den View auch Datensätze in die Basistabelle eingefügt oder geändert und gelöscht werden (siehe „DRIVE-SQL-Lexika“ [4], [5], [6], Anweisungen CREATE TEMPORARY VIEW und CREATE VIEW).

Transaktionen erhalten und sichern den konsistenten Zustand einer Datenbank. Eine Transaktion ist eine logische Folge von Anweisungen, die zwischen zwei Sicherungspunkten liegen. Bei Transaktionssicherung sollen entweder alle oder keine der in einer Transaktion liegenden Anweisungen ausgeführt werden. Der Beginn der ersten Transaktion wird nicht mit einer bestimmten SQL-Anweisung definiert. Die erste TA-initiierende SQL-Anweisung (siehe Abschnitt „Beginn einer Transaktion“ auf Seite 263) nach dem Programmstart oder dem nach Festschreiben oder Rücksetzen der vorherigen Transaktion wird vom Datenbanksystem als Transaktionsbeginn gewertet.

Fehlerbehandlung

Für bestimmte Fehlerklassen und DRIVE-Anweisungen können im Programm-Modus Reaktionen auf Fehler mit der WHENEVER-Anweisung festgelegt werden (siehe Abschnitt „Systemvariablen“ auf Seite 20 und Abschnitt „Fehler abfangen, Endekriterien“ auf Seite 88). DRIVE/WINDOWS reagiert dann nicht standardmäßig mit Programmabbruch, sondern führt die bei WHENEVER angegebenen Aktionen durch.

Fehler bei verteilter Verarbeitung können Sie in der Systemvariablen ERROR_STATE.ERROR über den Wert 'DIS ERROR' abfragen. Zusätzlich wird die Variable &DISTRIBUTION_STATE versorgt (siehe Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339 und Kapitel „Verteilte Anwendungen“ auf Seite 299).

NULL-Wert-Darstellung

Mit der Anweisung PARAMETER DYNAMIC NULL FORM legen Sie für ein alphanumerisches und/oder numerisches Datenfeld je ein Zeichen für die Darstellung des NULL-Werts am Bildschirm fest (siehe auch „DRIVE-Lexikon“ [3], Anweisung PARAMETER DYNAMIC).

Für alphanumerische Datenfelder kann als NULL-Wertzeichen jedes beliebige Zeichen verwendet werden. Für numerische Datenfelder ist als NULL-Wertzeichen nur eine Ziffer oder eines der Sonderzeichen * + - , . zugelassen.

Das NULL-Wertzeichen für alphanumerische Datenfelder ist auch gültig für Felder der Zeit-Datentypen. Das NULL-Wertzeichen für numerische Datenfelder ist auch gültig für Felder des Datentyps INTERVAL.

Das mit PARAMETER DYNAMIC vereinbarte NULL-Wertzeichen kann für DRIVE-Bildschirmformate durch Angabe eines anderen NULL-Wertzeichens in der NULL-Klausel der DECLARE FORM-Anweisung überschrieben werden.
Ein NULL-Wertzeichen, das über DECLARE FORM NULL festgelegt wurde, gilt nur für das jeweilige mit DECLARE FORM definierte Bildschirmformat.

Wenn Sie kein NULL-Wertzeichen explizit festlegen, gilt für alphanumerische oder numerische Datenfelder das Zeichen @ als Standardvorgabewert.

Entsprechend können Sie mit der Anweisung PARAMETER DYNAMIC NULL LIST ein Zeichen für die Darstellung des NULL-Werts bei Druckerausgaben festlegen. Hier ist der Standardvorgabewert ein Punkt.

9 Datenbank-Unterstützung

Dieses Kapitel beschreibt:

– die Objekte, die die einzelnen Datenbanksysteme kennen und wie sie angesprochen werden (Seite 246)

– Besonderheiten bei SESAM V2 (Seite 250)

– Syntaxunterschiede zwischen SQL-Dialekten und DRIVE/WINDOWS, insbesondere die DRIVE-Unterstützung der Cursorverarbeitung (ab Seite 255)

– Datenschutz über Zugriffsrechte und und Benutzeridentifikationen bei den verschiedenen Datenbanksystemen (Seite 81).

Mit der 4GL-Sprache DRIVE/WINDOWS können Datenbank-Anwendungen schnell und komfortabel entwickelt werden. Die Zugriffe auf Datenbanken werden in SQL (Structured Query Language) formuliert, der mit Abstand am weitesten verbreiteten relationalen Datenbank-Sprache.

Unterstützte DB-Server

DRIVE/WINDOWS ermöglicht mit SQL-Anweisungen den Zugriff auf drei verschiedene DB-Server:

– SESAM/SQL V1,

– UDS/SQL,

– SESAM/SQL V2,

Im BS2000 wird der DBH bei der Installation ausgewählt: SESAM/SQL V1, UDS/SQL oder SESAM/SQL V2. Der DRIVE-Anwender braucht keinen DBH zu spezifizieren, eventuelle Operanden von PARAMETER DYNAMIC- oder OPTION-Anweisungen werden ignoriert.

In einer DRIVE-Sitzung kann nur mit einem BS2000-Server, d.h. SESAM/SQL V1, SESAM/SQL V2 oder UDS/SQL, gearbeitet werden.

Mit der Funktion der Verteilten Transaktionsverarbeitung VTV können Sie jedoch verteilte DRIVE-Anwendungen programmieren, die auf alle BS2000-Server zugreifen (siehe Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339).

Verweise auf andere Handbücher

Eine ausführliche Beschreibung der SQL-Anweisungen finden Sie in der SQL-Sprachbeschreibung, die zum jeweiligen Datenbanksystem gehört:

– „SQL für SESAM/SQL“ V1 [17]

– „SQL für SESAM/SQL“ V2 [18]

– „SQL für UDS/SQL“ [25]

Die genaue Syntax der DRIVE-SQL-Anweisungen wird in Kurzform für jede Benutzerschnittstelle der verschiedenen Datenbanksysteme in einem zusätzlichen Handbuch dargestellt:

– „DRIVE-SQL-Lexikon für SESAM“ V1 [4]

– „DRIVE-SQL-Lexikon für SESAM“ V2 [5]

– „DRIVE-SQL-Lexikon für UDS“ [6]

Komplexe Anweisungsteile, die sowohl in DRIVE- als auch in SQL-Anweisungen enthalten sind, werden gesondert beschrieben im Kapitel Metavariablen des „DRIVE-Lexikon“ [3] und in den zusätzlichen Handbüchern für DRIVE-SQL (siehe [4], [5], [6]). Die DRIVE-Metavariablen stimmen weitgehend mit denen der Sprachbeschreibung für SESAM V1, SESAM V2 und UDS überein, d.h. es gibt eine eindeutige Zuordnung.

9.1 SQL-Objekte in DRIVE/WINDOWS

Ein DB-Server wird angesprochen durch Transaktionsanweisungen sowie durch Referenzen auf Datenbanken oder SQL-Objekte, die von diesem DB-Server (DBH) verwaltet werden. Zu den **persistenten** Objekten, d.h. Objekten, die in der Datenbank gespeichert werden, gehören Basistabellen der Anwendungen, Systemtabellen des DB-Servers, Spalten, Constraints, Indizes, Views und Synonyme. Namen und Strukturen persistenter SQL-Objekte gehören zu den Metadaten der jeweiligen Datenbank und werden vom Datenbanksystem in Systemtabellen verwaltet. Als Anwendung verwaltet DRIVE/WINDOWS keine Informationen über persistente SQL-Objekte. Mit der Anweisung DECLARE VARIABLE... LIKE TABLE... ist es allerdings möglich, Variablen mit der Struktur von Basistabellen, Systemtabellen oder Views zu definieren.

Namen und Strukturen von **temporären SQL-Objekten** gehören zu den anwendungsspezifischen Daten und werden vom SQL-Laufzeitsystem des DB-Servers verwaltet. Sie existieren höchstens bis zum Ende der DRIVE-Sitzung. Temporäre SQL-Objekte sind vor allem Cursor sowie temporäre Tabellen, temporäre Views und Korrelationen. Mit der Anweisung DECLARE VARIABLE ... LIKE ... können Sie Variablen mit der Struktur von temporären Views oder Cursortabellen definieren.

Mit dem Zuweisen einer Datenbasis sind DRIVE/WINDOWS alle Daten und Metadaten bekannt, d.h. alle Basis- und Systemtabellen und Systemviews eines SQL-Schemas . Die aktuelle Datenbasis weisen Sie zu mit den Anweisungen PARAMETER DYNAMIC DBSYSTEM oder OPTION DBSYSTEM und zusätzlich ggf. mit:

– der Anweisung PERMIT bei SESAM V1 und UDS

– den CATALOG-, SCHEMA- und AUTHORIZATION-Operanden bei SESAM V2 (siehe DRIVE-SQL-Lexikon [5].

Eine **Basistabelle** wird spezifiziert durch den Tabellennamen und die Tabellenstruktur. Die Tabellenstruktur wird durch Spalten und Tabellenconstraints (bei SESAM V2) spezifiziert. Optional können Sie die physikalische Speicherungsart spezifizieren. Die folgende Tabelle zeigt, wie Sie Tabellennamen für die von DRIVE/WINDOWS unterstützten DB-Server qualifizieren:

Datenbankserver	**Tabellenname**
SESAM V1	[schema-name .] tabellenname dabei ist der Schemaname gleich dem Tabellennamen, und beide sind gleich dem Datenbanknamen; zwei Relationen sind genau dann gleich, wenn sie den gleichen Schema- und Datenbanknamen haben
SESAM V2	[catalog-name .] [schema-name .] tabellenname zwei Relationen sind genau dann gleich, wenn sie den gleichen Katalog-, Schema- und Tabellennamen haben
UDS	[subschema-name .] satzartname

Die folgende Tabelle zeigt die Speicheroptionen für die von DRIVE/WINDOWS unterstützten DB-Server

Datenbankserver	**Speicheroptionen**
SESAM V1	Datei- u. Geräteangaben bei der Funktion URLADEN des Administrationsmonitors SESASB
UDS	Realm-Angaben in der Schema-DDL u. beim Dienstprogramm BINILOAD, SSL-Beschreibungen
SESAM V2	USING SPACE [catalogname .] space

Die (ggf. strukturierte) **Spalte** einer Basistabelle wird spezifiziert mit dem Spaltennamen, dem Datentyp, einem optionalen Defaultwert und einem optionalen Spaltenconstraint. Der Spaltenname ist bei allen DB-Servern ein tabellenweit eindeutiger Name. Bei SESAM V1 und UDS gelten besondere Regeln für Defaultwerte. Außerhalb einer Tabellenspezifikation wird eine Spalte (oder eine oder mehrere Spaltenkomponenten) mit der Metavariablen *satzelement* spezifiziert (siehe „DRIVE-SQL-Lexika“ [4], [5], [6]]).

Für SESAM V2 gilt folgender Zugriffsschutz: Ein autorisierter Benutzer (Grantor) vergibt **Privilegien** und berechtigt einen anderen Benutzer (Grantee), eine bestimmte Anweisung auf einem bestimmten SQL-Objekt auszuführen, z.B. SELECT auf die Tabelle eines Schemas. Mit der Anweisung GRANT vergeben Sie Privilegien, mit der Anweisung REVOKE entziehen Sie Privilegien und regeln so den Zugriffsschutz in SQL.

Eine Datenbank kann in **Schemata** aufgeteilt sein, die von Anwendern oder vom Datenbanksystem eingerichtet werden. Ein anwenderdefiniertes Schema ist einem Anwender zugeordnet, der Eigentümer des Schemas ist. Es enthält Metadaten zu Basistabellen, Views, Indizes und Privilegien. Systemdefinierte Schemata fassen Systemviews zusammen und enthalten datenbankspezifische Metadaten.

Views sind virtuelle Tabellen, die einen Ausschnitt aus einer oder mehreren Basistabellen definieren. In SESAM V2 gibt es persistente Views (siehe folgender Abschnitt). Außerdem können Sie mit der Anweisung CREATE TEMPORARY VIEW anwendungsspezifisch **temporäre Views** definieren und mit DROP TEMPORARY VIEW wieder löschen. Der Viewname ist eindeutig in bezug auf alle anwendungsspezifischen, temporären Views und datenbankweiten, persistenten Views und Tabellen.

Die Anweisung CREATE TEMPORARY VIEW ist in Programmen statisch nur am Anfang eines DRIVE-Programms erlaubt, d.h. vor den Verarbeitungsanweisungen. Zum Ablaufzeitpunkt können mit der Anweisung EXECUTE dynamisch temporäre Views definiert werden.

In DRIVE/WINDOWS können temporäre Views im Dialog-Modus sowie im Programm-Modus statisch (mit CREATE) und dynamisch (mit EXECUTE) definiert werden. Das an der ESQL-Schnittstelle notwendige Präfix „MODULE.“ entfällt in DRIVE/WINDOWS. Ein temporärer View darf sich nicht auf andere (temporäre oder persistente) Views beziehen. Der Gültigkeitsbereich eines temporären Views entspricht dem eines Cursor (siehe Seite 257). Dynamische Programm-Views und Dialog-Views können Sie löschen. Statische und vom Anwender noch nicht gelöschte dynamische Programm-Views werden von DRIVE/WINDOWS beim Übergang in den Dialog-Modus gelöscht. Vom Anwender noch nicht gelöschte Dialog-Views werden am Ende der DRIVE-Sitzung gelöscht (STOP oder EXIT). Zu Gültigkeitsbereich und Lebensdauer siehe auch Seite 257.

Sie können temporäre **Synonyme** für Tabellen- und Viewnamen (sog. Korrelationen bei SESAM V2) bei allen DB-Servern in Suchabfragen verwenden. Diese Synonyme existieren nur solange wie die Suchabfrage in Bearbeitung ist.

9.1.1 Zusätzlich unterstützte SQL-Objekte bei SESAM V2

– In SESAM V2 können Sie Tabellen- und Spalten**constraints** definieren, die aus dem optionalen Constraintnamen und der Spaltenangabe bestehen (siehe „DRIVE-SQL-Lexikon“ [5]], CREATE TABLE und ALTER TABLE). Der Constraintname ist datenbankweit eindeutig.

Für SESAM V2 unterstützt DRIVE/WINDOWS z.Zt. Indizes nicht in der Syntax. Basistabellen, die außerhalb von DRIVE/WINDOWS mit Indizes spezifiziert wurden, können jedoch in DRIVE/WINDOWS benutzt werden.

– Bei SESAM V2 können Sie mit einer etwas eingeschränkten SELECT-Anweisung, die Basistabellen und andere persistente Views referenziert, **persistente Views** definieren (siehe „DRIVE-SQL-Lexikon“ [5] , CREATE VIEW). Die SQL-Sprachbeschreibung von SESAM V2 [20] enthält einen Überblick zu den Systemviews von SESAM V2.

  Es wird empfohlen, bei SESAM V2 mit persistenten Views zu arbeiten.

9.1.2 SQL-Objekte definieren mit DDL-Anweisungen

Alle Nutzdaten werden logisch in Basistabellen organisiert, die der Anwender für SESAM V2 mit DDL(Data Definition Language)-Anweisungen erzeugt (in SESAM V1 und UDS mit Dienstprogrammen).

DRIVE/WINDOWS für SESAM V2 bietet weitere DDL-Anweisungen für die Definition von Datenstrukturen und Integritätsbedingungen. Diese Anweisungen sind ausführbare SQL-Anweisungen und müssen daher im Ausführungsteil von Programmen stehen (siehe auch „DRIVE-SQL-Lexikon“ [5]):

SQL-Anweisung	**DB-Server**
ALTER TABLE	SESAM V2
CREATE SCHEMA	SESAM V2
CREATE TABLE	SESAM V2
CREATE TEMPORARY VIEW	SESAM V1 u. V2, UDS
CREATE VIEW	SESAM V2
DROP SCHEMA	SESAM V2
DROP TABLE	SESAM V2
DROP TEMPORARY VIEW	SESAM V1 u. V2, UDS
DROP VIEW	SESAM V2

Achten Sie darauf, daß eine DDL- Anweisung im Ausführungsteil eines Programms keine deklarative Anweisung im Deklarationsteil eines Programms zerstört: Z.B. Anlegen eines Cursors im Deklarationsteil und Ändern der Tabelle, auf die der Cursor deklariert wurde, über die ALTER TABLE-Anweisung im Ausführungsteil des Programmes. Wird anschließend versucht, über den Cursor auf die Tabelle zuzugreifen, kommt es zu einem SQL-Fehler.

Die DDL-Anweisungen können auch dynamisch, d.h. zum Ablaufzeitpunkt, angegeben werden (siehe „DRIVE-Lexikon“ [3], Anweisung EXECUTE).

Bei SESAM V2 können in ein- und derselben Transaktion entweder nur DDL-Anweisungen oder keine DDL-Anweisungen ausgeführt werden, d.h. ein Mischen von DML- und DDL-Anweisungen ist nicht möglich.

9.2 Besonderheiten bei SESAM V2

DRIVE/WINDOWS V2 unterstützt den DB-Server SESAM V2 mit

– fast allen numerischen, alphanumerischen und Zeitpunktausdrücken (siehe „DRIVE-SQL-Lexikon für SESAM“ V2 [5], Metavariable *sqlausdruck*), wobei allerdings die SESAM V2-Zeitpunktfunktionen in DRIVE/WINDOWS ohne Unterstrich spezifiziert werden müssen.

– allen Bedingungen (Metavariable *bedingung*) und

– allen Abfrageblöcken (Metavariable *abfrageausdruck*).

Mit der Anweisung DECLARE VARIABLE...LIKE TABLE/CURSOR... werden SESAM-Datentypen auf DRIVE-Datentypen abgebildet (siehe Kapitel „Variablen und Konstanten einsetzen“ auf Seite 17). DRIVE-Datentypen werden den SESAM-Ausdrücken in der SELECT-Liste von Cursorbeschreibungen entsprechend den SESAM V2-Konventionen zugewiesen, d.h. mit der SQL-Norm übereinstimmend (siehe „DRIVE-SQL-Lexika“ [4], [5], [6], DECLARE CURSOR-Anweisung und Metavariable *abfrageausdruck*).

9.2.1 Unterschiede zwischen SESAM V1 und V2

Die Übersetzung eines DRIVE-Programms für SESAM V2 muß in einer eigenen Transaktion ablaufen. Wenn bei einer CALL-Anweisung eine Transaktion offen ist und kein Zwischencode des mit CALL gerufenen Programms vorliegt, wird das Programm abgebrochen. D.h. bei einer CALL-Anweisung muß entweder Zwischencode vorliegen oder die Transaktion abgeschlossen sein.

DRIVE/WINDOWS unterstützt den SQLSTATE von SESAM V2 durch die neue Systemvariable &SQL_STATE (siehe Kapitel „Variablen und Konstanten einsetzen“ auf Seite 17). Die Fehlerklassen über &DML_STATE werden nach dem SQLCODE von SESAM V2 klassifiziert, die Fehlerklassen über &SQL_STATE.CLASS normkonform. Für DRIVE V6- und DRIVE/WINDOWS V1-Anwendungen wird empfohlen, die Fehlerbehandlung von &SQL_CODE und &DML_STATE auf &SQL_STATE umzustellen. Das Handbuch „SESAM/SQL-Server, Meldungen“ [43] stellt SQLCODEs und SQLSTATEs gegenüber.

- Temporäre Views sind nicht normkonform. SESAM V2 unterstützt sie aus Kompatibilitätsgründen. DRIVE/WINDOWS verwaltet temporäre Views auch beim Erzeugen und Ausführen von Plänen für SQL-Anweisungen, die temporäre Views referenzieren. Aus Performancegründen wird daher empfohlen, DRIVE-Anwendungen von temporären auf persistente Views umzustellen. Dies kann z.B. session- oder vorgangsspezifisch geschehen, indem die anwendungsspezifischen Views zu Beginn einer Session oder eines Vorgangs definiert und am Ende wieder gelöscht werden.

- SESAM V2 verwendet die zwei Information Schemata
  - INFORMATION_SCHEMA (benutzerspezifischer Zugriff),
  - SYS_INFO_SCHEMA (Zugriff zunächst nur durch den universellen Benutzer).

  Informationen aus diesen Schemata können Sie mit SELECT-Anweisungen und Cursor-Deklarationen abfragen (siehe SQL-Lexikon [5]).

- Bei SESAM spricht man bei Datenbanken auch von Katalogen. Bei SESAM V1 enthält ein Katalog ein SQL-Schema, das eine Basistabelle enthält. Katalog-, Schema- und Tabellenname sind also identisch. Bei SESAM V2 kann ein Katalog mehrere SQL-Schemata mit mehreren Basistabellen enthalten. Es gibt einfache Tabellennamen und mit Katalog- und Schemanamen qualifizierte Tabellennamen(siehe Seite 254). Die Migration von SESAM V1 Basistabellen in SESAM V2 Kataloge ist also nicht immer eindeutig und kann sich auf die Anwendungen auswirken.

- Die Anweisung PERMIT hat bei SESAM V2 im New-Style keine Wirkung, sondern setzt nur den SQLSTATE. Sie wird jedoch im Old-Style für den Zugriff auf paßwortgeschützte CALL-DML-Tabellen benötigt. Die Anweisung GRANT vergibt Privilegien, REVOKE entzieht sie.

- Anweisungen zur Session-Einstellung:
  - SET CATALOG - Datenbanknamen voreinstellen (nur für dynamische SQL)
  - SET SCHEMA - Schemanamen voreinstellen (nur für dynamische SQL)
  - SET SESSION AUTHORIZATION (Berechtigungsschlüssel für die aktuelle SQL-Session voreinstellen)

- DDL-Anweisungen (siehe oben)

- Zum Konzept des Konsistenzlevels und dem Beginn einer Transaktion siehe Kapitel „Transaktionskonzept“ auf Seite 261

DRIVE-Anwendungen für SESAM V1 auf SESAM V2 umstellen

Mit DRIVE V6 (BS2000) oder DRIVE/WINDOWS V1 (BS2000/SINIX) erstellte Anwendungen mit SESAM V1 sind sourcekompatibel im Zugriff auf SESAM V2, da die SQL-Anweisungen von SESAM V2 kompatibel mit den SQL-Anweisungen von SESAM V1 sind. Voraussetzung: Alle in der jeweiligen Anwendung verwendeten Tabellen von SESAM V1 sind in dasselbe SQL-Schema eines Katalogs migriert worden. Andernfalls müssen ggf. Tabellennamen mit Katalog- und Schemanamen qualifiziert werden oder entsprechende OPTION-Anweisungen für statische SQL und SET-Anweisungen für dynamische SQL ergänzt werden (vgl. Beispiel im „DRIVE-SQL-Lexikon für SESAM“ V2 [5]).

Um den Änderungsumfang für bestehende DRIVE-Anwendungen so gering wie möglich zu halten, können Sie sich einer neuen Defaultwerttechnik für den SQL-User-Namen (Berechtigungsschlüssel), den Katalognamen und den Schemanamen bedienen, die über die Operanden AUTHORIZATION, CATALOG und SCHEMA der PARAMETER DYNAMIC-Anweisung realisiert wurde. Haben Sie die oben genannte Migrationsvoraussetzung beachtet, so kommen Sie sogar ohne jegliche Sourceänderung aus. Sie brauchen dann nur zu Beginn der DRIVE-Sitzung den einen Katalognamen, den einen Schemanamen und den Eigentümer dieses Schemas als SQL-User anzugeben (siehe DRIVE-SQL-Lexikon [5]).

Die Operanden CATALOG und AUTHORIZATION der Anweisung PARAMETER DYNAMIC werden in SESAM V1-Umgebung mit der DRIVE-Fehlermeldung DRI0229 abgewiesen.

Die Operanden CATALOG und AUTHORIZATION der Anweisung OPTION werden in SESAM V1-Umgebung ignoriert. Der Operand SCHEMA bei PARAMETER DYNAMIC und OPTION hat eine andere Bedeutung (siehe „DRIVE-SQL-Lexika“ [4], [5]).

9.2.2 SESAM V2-DBH zuordnen

Wie bei SESAM V1 greift DRIVE/WINDOWS auf eine SESAM V2-Datenbank mit Hilfe eines zugeordneten DBH (Database Handler) zu. Dabei sind folgende Einsatzformen zu unterscheiden:

– lokaler Zugriff im BS2000 (TIAM- oder UTM-Betrieb)

  DRIVE/WINDOWS wird mit einem Konnektionsmodul (SESMOD im TIAM-, SESUTMC im UTM-Betrieb) gebunden, dem beim Starten der Name eines „independent DBH“ und der Konfigurationsname mitgegeben wird. Dieser DBH wurde zuvor mit einer passenden Konfigurationsdatei gestartet. Der zugeordnete DBH kann bis zu 254 Datenbanken verwalten, auf deren Daten und Metadaten in einer DRIVE-Sitzung (TIAM-Betrieb) oder im UTM-Vorgang (UTM-Betrieb) zugegriffen werden kann. Voraussetzung: Die erforderlichen Zugangsberechtigungen sind vorhanden.

Bei verteilten Datenbanken mit SESAM-DCN kann neben diesem „Home-DBH“ mit weiteren „Remote-DBH“ gearbeitet und auf die von ihnen verwalteteten Datenbanken zugegriffen werden. Der Zugriff ist für den DRIVE/WINDOWS-Anwender transparent. Zu einem Zeitpunkt kann jede Datenbank nur von einem DBH verwaltet werden.

– Remote-Zugriff

  Über DRIVE/WINDOWS-NET arbeiten Sie auf dem Client und greifen transparent auf den Datenbestand des BS2000-Servers zu. Diesem können, wie oben bei lokalem Zugriff beschrieben, ein oder mehrere DBHs zugeordnet werden. Zur Konfigurierung von DRIVE/WINDOWS-NET siehe DRIVE-SPU [7, 10], Kapitel Remote-Zugriffe auf BS2000-Datenbanken.

– verteilte DRIVE-Anwendungen mit UPIC

  Auf der BS2000-Seite wird ein UTM-UPIC-Server eingesetzt, dem, wie oben bei lokalem Zugriff beschrieben, ein oder mehrere DHBs zugeordnet werden (siehe Kapitel „Verteilte Anwendungen“ auf Seite 299). Auf der Client-Seite enthält DRIVE/WINDOWS-NET eine UTM-UPIC-Connection-Phase, über die DRIVE/WINDOWS als Client mit dem BS2000-Server kommuniziert. Der Zugriff auf SESAM V2 erfolgt lokal im BS2000-Server. Es kann nicht gleichzeitig ein Remote-Zugriff auf SESAM erfolgen, da über UPIC zu einem Zeitpunkt nur ein Server adressiert werden kann.

– verteilte DRIVE-Anwendungen mit UTM-D (Verteilte Transaktionsverarbeitung VTV)

  Als Server wird im BS2000 eine UTM-Anwendung eingesetzt, die die UTM-X-Anwendung im SINIX adressiert und umgekehrt. Dem Client werden dadurch dieselben DBHs und Datenbanken zugeordnet wie dem Server. Die Zuordnung von DBH und Datenbanken wird durch die Generierung der UTM-Anwendung im Server bestimmt. Jeder Partner (Client oder Server) auf SINIX kann über UTM-D zusätzlich einen Remote-Zugriff auf SESAM ausführen, und jeder Partner (Client oder Server) im BS2000 kann lokal auf SESAM zugreifen.

Zugangsberechtigungen zu einer SESAM V2-Datenbank

Beim Anlegen einer Datenbank mit der SESAM V2-Anweisung CREATE CATALOG über den Utility-Monitor durch einen befugten BS2000-Systembenutzer kann ein BS2000-Kennwort und muß ein Berechtigungsschlüssel angegeben werden. Das Kennwort wird auf alle Dateien der Datenbank vererbt und muß beim Starten eines DBH, der die Datenbank verwalten soll, angegeben werden. Der Berechtigungsschlüssel identifiziert einen SQL-Benutzer, dem eine BS2000-Systembenutzer-Kennung zugeordnet ist. Die Standard-Voreinstellung ist die Kennung, unter der die Anweisung CREATE CATALOG angegeben wurde.

Ein Berechtigungsschlüssel zusammen mit einer Systembenutzer-Kennung heißt Systemzugang. Der mit CREATE CATALOG festgelegte Systemzugang ist der erste für die Datenbank und identifiziert den sog. universellen Benutzer. Dieser kann mit der Anweisung

CREATE USER über den Utility-Monitor von SESAM weitere Berechtigungsschlüssel einrichten und mit CREATE SYSTEM USER im Utility-Monitor Berechtigungsschlüssel zu Systemzugängen erweitern.

Wollen Sie mit einem DRIVE-Programm auf eine SESAM V2-Datenbank zugreifen, muß ein Systemzugang eingerichtet sein, der je nach gewählter Einsatzform folgenden Bedingungen genügt:

– bei lokalem Zugriff im BS2000 muß die Systembenutzer-Kennung mit der Kennung von DRIVE/WINDOWS als TIAM- oder UTM-Anwendung übereinstimmen

– bei Remote-Zugriff muß die Systembenutzer-Kennung mit der des zugehörigen UTM-UPIC-Servers übereinstimmen.

– Bei verteilten Anwendungen muß nur der Client als SQL-Benutzer ausgewiesen werden.

In DRIVE/WINDOWS weist man sich über den Operanden AUTHORIZATION der OPTION-Anweisung als SESAM V2-Benutzer aus. Der Server und der Auftragnehmer müssen sich selbst ausweisen. Die Angabe AUTHORIZATION gilt für die DRIVE-Sitzung und kann in transaktionslosem Zustand durch die DRIVE-SQL-Anweisung SET SESSION geändert werden (siehe „DRIVE-SQL-Lexikon“ [5]).

9.2.3 Kataloge und SQL-Schemata angeben

Durch die Organisation von Tabellen, Views und Indizes in Datenbanken und Schemata kann jedes dieser SQL-Objekte nur in bezug auf einen bestimmten Katalog und ein bestimmtes SQL-Schema eindeutig identifiziert werden: Der einfache Objektname wird qualifiziert mit dem Schemanamen und dem Katalognamen `[ [catalog–name . ] [schema–name . ] ]`. Werden der Katalogname und/oder der Schemaname nicht angegeben, gelten die internen Standardvoreinstellungen. Diese ergeben sich für statische SQL aus den Operanden CATALOG und SCHEMA der OPTION-Anweisung; für dynamische SQL werden sie mit den Anweisungen SET CATALOG und SET SCHEMA eingestellt oder über die Anweisung PARAMETER DYNAMIC (siehe „DRIVE-SQL-Lexikon“ [5]).

Für die Übersichtlichkeit einer DRIVE-Anwendung mit SESAM/SQL V2 wird empfohlen, die Anwendung so in einzelne Programme zu strukturieren, daß jedes Programm zumindest statisch hauptsächlich auf SQL-Objekte eines Schemas zugreift. Dieses Schema und der dazugehörige Katalog können dann in der OPTION-Anweisung eingestellt werden. Zugriffe auf SQL-Objekte eines anderen Schemas des eingestellten Katalogs oder auf Schemata eines anderen Katalogs müssen dann qualifizierte Objektnamen verwenden.

9.3 Syntaxunterschiede SQL-Dialekte - DRIVE/WINDOWS

Dieser Abschnitt listet die wesentlichen Funktionen und Syntaxelemente auf, die DRIVE/WINDOWS zusätzlich zu oder abweichend von den jeweiligen SQL-Sprachbeschreibungen des DB-Servers unterstützt.

- Variablen

  Der Begriff der Variablen in DRIVE/WINDOWS steht für eine Benutzervariable, d.h. für eine Variable, die der Programmierer im Deklarationsteil des Programms selbst definiert.
  Die DRIVE-Metavariable *varname* bezeichnet eine Benutzervariable. Diese muß das Präfix „&“ haben.
  Der Begriff Indikatorvariable bezeichnet in DRIVE-Anweisungen eine Variable, die zusammen mit der USING-Klausel bei Parameterübergaben in andere Programmiersprachen benötigt wird (siehe Kapitel „Variablen und Konstanten einsetzen“ auf Seite 17). Der Indikatorwert gibt an, ob der zugehörige Parameterwert als NULL-Wert zu interpretieren ist oder nicht.
  Bei SESAM und UDS ist die Semantik der Indikatorvariablen erweitert (siehe „SQL für SESAM“ V1 [4], „SQL für SESAM“ V2 [5] und „SQL für UDS“ [6]). Innerhalb von SQL-Anweisungen werden in DRIVE/WINDOWS keine Indikatorvariablen benötigt. Vielmehr beinhalten DRIVE-Variablen stets auch NULL-Wertanzeigen.

  Mit den Klauseln LIKE TABLE und LIKE CURSOR der DECLARE VARIABLE-Anweisung können Sie komfortabel eine strukturierte Variable &varname definieren, deren Struktur der angegebenen Tabelle (Basistabelle, Systemtabelle, View, temporärer View) oder dem angegebenen Cursor (Ergebnistabelle) entspricht. Innerhalb von SQL-Anweisungen kann dann durch die Angabe von &varname.* komfortabel eine Variablenliste spezifiziert werden, die der Spaltenliste der Tabelle/des Cursors entspricht.

  Beispiele

  ```
  INSERT INTO TABLE tabelle VALUES (&varname.*);

  FETCH cursor INTO &varname.*;
  ```

- Dialog- und Programm-Modus

  Bei den Anweisungen FETCH und SELECT besteht in DRIVE/WINDOWS ein Unterschied zwischen Dialog- und Programm-Modus:

  Im Dialog-Modus werden die Werte von Satzelementen am Bildschirm ausgegeben. Die INTO-Klausel ist nicht erlaubt. Die INTO-Klausel muß im Programm-Modus angegeben werden. Die Werte von Satzelementen werden mit der INTO-Klausel in Variablen übertragen.

- Transaktionssicherung

  DRIVE/WINDOWS bietet für die Anweisung ROLLBACK WORK die Klausel WITH RESET an. Die einfache ROLLBACK-Anweisung setzt alle Datenbank-Anweisungen in der aktuellen Transaktion zurück. Mit der WITH RESET-Klausel wird zusätzlich die DRIVE-Transaktion zurückgesetzt. D.h. die Inhalte von Variablen und Formaten werden auf den letzten COMMIT-Stand zurückgesetzt, und der Programmlauf wird nach der letzten COMMIT WORK-Anweisung fortgesetzt (siehe Kapitel „Transaktionskonzept“ auf Seite 261 und bei Verteilung die entsprechenden Kapitel Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339 und Kapitel „Verteilte Anwendungen“ auf Seite 299).

- DRIVE-Anweisungen und -Klauseln für Cursor

  DRIVE/WINDOWS bietet für alle unterstützten DB-Server eine DECLARE CURSOR-Anweisung an, die gegenüber der SQL-Norm erweitert ist und zu unterschiedlichen Cursortypen führt:

Cursortyp	**DECLARE-Anweisung**
permanenter Cursor	mit PERMANENT
temporärer Cursor	mit TEMPORARY (Vorbelegung)
SCROLL-Cursor	mit SCROLL (nicht UDS)
Schub-Cursor	mit PREFETCH n (SESAM V1 und SESAM V2)
UPDATE-Cursor	mit FOR UPDATE (SESAM V2)
statischer Cursor	im Dialog-Modus oder im Deklarationsteil eines DRIVE-Programms
dynamischer Cursor	mit einer EXECUTE-Anweisung zum Ablaufzeitpunkt
variabler Cursor	im Deklarationsteil eines DRIVE-Programms ohne FOR-Klausel und zum Ablaufzeitpunkt mit einer EXECUTE-Anweisung um eine dynamische FOR-Klausel ergänzt

  – Mit der DRIVE-Anweisung CYCLE *cursorname* INTO *variable* können Sie komfortabel in einer Cursorschleife Datenbanksätze in eine Variablenliste einlesen (siehe Abschnitt „Schleifen programmieren (CYCLE)“ auf Seite 86 und „DRIVE-Lexikon“ [3]).

  – Schubmodus bei SESAM V1 und SESAM V2

    Die PREFETCH-Klausel für Cursorverarbeitung im Schubmodus aus der SESAM V1 (siehe DRIVE-SQL-Lexikon [4], DECLARE CURSOR) gilt mit derselben Semantik bei SESAM V2. Darüberhinaus bietet DRIVE/WINDOWS bei SESAM V2 die Möglichkeit, den Schubmodus über eine PRAGMA-Anweisung zu aktivieren (siehe „DRIVE-SQL-Lexikon“ [5]).

- Zum variablen Cursor siehe Abschnitt „Dynamische SQL-Anweisungen“ auf Seite 104 und „SQL-Lexika“ [4], [5], [6]. Zum Gültigkeitsbereich des variablen Cursor siehe Abschnitt „Auswirkungen der Transaktionssteuerung auf Definitionen“ auf Seite 281.
- Die DRIVE-Anweisungen DROP CURSOR *cursor* und DROP CURSORS löschen Cursor. Die Anweisungen können im Dialog-Modus von DRIVE/WINDOWS oder dynamisch, d.h. zum Ablaufzeitpunkt, erzeugt und ausgeführt werden. D.h. mit DROP können Dialog-Cursor, variable Cursor und dynamische Cursor gelöscht werden.
- Im Programm-Modus bleibt ein Cursor, der mit PERMANENT definiert wurde, über das Programmende hinaus mit seiner Cursorposition erhalten, wenn das Programm mit CALL aufgerufen wurde. Ein Cursor, der mit TEMPORARY definiert wurde, wird bei Programmende geschlossen, die Cursorposition geht verloren.
- In einer DRIVE-Sitzung können im Dialog- und Programm-Modus insgesamt maximal 20 nicht-statische SESAM V2-Cursor angelegt werden.
- Informationen über Dialog-Cursor können Sie im Dialog-Modus mit SHOW CURSOR(s) abfragen (gilt nicht für SESAM V2).

- NULL-Wert-Darstellung

  DRIVE/WINDOWS bietet mit der Anweisung PARAMETER DYNAMIC NULL die Möglichkeit, für Ausgaben die Darstellung von **NULL-Werten** abweichend vom Standard zu wählen (bei Bildschirmformaten das Sonderzeichen @, bei Druckerausgaben den Punkt „.“, siehe „DRIVE-Lexikon“ [3], Anweisung PARAMETER DYNAMIC). Die mit PARAMETER DYNAMIC vereinbarte NULL-Wert-Darstellung bei Ausgaben kann für DRIVE-Bildschirmformate durch Angabe einer anderen NULL-Wert-Darstellung in der NULL-Klausel der DECLARE FORM-Anweisung überschrieben werden. Wird die Darstellung des NULL-Wertes nicht mit der PARAMETER- oder DECLARE FORM-Anweisung vereinbart, gilt die Standard-Voreinstellung.

9.3.1 Gültigkeitsbereich von Cursor und temporärem View

Ein Dialog-Cursor/temporärer Dialog-View ist gültig zwischen den Anweisungen DECLARE und DROP. Er ist in keinem Programm gültig.

Ein statischer oder variabler Programm-Cursor/temporärer Programm-View ist in dem Programm gültig, in dem er deklariert wurde: Im Deklarationsteil ab seiner DECLARE-Anweisung, im Ablaufteil (Body) und in jedem internen Unterprogramm.

Ein dynamischer Programm-Cursor/ temporärer Programm-View ist zum Ablaufzeitpunkt des Programms gültig für alle Anweisungen, die nach seiner Deklaration durchlaufen werden. Variable Programm-Cursor verhalten sich gegenüber dem DB-Server wie dynamische.

Siehe auch Kapitel „Transaktionskonzept“ auf Seite 261.

9.3.2 Lebensdauer von Cursor und temporärem View

Die Lebensdauer eines Dialog-Cursor reicht von seiner DECLARE-Anweisung bis zum Löschen durch den Anwender (DROP) oder DRIVE/WINDOWS am Ende der DRIVE-Sitzung.

Die Lebensdauer eines statischen Programm-Cursor beginnt zum Übersetzungszeitpunkt bei seiner DECLARE-Anweisung und zum Ablaufzeitpunkt beim Aufruf des Programms, in dem er deklariert wird, mit DO oder CALL.

Die Lebensdauer eines variablen Programm-Cursor beginnt bei DRIVE/WINDOWS mit der statischen Deklaration seines Namens, beim DB-Server mit der dynamischen Deklaration seiner FOR-Klausel.

Bei einem reinem Übersetzungslauf (Anweisung COMPILE) endet die Lebensdauer aller Programm-Cursor mit der Übersetzung. Bei einer Übersetzung mit anschließender Ausführung (Anweisungen DO, DEBUG, CALL) bleiben die Cursor zum Ablaufzeitpunkt erhalten.

Temporäre statische, dynamische oder variable Programm-Cursor werden von DRIVE/ WINDOWS bei Transaktionsende (COMMIT WORK) auf der nächst höheren Programmstufe gelöscht oder spätestens beim Übergang in den Dialog-Modus. Permanente Programm-Cursor werden von DRIVE/WINDOWS beim Übergang in den Dialog-Modus gelöscht, falls das Programm mit CALL aufgerufen wurde und im rufenden Programm kein COMMIT oder ROLLBACK WORK erfolgt. D.h. permanente offene Cursor behalten bei Programmende ihre Position unverändert bei. Dynamische und variable Cursor können auch anwenderseitig gelöscht werden (siehe DROP CURSOR-Anweisung in den DRIVE-Lexika [4], [5] und [6]).

Dynamische temporäre Programm-Views und temporäre Dialog-Views können vom Anwender gelöscht werden. Statische sowie noch vorhandene dynamische temporäre Programm-Views werden von DRIVE/WINDOWS beim Übergang in den Dialog-Modus gelöscht. Noch vorhandene temporäre Dialog-Views löscht DRIVE/WINDOWS beim Ende der DRIVE-Sitzung.

Zur Lebensdauer von Cursor und temporärem View siehe auch Abschnitt „Auswirkungen der Transaktionssteuerung auf Definitionen“ auf Seite 281.

9.3.3 Cursorposition

Die Cursorposition ist ein Verweis vor, auf oder hinter genau einen Satz der Cursor-Ergebnistabelle. Sie ist nur während der Lebensdauer des Cursor definiert und wird durch folgende Anweisungen beeinflußt:

Anweisung	**Cursorposition nach erfolgreicher Anweisungsausführung**
BREAK CYCLE	unspezifiziert
CLOSE	unspezifiziert
CYCLE	auf den nächsten Satz oder unspezifiziert
DECLARE CURSOR	unspezifiziert
DELETE ...WHERE CURRENT OF...	vor den nächsten Satz oder hinter den letzten Satz
DROP CURSOR(S)	undefiniert
END CYCLE	unspezifiziert
FETCH	auf den positionierten Satz
OPEN	vor den ersten Satz
RESTORE	nach den zum letzten FETCH gehörigen Satz
STORE	vor den nächsten oder nach den letzten Satz und abgespeichert (kein FETCH mehr möglich)
UPDATE...WHERE CURRENT OF...	auf den zum letzten FETCH gehörigen Satz

DRIVE/WINDOWS schließt offene temporäre statische Programm-Cursor am Ende des Programms, in dem sie deklariert wurden, oder bei Programmabbruch. Permanente offene Cursor hingegen bleiben offen und behalten daher ihre Cursorposition bei. Die Auswirkungen von Transaktionsanweisungen auf die Lebensdauer und Position eines Cursor ist im Kapitel „Transaktionskonzept“ auf Seite 261 beschrieben.

9.4 Zugriffsschutz

DRIVE/WINDOWS unterstützt die Schutzmechanismen von Datenbanksystemen und stellt Ihnen Anweisungen zur Verfügung, mit denen Zugriffsberechtigungen vergeben oder erbracht werden.

Für die verschiedenen Datenbanksysteme gibt es folgende DRIVE-Anweisungen:

Datenbank-system	**Anweisung**	**Funktion**
SESAM V1	PERMIT	Angabe von Benutzeridentifikationen, die beim Zugriff auf das angegebene Schema oder die angegebene SESAM-Datenbank überprüft werden; hat Auswirkungen auf New-Style und Old-Style
SESAM V2	PERMIT	Angabe von Benutzeridentifikationen, die beim Old-Style-Zugriff auf CALL-DML-Tabellen überprüft werden; hat nur Auswirkungen auf Old-Style
SESAM V2	GRANT	Vergabe von Zugriffsrechten für Datenbanken, Basistabellen oder Views sowie für Storage Groups; hat nur Auswirkungen auf New-Style
SESAM V2	SET SESSION AUTHORIZATION	Vergabe einer Zugangsberechtigung zu einer Datenbank; hat nur Auswirkungen auf New-Style
UDS	PERMIT	Angabe von Benutzeridentifikationen, die beim Zugriff auf das angegebene SQL-Schema der UDS-Datenbank überprüft werden

Die Transaktionssicherung, die auch den Zugriffsschutz berührt, ist im Kapitel „Transaktionskonzept“ auf Seite 261 ausführlich beschrieben.

10 Transaktionskonzept

Dieses Kapitel beschreibt:

- die Definition und Steuerung einer Transaktion (ab Seite 262)
- die Transaktionssicherung (ab Seite 272)
- die Auswirkungen der Transaktionssteuerung auf Definitionen (z.B. Views oder Variablen) (ab Seite 281)
- Programmierhinweise und -regeln zum Transaktionsbetrieb (ab Seite 292).

10.1 Definition und Steuerung einer Transaktion

Eine (SQL-)Transaktion wird definiert als eine logische Folge von SQL-Anweisungen, die als eine **geschlossene** und **unteilbare** Einheit behandelt wird. SQL-Transaktionen sichern und erhalten konsistente Datenbestände der Datenbank, indem logisch zusammenhängende SQL-Anweisungen, die Daten der Datenbank verändern, in einer Transaktion zusammengefaßt werden. Treten während einer Transaktion unvorhergesehene Fehler auf, kann durch das anwender- oder systemseitige Rücksetzen einer Transaktion die Inkonsistenz der Daten wieder behoben werden. Dabei gilt das Alles-oder-Nichts-Prinzip, d.h. es werden entweder alle innerhalb einer Transaktion angestrebten Aktionen oder keine durchgeführt.

Der konsistente Zustand einer Datenbank bedeutet vor allem Daten- und Benutzerintegrität durch Einhalten aller vereinbarten Constraints und Privilegien. Wichtige Constraints sind

– „entity integrity constraint“, d.h. jeder Satz kann tabellenweit eindeutig über einen Primärschlüssel identifiziert werden

– „semantic integrity constraint“, d.h. für alle Attribute werden tabellenweit die zulässigen Werte vorgeschrieben (vor allem Datentyp, Standardvorbelegung und Wertemenge)

– „referential integrity constraint“, d.h. Tabellen werden durch Fremdschlüssel verknüpft, die dann bevorzugte, weil optimierbare Joinattribute darstellen.

Konkurrierende Zugriffe auf Sätze innerhalb paralleler Transaktionen werden vom Datenbanksystem koordiniert und ggf. vom Transaktionsmonitor, nicht aber von DRIVE/WINDOWS.

In DRIVE-Anwendungen greifen SQL-Transaktionen nicht nur lesend oder schreibend auf Daten von Datenbanken zu, sondern auch auf Daten des DRIVE-Programms, indem die SQL-Anweisungen DRIVE-Variablen referenzieren. In Bildschirmformate eingelesene Daten werden in DRIVE-Variablen übertragen und anschließend über SQL-Anweisungen (INSERT, UPDATE) in Datenbanken. Umgekehrt werden Daten aus Datenbanken (SELECT, FETCH) in DRIVE-Variablen eingelesen und dann weitergeleitet, z.B. in Bildschirmformaten ausgegeben.

Erweitert man also den Begriff der SQL-Transaktion als logische Folge von SQL-Anweisungen auf eine Folge von SQL- und DRIVE-Anweisungen, so ergibt sich die DRIVE-Transaktion. Jede SQL-Transaktion läuft also in einer DRIVE-Transaktion ab. Umgekehrt kann in einer offenen DRIVE-Transaktion höchstens eine SQL-Transaktion offen sein. In einer DRIVE-Transaktion werden neben Daten auch DRIVE-Objekte manipuliert (Bildschirm- und Listenformate). Eine DRIVE-Transaktion ist entweder eine reine New-Style-Transaktion oder eine reine Old-Style-Transaktion, weil beim Wechsel vom New-Style in den Old-Style und zurück keine Transaktion offen sein darf.

Das jeweilige Datenbanksystem garantiert in der SQL-Transaktion die Konsistenz von Daten der Datenbank. DRIVE/WINDOWS garantiert in der DRIVE-Transaktion die Konsistenz von Daten und Objekten von DRIVE-Programmen. Parallele DRIVE-Transaktionen konkurrieren nur um Daten in Datenbanken, nicht um DRIVE-Daten oder -Objekte, da jedes DRIVE-Programm seine eigene Umgebung hat.

Den Beginn einer Transaktion definieren Sie nicht mit einer bestimmten SQL-Anweisung (DRIVE/WINDOWS kennt kein BEGIN). Die erste SQL-Anweisung nach dem Rücksetzen oder Festschreiben der vorherigen Transaktion oder dem Programmstart wird in der Regel vom Datenbanksystem als Transaktionsbeginn gewertet (Ausnahmen siehe Seite 263).

Bei der Programmierung von Transaktionen ist zu beachten, daß diese wirklich unteilbar sind. Umfangreiche Transaktionen können unter ungünstigen Umständen ganze Datenbanken für den Zugriff von Transaktionen anderer Programmierer sperren.

10.1.1 Übersicht der Anweisungen

Folgende Anweisungen gibt es für die Steuerung von Transaktionen:

SQL-Anweisung	**Datenbanksysteme**	**Beschreibung**
SET TRANSACTION	SESAM V1 u. V2, UDS	Transaktionseigenschaften festlegen (TA-Level u. TA-Modus)
COMMIT WORK	alle DB-Systeme	Transaktion beenden u. neuen Sicherungspunkt einrichten
ROLLBACK WORK [WITH RESET]	alle DB-Systeme	Transaktion beenden u. auf den letzten Sicherungspunkt zurücksetzen

10.1.2 Beginn einer Transaktion

Den Beginn einer Transaktion legen Sie nicht mit einer bestimmten SQL-Anweisung fest. Eine Transaktion (TA) beginnt im transaktionslosen Zustand mit einer TA-initiierenden SQL-Anweisung. Das sind alle SQL-Anweisungen mit **Ausnahme** der folgenden:

SQL-Anweisung	**Datenbanksysteme**
COMMIT WORK ROLLBACK WORK	alle DB-Systeme
SET TRANSACTION PERMIT	SESAM V1 u. V2, UDS

SQL-Anweisung	**Datenbanksysteme**
SET SESSION SET CATALOG SET SCHEMA	SESAM V2 } nur für dynamische Anweisungen mit SESAM V2
nur in TA-losem Zustand erlaubt	**Datenbanksysteme**
SET TRANSACTION PERMIT	SESAM V1 u. V2, UDS
SET SESSION	SESAM V2

Wird eine laufende Transaktion vom Datenbanksystem zurückgesetzt, beginnt analog mit der nächsten TA-initiierenden SQL-Anweisung eine neue Transaktion.

Arbeiten Sie mit UTM, enthält jede UTM-Transaktion entweder keine oder genau eine SQL-Transaktion. Unter UTM beginnt daher eine SQL-Transaktion mit der ersten TA-initiierenden SQL-Anweisung einer UTM-Transaktion.

Spezielle Objekte und Anweisungen, mit denen diese manipuliert werden, unterliegen nicht der DRIVE-Transaktionssicherung:

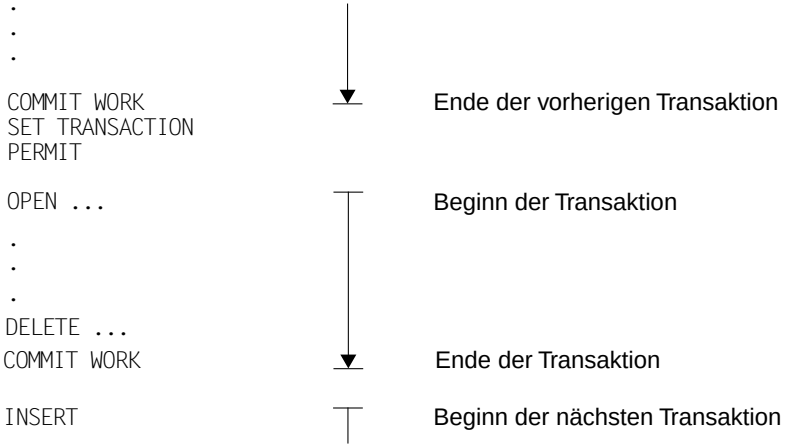
– Dateien und File-Anweisungen

– Reports und Report-Anweisungen

Im Debugger vorgenommene Einstellungen, z.B. Haltepunkte setzen, Aktionen hinterlegen etc., unterliegen nicht der Transaktionssicherung.

Beispiel

Beginn einer Transaktion je nach SQL-Anweisung (SESAM V1 und V2, UDS)



10.1.3 Transaktionseigenschaften festlegen

Zu den Eigenschaften einer Transaktion gehören u.a. der Transaktionsmodus und der Isolations- oder Konsistenzlevel. Der Transaktionsmodus legt fest, ob eine Transaktion nur lesend oder auch schreibend auf Daten und Metadaten zugreifen darf. Bei Transaktionsende werden alle Sperren freigegeben. Bei Cursorverarbeitung werden ggf. bei FETCH der vorher gelesene Satz und bei CLOSE alle gelesenen Sätze entsperrt. Der Isolations- oder Konsistenzlevel bestimmt den Einfluß von SQL-Anweisungen einer Transaktion auf konkurrierende, d.h. teilweise gleichzeitig ablaufende Transaktionen: er gibt an, wie stark das Lesen von Sätzen in der Transaktion durch gleichzeitige Schreibzugriffe einer konkurrierenden Transaktion beeinflußt wird.

Bei gleichzeitigem Zugriff zweier Transaktionen T1 und T2 auf einen oder mehrere Sätze können die folgenden Phänomene auftreten, falls nicht genügend Sperren gesetzt oder gesetzte Sperren nicht genügend beachtet werden.

P1: dirty read

T1 liest einen Satz, der von T2 geändert, aber noch nicht freigegeben wurde. Wird T2 zurückgesetzt, hat T1 einen nicht existierenden Satz gelesen.

Ursache: T1 berücksichtigt nicht, daß T2 einen Satz noch gesperrt hat.

P2: non-repeatable read

T1 liest einen Satz, der anschließend von T2 geändert wird. Nachdem T2 die Änderung festschreibt, kann T1 auf den Satz in seinem ursprünglichen Zustand nicht mehr zugreifen.

Ursache: T1 sperrt nach dem Lesen den Satz nicht, so daß der Satz von T2 geändert werden kann.

P3: phantoms

T1 liest alle Sätze, die über eine SELECT-Klausel ausgewählt wurden. Anschließend werden von T2 Sätze aufgenommen, die ebenfalls der mit dieser SELECT-Klausel vereinbarten Bedingung genügen. Wiederholt T1 den Lesevorgang mit derselben SELECT-Bedingung, ist die Treffermenge größer.

Ursache: T1 hat nach dem Lesen die Sätze nicht gesperrt, so daß von T2 Sätze eingefügt werden konnten.

P4: lost update

T1 ändert einen Satz. Anschließend ändert T2 denselben Satz. Wird T1 zurückgesetzt, geht auch die Änderung von T2 verloren.

Ursache: T1 und T2 sperren sich gegenseitig nicht. Deshalb wird nach einem Zurücksetzen der Transaktion auf den Stand vor Transaktionsbeginn zurückgesetzt.

Die Phänomene P1, P2 und P3 betreffen lesende Transaktionen, die mit schreibenden Transaktionen konkurrieren. Das Phänomen P4 kann bei keinem der von DRIVE/WINDOWS unterstützten Datenbanksysteme auftreten, weil ein geänderter Satz bis Transaktionsende gesperrt bleibt.

Welche der Phänomene P1, P2, P3 auftreten können, hängt vom eingestellten oder standardmäßig wirksamen Transaktionslevel der beteiligten Transaktionen ab. Je höher die Level der konkurrierenden Transaktionen sind, desto mehr sind sie voneinander isoliert und desto weniger Phänomene können auftreten. Es werden aber auch desto mehr Sperren gesetzt und beachtet, d.h. die Transaktionsparallelität ist geringer. Bei allen Transaktionsleveln ist das Alles-oder Nichts-Prinzip gültig (s.o.).

SESAM V1

Mit der Anweisung SET TRANSACTION können die Konsistenzlevel 0, 1, 2 und 3 (Standardwert) für jeweils eine Transaktion eingestellt werden. Je nach Konsistenzlevel werden folgende Phänomene ausgeschlossen und beim Lesen Sperren gesetzt und beachtet:

Konsistenzlevel	**mögliche Phänomene**	**TA setzt Lesesperre**	**TA beachtet Lesesperre**	**TA setzt Schreibsperre**
0	P1, P2, P3	nein	nein	nein
1	P1, P2[1], P3	nein	nein	ja
2	P2, P3	nein	ja	nein
3	P3	nein	ja	ja

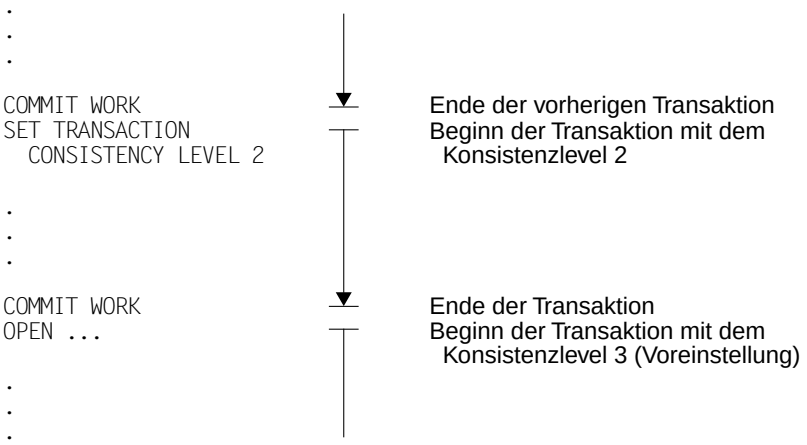
[1] Bei Konsistenzlevel 1 kann P2 nur für Sätze auftreten, die zuvor mit „dirty read“ gelesen wurden.

Mit SET TRANSACTION kann außerdem der Transaktionsstatus READ ONLY (nur lesend) oder READ WRITE (auch schreibend) festgelegt werden. Standardwert ist READ ONLY bei Konsistenzlevel 0 oder 1 und READ WRITE bei Konsistenzlevel 2 oder 3.

Wollen Sie mit der SET TRANSACTION-Anweisung den Konsistenzlevel festlegen, muß diese Anweisung die erste von PERMIT verschiedene Anweisung der Transaktion sein.

Beispiel

Wechsel von standardmäßigem und mit der Anweisung SET TRANSACTION eingestelltem Konsistenzlevel.



SESAM V2

Mit der Anweisung SET SESSION AUTHORIZATION können Sie den SQL-User für alle dynamischen SQL-Anweisungen der Transaktion festlegen. Die SQL-User für die statischen SQL-Anweisungen der Transaktion werden bereits zum Übersetzungszeitpunkt festgelegt (siehe DRIVE-SQL-Lexikon [5]).

Mit der Anweisung SET TRANSACTION können die Konsistenzlevel 0, 1, 2, 3 und 4 (Standardwert) oder die vier normkonformen Isolationslevel für die aktuell zu beginnende Transaktion eingestellt werden. Je nach Level werden folgende Phänomene ausgeschlossen und beim Lesen Sperren gesetzt und beachtet:

Konsistenz-level	**Isolationslevel**	**mögliche Phänomene**	**TA setzt Lesesperre**	**TA beachtet Lesesperre**	**TA setzt Schreisperre**
0	READ UNCOMMITTED	P1, P2, P3	nein	nein	nein
1	-	P1, P2[1], P3	nein	nein	ja
2	READ COMMITTED	P2, P3	nein	ja	nein
3	REPEATABLE READ	P3	nein	ja	ja
4	SERIALIZABLE	-	ja	ja	ja

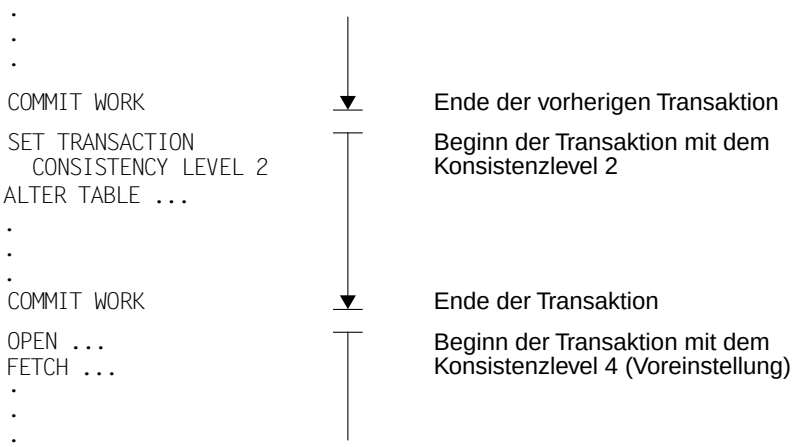
[1] Bei Konsistenzlevel 1 kann P2 nur für Sätze auftreten, die zuvor mit „dirty read“ gelesen wurden.

Mit SET TRANSACTION kann außerdem der Transaktionsmodus READ ONLY (nur lesend) oder READ WRITE (auch schreibend) festgelegt werden. Standardwert ist READ ONLY bei Konsistenzlevel 0 oder 1 und READ WRITE bei Konsistenzlevel 2, 3 oder 4.

SQL-Anweisungen, die auf Metadaten zugreifen, sog. DDL-Anweisungen (Data Description Language), z.B. ALTER TABLE, und SQL-Anweisungen, die auf Nutzdaten zugreifen, sog. DML-Anweisungen (Data Manipulation Language), z.B. UPDATE, sind **nicht** in derselben Transaktion erlaubt. Dies gilt für alle Transaktionslevel.

Beispiel

Wechsel von standardmäßigem und mit der Anweisung SET TRANSACTION eingestelltem Konsistenzlevel.



Bei Transaktionsende wird wieder die Voreinstellung für den Konsistenz- oder Isolationslevel (4 oder SERIALIZABLE oder gemäß Festlegung in der Konfigurationsdatei) gültig.

UDS

Mit der Anweisung SET TRANSACTION können die Konsistenzlevel 2 und 3 (Standardwert) eingestellt werden. Je nach Level werden folgende Phänomene ausgeschlossen und beim Lesen Sperren gesetzt und beachtet:

Konsistenz-level	mögliche Phänomene	TA setzt Lesesperre	TA beachtet Lesesperre	TA setzt Schreibsperre
2	P2, P3	ja	ja	nein
3	P3	nein	ja	ja

Sätze können also nie mit „dirty read“ gelesen werden.

Ein Transaktionsmodus kann nicht eingestellt werden. Jede Transaktion kann lesend und auch schreibend auf Daten zugreifen.

10.1.4 Transaktion beenden mit COMMIT WORK

Mit der Anweisung COMMIT WORK wird die Transaktion beendet und ein neuer Sicherungspunkt geschrieben. Alle Änderungen in den Basistabellen seit dem Beginn der Transaktion werden in der Datenbank festgeschrieben. Im Programm-Modus kann das Beenden einer Transaktion durch COMMIT WORK gekoppelt werden mit einer DISPLAY-, SEND MESSAGE- oder STOP-Anweisung (siehe „DRIVE-SQL-Lexika“ [4], [5], [6],).

10.1.5 Transaktion zurücksetzen mit ROLLBACK WORK

Mit der Anweisung ROLLBACK WORK wird eine Transaktion zurückgesetzt, d.h. alle Änderungen in den Basistabellen seit dem Beginn der Transaktion werden rückgängig gemacht. Daten und Metadaten der Datenbank haben den Stand des letzten Sicherungspunktes (COMMIT WORK), weil geänderte Sätze bis Transaktionsende gesperrt bleiben. Das DRIVE-Programm wird mit der auf ROLLBACK WORK folgenden Anweisung fortgesetzt.

Im Programm-Modus kann das Rücksetzen einer Transaktion mit ROLLBACK WORK mit der Klausel WITH RESET gekoppelt werden. In diesem Fall wird nicht nur die SQL-Transaktion, sondern auch die DRIVE-Transaktion zurückgesetzt, d.h. Inhalte von DRIVE-Variablen, Bildschirm- und Listenformaten, Fenster und Inhalte der grafischen Benutzeroberfläche. Das DRIVE-Programm wird mit der Anweisung fortgesetzt, die auf das letzte COMMIT WORK folgt.

Wird das Durchlaufen der Anweisung ROLLBACK WORK WITH RESET nicht von einer Bedingung abhängig gemacht (IF oder CASE), besteht die Gefahr einer Endlosschleife, da das DRIVE-Programm mit der Anweisung fortgesetzt wird, die auf die letzte COMMIT WORK-Anweisung folgt.

Beispiel 1

```
COMMIT WORK;
...
ROLLBACK WORK WITH RESET;
```

} Endlosschleife

Beispiel 2

```
COMMIT WORK;
...
IF bedingung
THEN ROLLBACK WORK WITH RESET
ELSE ROLLBACK WORK;
END IF;
```

keine Endlosschleife, wenn bedingung irgendwann FALSE ist

Beispiel 3

In diesem Beispiel wird das Verhalten von ROLLBACK WORK ohne WITH RESET dem mit WITH RESET gegenübergestellt.

	Wert von feld	Wert von &V
`SET &V = '1'`		1
`UPDATE t1 SET feld = &V WHERE ...`	1	
`COMMIT WORK;`		
`SET &V = '2'`		2
`UPDATE t1 SET feld = &V WHERE ...`	2	
`IF bedingung`		
`THEN ROLLBACK WORK;`	1	2
`ELSE ROLLBACK WORK WITH RESET;`	1	1
`END IF;`		

10.1.6 Transaktionsabbruch

Ein Transaktionsabbruch, d.h. ein systemseitiges kontrolliertes Rücksetzen kann bei SESAM V1 und V2 und UDS auch durch das Datenbanksystem oder den Datenbank-Administrator erfolgen, z.B. um Deadlocks oder Longlocks zu vermeiden. Dies wird mit einem SQLCODE der Form -1xxx oder einem SQLSTATE der Form 40xxx gemeldet. DRIVE/WINDOWS reagiert wie bei der Anweisung ROLLBACK WORK WITH RESET.

Vor dem Verlassen eines mit DO (oder DEBUG) aufgerufenen Programms durch END PROCEDURE, BREAK, STOP oder Folge-DO muß ein COMMIT WORK oder ein ROLLBACK WORK durchlaufen worden sein, sonst erfolgt Programmabbruch mit Rücksetzen der Transaktion (siehe auch Seite 277).

Wenn am Ende einer DRIVE-Sitzung (STOP) die Transaktion nicht beendet ist, meldet DRIVE/WINDOWS einen Fehler, ohne intern eine Anweisung an das Datenbanksystem zu schicken. DRIVE/WINDOWS wird nicht beendet.

10.1.7 DRIVE-Anweisungen im transaktionslosen Zustand

Folgende DRIVE-Anweisungen können nur im transaktionslosen Zustand ausgeführt werden:

- ein SESAM V2-Programm implizit vor der Ausführung übersetzen (Anweisung CALL)
- ein Old-Style-Programm aus einem New-Style-Programm heraus aufrufen (Anweisung CALL)
- im Dialog-Modus:
  - eine Übersetzungsphase mit der Anweisung COMPILE im Dialog-Modus starten
  - ein Programm ausführen mit der Anweisung DO oder testen mit der Anweisung DEBUG im Dialog-Modus

Bei lokalem und Remote-Zugriff kann in offener DRIVE-Transaktion nur eine Datenbank-Transaktion offen sein.

Bei verteilten UTM-Anwendungen (VTV) kann zu jeder lokalen DRIVE-Anwendung, die sich nicht in transaktionslosem Zustand befindet, nur eine UTM-Transaktion und nur eine Datenbank-Transaktion offen sein (zur Definition siehe Abschnitt „Definition und Steuerung einer Transaktion“ auf Seite 262). Über UTM-D werden alle beteiligten lokalen (DRIVE- und SQL-)Transaktionen zu einer verteilten Transaktion synchronisiert.

Bei verteilten Anwendungen gilt für den DRIVE-Client (Präsentationsumgebung) wie im lokalen Betrieb, daß in offener DRIVE-Transaktion nur eine SQL-Transaktion offen sein kann. Dies gilt auch für einen DRIVE-Server (Verarbeitungsumgebung), falls er eine lokale DRIVE-UTM-Anwendung ist. Ist der DRIVE-Server eine DRIVE-VTV-Anwendung, gilt das im letzten Absatz zu VTV Beschriebene.

10.2 Transaktionssicherung

Dieser Abschnitt beschreibt die lokale Transaktionssicherung. Zur Transaktionssicherung in verteilten Anwendungen siehe Kapitel „Verteilte Anwendungen“ auf Seite 299, zur Transaktionssicherung bei VTV siehe Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339.

Der Beginn einer DRIVE-Sitzung oder eines UTM-Vorgangs, jede COMMIT WORK-Anweisung und jede ROLLBACK WORK-Anweisung stellen einen Synchronisationspunkt für die Datenbanken dar, die über den zugeordneten DBH angeschlossen sind. Sessionbeginn und Vorgangsbeginn sowie die COMMIT WORK-Anweisung stellen außerdem einen Sicherungspunkt dar. Zwischen zwei Synchronisationspunkten liegt eine SQL-Transaktion, die eine Folge von SQL-Anweisungen für den Datenbankzugriff zusammenfaßt. Die Synchronisierung bei COMMIT WORK besteht darin, einen neuen Sicherungspunkt zu vereinbaren, die bei ROLLBACK WORK darin, auf den letzten Sicherungspunkt zurückzusetzen. Da die Transaktion eine unteilbare Einheit ist, werden entweder alle oder keine ihrer Anweisungen zurückgesetzt. Wird eine Transaktion auf den letzten Sicherungspunkt zurückgesetzt, wirkt keine ihrer Anweisungen auf die jeweils referenzierte Datenbank.

Im UTM-Betrieb synchronisiert DRIVE/WINDOWS auf Vorgangsebene UTM-Transaktionen, SQL-Transaktionen und DRIVE-Transaktionen. Das jeweilige Datenbanksystem und nicht DRIVE/WINDOWS koordiniert konkurrierende Zugriffe mehrerer SQL-Transaktionen auf denselben Datenbestand. Dazu wird für jede Transaktion implizit oder mit der Anweisung SET TRANSACTION das Sperrverhalten gegenüber konkurrierenden Transaktionen festgelegt.

DRIVE-Anwendungen werden immer transaktionsgesichert ausgeführt. Bei Anwendungsende darf keine SQL-Transaktion mehr offen sein.

Dies bedeutet im Extremfall, daß in einer Anwendung mit Datenbankzugriffen, in der weder ein COMMIT WORK noch ein ROLLBACK WORK durchlaufen wird (oder gar nicht programmiert ist), alle Datenbankzugriffe zurückgesetzt werden.

DRIVE/WINDOWS garantiert Transaktionssicherung des Datenbestands im Normalfall, d.h. bei ordnungsgemäßem Ende oder bei fehlerhaftem Ende von Anwendungen unter der Kontrolle von DRIVE/WINDOWS (interner Fehler). Bei fehlerhaftem Ende von Anwendungen, bei dem DRIVE/WINDOWS nicht die Kontrolle zurückerhält (externer Fehler, Systemzusammenbruch), garantiert das jeweilige Datenbanksystem die Konsistenz des Datenbestands. Die Transaktion ist daher bei DRIVE/WINDOWS die Einheit für isolierte, konkurrierende SQL-Zugriffe, die stets konsistente Zustandswechsel von Datenbanken garantiert. Erst bei Transaktionsende entscheidet sich, ob entweder alle SQL-Zugriffe oder kein SQL-Zugriff Wirkungen in den angesprochenen Datenbanken hinterläßt.

Beim Datenbanksystem SESAM V2 ist zu beachten, daß es Transaktionen des SESAM/SQL-Laufzeitsystems gibt, die außerhalb von DBH-Transaktionen ablaufen, nämlich entweder unabhängig von DBH-Transaktionen oder vor der ersten TA-initiierenden SQL-Anweisung. Diese Transaktionen des Laufzeitsystems unterliegen auch der Transaktionssicherung. D.h. die Wirkung einer nicht-TA-initiierenden SQL-Anweisung kann mit COMMIT WORK festgeschrieben und mit ROLLBACK WORK zurückgesetzt werden.

Die Transaktion ist bei DRIVE/WINDOWS auch die Einheit für den Wiederanlauf, d.h. für die Herstellung eines konsistenten Zustands innerhalb und zwischen Anwendungsdaten und Datenbankdaten nach einem Fehler. Ein **interner Wiederanlauf** tritt ein nach Rücksetzen der Transaktion durch das Datenbanksystem oder den Datenbank-Administrator (siehe Seite 270) oder durch Ausführung der folgenden DRIVE-Anweisungen: ROLLBACK WORK und ROLLBACK WORK WITH RESET. Während bei ROLLBACK WORK nur ein Transaktionswiederanlauf stattfindet (konsistente Datenbankdaten), finden ein Transaktionswiederanlauf und ein Anwendungswiederanlauf statt (konsistente Anwendungsdaten und Konsistenz zwischen Datenbanken und Anwendung) bei ROLLBACK WORK WITH RESET sowie bei Transaktionsabbruch (zu Besonderheiten bei Verteilter Verarbeitung siehe Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339 und Kapitel „Verteilte Anwendungen“ auf Seite 299).

Die BS2000-Datenbanksysteme geben beim internen Rücksetzen die Meldung TA_CANCELLED aus. Ein **externer Wiederanlauf** ist im UTM-Betrieb möglich und tritt beim Hochfahren der UTM-Anwendung nach einem Systemzusammenbruch ein.

Nach dem internen oder externen Wiederanlauf kann die DRIVE-Anwendung am letzten Sicherungspunkt fortgesetzt werden. Dieser bezieht sich immer auf SQL-Transaktionen, bei UTM-Betrieb auch auf UTM-Transaktionen und (außer beim internen Wiederanlauf nach ROLLBACK WORK ohne RESET) auch auf DRIVE-Transaktionen.

10.2.1 Transaktionssicherung im TIAM-Betrieb

Die Transaktionssicherung wirkt in Verbindung mit SESAM V1, SESAM V2 und UDS-Datenbanken und wirkt lokal sowie remote.

Für die Transaktionssicherung im Dialog- und Programm-Modus gilt:

– Beim Rücksetzen der Transaktion durch die Datenbank oder explizitem Rücksetzen mit der SQL-Anweisung ROLLBACK WORK werden die Daten auf den letzten Sicherungspunkt zurückgesetzt.

– Beim Remote-Zugriff auf SESAM V1 und V2- und UDS-Datenbanken wird von DRIVE/WINDOWS über DRIVE/WINDOWS-NET eine Transaktionssynchronisation zwischen der DRIVE/WINDOWS-Transaktionssicherung und der Datenbanksystem-Transaktionssicherung (BS2000) gewährleistet.

Wiederanlauf

DRIVE/WINDOWS bietet den Wiederanlauf im Programm-Modus für SESAM V1- und V2- und UDS- Datenbanken an. Es unterstützt einenTransaktionswiederanlauf (SQL-Transaktionen) und einen Anwendungswiederanlauf (SQL- und DRIVE-Transaktionen).

Zwischen lokalem und Remote-Zugriff besteht kein Unterschied. Es gelten folgende Regeln:

– Setzt das Datenbanksystem eine Transaktion zurück oder wird ein ROLLBACK WORK WITH RESET durchgeführt, so setzt DRIVE/WINDOWS auf die Anweisung nach dem letzten COMMIT WORK (≡ Transaktionsende) zurück. Die Daten im DRIVE-Programm werden ebenfalls auf den Stand vom letzten Transaktionsende zurückgesetzt. Falls das Programm kein vorausgehendes COMMIT enthält, wird es abgebrochen.

– Wird ein ROLLBACK WORK durchgeführt, setzt DRIVE/WINDOWS die Transaktion beim Datenbanksystem zurück. Es wird mit der auf ROLLBACK WORK folgenden Anweisung fortgesetzt. Die Daten im DRIVE-Programm werden dabei nicht auf den Stand vom letzten Transaktionsende zurückgesetzt.

DRIVE/WINDOWS bietet im TIAM-Betrieb keinen externen Wiederanlauf an.

10.2.2 Transaktionssicherung im UTM-Betrieb

Eine Besonderheit im UTM-Betrieb ist die von DRIVE/WINDOWS zur Verfügung gestellte Synchronisation von SQL-Transaktionen der Datenbanksysteme SESAM V1 und V2 und UDS mit UTM-Transaktionen. Um Konsistenz zu gewährleisten, müssen UTM und das zugeordnete Datenbanksystem ihre Transaktionen gemeinsam beenden. Eine UTM-Transaktion enthält daher entweder keine oder genau eine (vollständige) SQL-Transaktion.

In der folgenden Beschreibung werden für Programmierer, die DRIVE-Programme und C/COBOL-Programme koppeln in Klammern die KDCS-Aufrufe angegeben, die DRIVE intern umsetzt und an UTM schickt. Für die reine DRIVE-Programmierung brauchen Sie keine Kenntnisse der UTM-Aufrufe.

Eine SQL-Transaktion wird beendet (COMMIT WORK) oder zurückgesetzt (ROLLBACK WORK), indem die zugehörige UTM-Transaktion beendet (PEND SP/RE/FI) oder zurückgesetzt (RSET oder PEND RS) wird. Dabei gilt folgende Abbildung:

– PEND FI nur bei STOP

– RSET bei ROLLBACK WORK sowie bei ROLLBACK WORK WITH RESET, wenn kein Remote-Zugriff auf eine BS2000-Datenbank und keine verteilte UTM-Anwendung (Verteilte Transaktionsverarbeitung VTV) stattfinden

– PEND RS bei ROLLBACK WORK WITH RESET im Betrieb mit Remote-Zugriff auf eine BS2000-Datenbank oder VTV.

Die Anweisung STOP beendet neben dem laufenden Dialogschritt auch den gesamten UTM-Vorgang (PEND FI).

Bis UTM (BS2000) V3.2 war mit dem ordnungsgemäßen Ende einer UTM-Transaktion immer auch der Dialogschritt zu Ende (PEND RE), so daß in DRIVE V6.x und DRIVE/WINDOWS V1.0 die COMMIT WORK-Anweisung mit einer Bildschirmausgabe gekoppelt war und der Anwender um eine Quittung gebeten wurde (DRIVE-Meldung in der Meldungszeile der Alpha-Oberfläche). Seit UTM (BS2000) V3.3 kann jedoch eine UTM-Transaktion beendet und der Dialogschritt fortgesetzt werden (PEND SP). Ab DRIVE/WINDOWS V1.1, also auch in DRIVE/WINDOWS V2, ist die einfache Anweisung COMMIT WORK auch im UTM-Betrieb nicht mehr mit einer Bildschirmausgabe gekoppelt. Eine SQL-Transaktion braucht also nun nicht mehr mindestens einen Dialogschritt zu umfassen.

COMMIT WORK und ROLLBACK WORK sind als einzige SQL-Anweisungen auch in DRIVE-Programmen erlaubt, die keine OPTION-Anweisung mit dem Operanden DBSYSTEM enthalten, also keine Datenbankprogramme sind. In diesem Fall wirken die Anweisungen COMMIT WORK und ROLLBACK WORK auf das Datenbanksystem, das im Rahmen der aktuellen CALL-Programmhierarchie zuletzt angesprochen wurde oder, falls noch kein Datenbanksystem angesprochen wurde, nur auf UTM- und DRIVE-Transaktionen. Ist die OPTION DBSYSTEM mit dem Wert OFF belegt, wirken die Anweisungen COMMIT WORK und ROLLBACK WORK auf UTM- und DRIVE-Transaktionen. Ist ein anderer Wert für OPTION DBSYSTEM angegeben, wirken diese beiden Anweisungen auf UTM-, DRIVE- und SQL-Transaktionen.

10.2.2.1 Dialog-Modus

Im Dialog-Modus können Sie auf das Datenbanksystem zugreifen, das der geladenen Variante entspricht.

DRIVE/WINDOWS bettet jede SQL- und jede DRIVE-Anweisung in einen UTM-Dialogschritt ein. Die Anweisung COMMIT WORK beendet sowohl den UTM-Dialogschritt als auch die SQL- und UTM-Transaktion. Es können sowohl UTM-Einschritt- als auch UTM-Mehrschritt-Transaktionen beendet werden.

In der folgenden Beschreibung werden in Klammern die KDCS-Aufrufe angegeben, die DRIVE intern umsetzt und an UTM schickt. Für die reine DRIVE-Programmierung brauchen Sie keine Kenntnisse der UTM-Aufrufe.

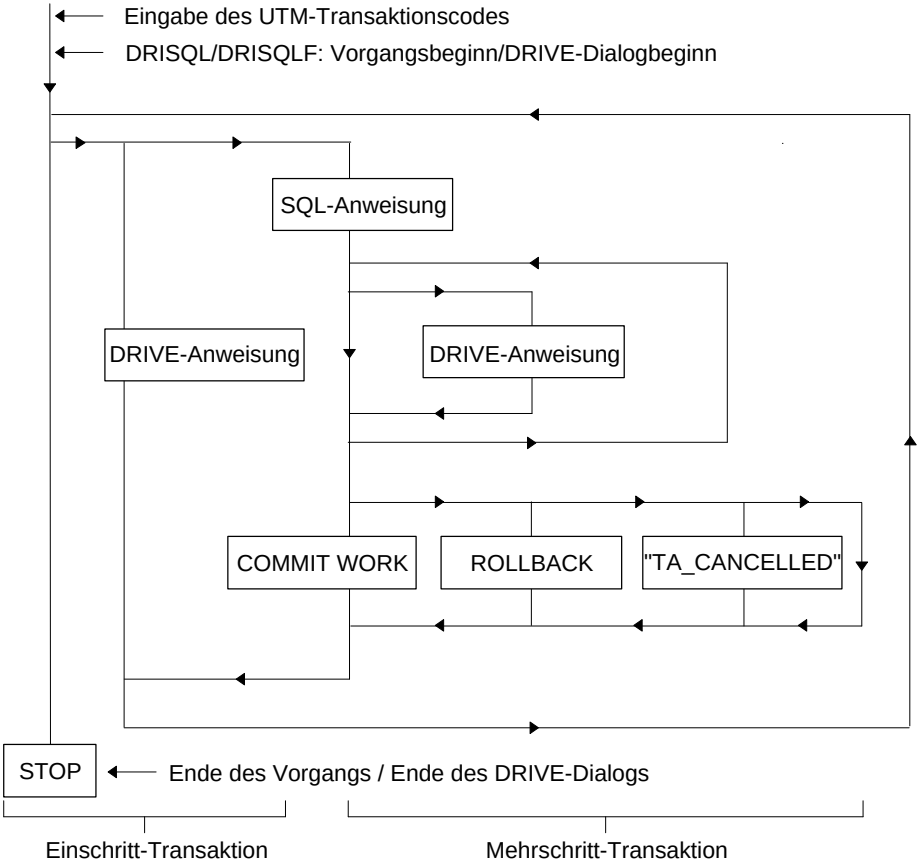
Einschritt-Transaktionen sind reine DRIVE-Anweisungen (keine SQL-Anweisungen) oder SQL-Anweisungen, die im transaktionslosen Zustand ablaufen (sie werden mit PEND RE abgeschlossen). **Mehrschritt-Transaktionen** beginnen im transaktionslosen Zustand mit einer TA-initiierenden SQL-Anweisung (siehe Seite 263), die den Dialogschritt beendet,

aber nicht die UTM-Transaktion (sie werden mit PEND KP abgeschlossen). Jede folgende DRIVE-Anweisung außer COMMIT WORK und ROLLBACK WORK beendet ebenfalls einen Dialogschritt, aber nicht die Transaktion.

Die Anweisung STOP beendet neben dem laufenden Dialogschritt auch den gesamten UTM-Vorgang. DRIVE/WINDOWS unterstützt die Synchronisation der Datenbank-Transaktion mit den UTM-Transaktionen.

Die folgende Grafik stellt die möglichen Abläufe von transaktionsgesicherten SESAM V1 und V2- und UDS-UTM-Zugriffen im Dialog-Modus dar

.



Beim internen Zurücksetzen einer Transaktion durch das Datenbanksystem (TA_CANCELLED) werden die SQL-, die DRIVE- und die UTM-Transaktion synchronisiert zurückgesetzt (RSET-Aufruf an UTM).

– Beim Rücksetzen der Transaktion mit der SQL-Anweisung ROLLBACK WORK werden die Daten in der Datenbank auf den letzten Sicherungspunkt zurückgesetzt. Danach ist die nächste Dialog-Anweisung möglich.

– Beim Rücksetzen der Transaktion mit der Anweisung ROLLBACK WITH RESET wird auf den Stand des letzten COMMIT WORK zurückgesetzt. Danach ist die nächste Dialog-Anweisung möglich.

10.2.2.2 Programm-Modus

Der lokale BS2000-Zugriff wird bereits bei der Installation gewählt, eventuelle OPTION-Angaben werden ignoriert.

Während DRIVE/WINDOWS im Dialog-Modus jede Anweisung in einen UTM-Dialog-Schritt einbettet, muß innerhalb von DRIVE-Programmen zwischen den einzelnen Anweisungen differenziert werden:

– Anweisungen, die eine Bildschirmausgabe bewirken, beenden einen UTM-Dialogschritt. Es handelt sich dabei um folgende Anweisungen:

```
BREAK
COMMIT WORK WITH DISPLAY / SEND MESSAGE / STOP
DISPLAY FORM
DISPLAY SCREEN
END PROCEDURE (beim Übergang von Programm- in Dialog-Modus)
FILL FORM (bei Bildschirm-Überlauf)
SEND MESSAGE
```

– Bei auftretenden Fehlern zum Analyse- oder Ablaufzeitpunkt des Programms wird ein UTM-Dialogschritt beendet, und es erfolgt eine Bildschirmausgabe.

– Reine Verarbeitungs-Anweisungen führen nicht zu Bildschirmausgaben und damit nicht zur Beendigung eines UTM-Dialogschrittes.

Regeln für die Synchronisation der Transaktionen in DRIVE-Programmen

In der folgenden Beschreibung werden für Programmierer, die DRIVE-Programme und C/COBOL-Programme koppeln in Klammern die KDCS-Aufrufe angegeben, die DRIVE intern umsetzt und an UTM schickt. Für die reine DRIVE-Programmierung brauchen Sie keine Kenntnisse der UTM-Aufrufe.

– Die Anweisungen DISPLAY FORM, DISPLAY SCREEN, SEND MESSAGE, FILL FORM bei Bildschirmüberlauf:

  Es wird der Dialogschritt beendet, aber nicht die UTM-Transaktion (interner PEND KP, UTM-Mehrschritttransaktion).

– Die Anweisung COMMIT WORK

  COMMIT WORK beendet immer auch die UTM-Transaktion (Einschritt- und Mehrschritt-Transaktion). In Dialog- Programmen im Betrieb mit TP-Monitor bewirkt die Anweisung ohne WITH-Klausel , daß die Transaktion beendet wird und der Programmlauf ohne Bildschirmausgabe fortgesetzt wird (interner PEND SP). Wird ein DRIVE-Programm als UTM-Asynchronvorgang gestartet, beendet COMMIT WORK auch den DRIVE-Vorgang.

  Wird auf einen Sicherungspunkt zurückgesetzt, der eine Ausgabe von mehreren Teilbildschirmen (bedingt durch einen impliziten Überlauf bei DISPLAY) abschließt, wird nur der letzte Teilbildschirm erneut ausgegeben. In vorherige Bildschirme eingelesene Werte können nicht mehr verändert werden.

  Erfolgt bei implizitem Überlauf eine Bildschirmausgabe, wird der Dialog-Schritt mit PEND KP beendet. Empfehlung: Ein Transaktionsende mit gekoppelter Bildschirmausgabe sollte ohne impliziten Bildschirm-Überlauf erfolgen.

– Die Anweisung ROLLBACK WORK

  ROLLBACK WORK setzt die SQL- und die UTM-Transaktion zurück. Es wird keine Nachricht ausgegeben (interner RSET an UTM). Die DRIVE-Anwendung wird mit der Anweisung nach ROLLBACK WORK weitergeführt.

– Die Anweisung ROLLBACK WORK WITH RESET

  ROLLBACK WORK WITH RESET setzt auf den letzten COMMIT-Stand zurück. Die DRIVE-Anwendung wird mit der Anweisung, die auf den letzten durchgeführten COMMIT WORK folgt, weitergeführt. Wenn noch kein COMMIT WORK im Programm durchlaufen wurde, wird das Programm abgebrochen. Der Anwender muß das DRIVE-Programm neu starten.(Intern wird ein RSET un UTM geschickt und die DRIVE-Transaktion zurückgesetzt; bei Remote-Zugriff auf BS2000-Datenbanken und bei VTV wird ein PEND RS (Vorgangswiederanlauf) an UTM geschickt.)

– Die Anweisungen BREAK und END PROCEDURE beim Übergang in den Dialog-Modus sowie Syntax- und Ablauffehler

  Ist bei diesen Anweisungen oder zum Zeitpunkt einer Fehlermeldung noch eine SQL-Transaktion offen, setzt DRIVE/WINDOWS sie zurück und beendet den Dialogschritt und die UTM-Transaktion. (Intern wird ein RSET-Aufruf an UTM geschickt und ein PEND RE.)

  Ist zum Zeitpunkt der Bildschirmausgabe keine SQL-Transaktion offen, wird dieser Dialogschritt abgeschlossen (interner PEND RE), um die UTM-Transaktion zu beenden.

COMMIT WORK und ROLLBACK WORK [WITH RESET] sind auch erlaubt, wenn keine Anweisung OPTION DBSYSTEM gegeben wurde. In diesem Fall wirken diese Anweisungen auf das Datenbanksystem, das im Rahmen der aktuellen CALL-Programmhierarchie zuletzt angesprochen wurde oder, falls noch kein Datenbanksystem angesprochen wurde, nur auf UTM- und DRIVE-Transaktionen. Ist die OPTION DBSYSTEM mit dem Wert OFF belegt, wirken die Anweisungen COMMIT WORK und ROLLBACK WORK [WITH RESET] auf UTM- und DRIVE-Transaktionen. Ist ein anderer Wert für OPTION DBSYSTEM angegeben, wirken diese beiden Anweisungen auf UTM-, DRIVE- und SQL-Transaktionen.

Wiederanlauf

DRIVE/WINDOWS im UTM-Betrieb bietet genau dann einen Wiederanlauf, wenn auch UTM dazu in der Lage ist.

Spezialinformation

Im Betrieb mit TP-Monitor werden die Daten, die DRIVE/WINDOWS für einen eventuellen Wiederanlauf benötigt, gemäß der UTM-Transaktionskonvention gesichert. Interne Bereiche, die DRIVE/WINDOWS zur Ausführung von Anweisungen eines einzelnen Anwenders benötigt, werden in Lokalen Sekundären Speicherbereichen (LSSB) gesichert. UTM setzt im Fehlerfall die LSSBs auf den Stand zurück, den sie zum Zeitpunkt der letzten ordnungsmäß abgeschlossenen UTM-Transaktion (PEND) hatten.

In der folgende Tabelle ist dargestellt, wie DRIVE/WINDOWS mit TP-Monitor in den einzelnen Fehler- bzw. Wiederanlaufsituationen reagiert.

Vorgangsspezifisches Verhalten im Fehlerfall

Ursache	Wirkung
Vorgang DRIVE wurde beendet wegen Programmfehler	Nach Eingabe des TACs DRISQL wird der Eröffnungstext ausgegeben. Es findet kein Wiederanlauf statt.
Vorgang DRIVE wurde unterbrochen durch KDCOFF oder KDCSHUT (nach abgeschlossener Transaktion)	Nach Neuanmelden bei UTM (EingabeOPEN KDCSIGNund UTM-Anwendungsname) wird der Bildschirm des letzten Sicherungspunktes ausgegeben; dies ist bei einfachem COMMIT WORK der erste Bildschirm der abgebrochenen Transaktion (beim Bildschirmwiederanlauf) u. bei komplexem COMMIT WORK der mit dieser Anweisung verbundene Bildschirm. Erfolgte die letzte Bildschirmausgabe im Dialog-Modus, so kann der Anwender normal weiterarbeiten. Auch Programme laufen nach Eingabe des Anwenders normal weiter.

Ursache	Wirkung
Vorgang DRIVE wurde unterbrochen wegen Zeitüberschreitung (PEND KP-Timer) oder KDCSHUT KILL (bei offener Mehrschritt-Transaktion)	Nach Neuanmelden bei UTM (EingabeOPEN KDCSIGNund UTM-Anwendungsname) wird der Bildschirm des letzten Sicherungspunktes ausgegeben; dies ist bei einfachem COMMIT WORK der erste Bildschirm der abgebrochenen Transaktion (beim Bildschirmwiederanlauf) u. bei komplexem COMMIT WORK der mit dieser Anweisung verbundene Bildschirm. Der Anwender kann normal weiterarbeiten, Programme laufen nach Bildschirmeingabe des Anwenders normal weiter.
Vorgang DRIVE wurde unterbrochen durch Ausschalten des Terminals nach erfolgter Eingabe UTM-Anwendung wurde abgebrochen durch fehlerhafte Beendigung von UTM (Datenbanksystem bleibt aktiv!) oder Systemabsturz	Nach Neuanmelden bei UTM (EingabeOPEN KDCSIGN) wird der Bildschirm des letzten Sicherungspunkts ausgegeben; dies ist bei einfachem COMMIT WORK der erste Bildschirm der abgebrochenen Transaktion (beim Bildschirmwiederanlauf) u. bei komplexem COMMIT WORK der mit dieser Anweisung verbundene Bildschirm. Der Anwender muß lediglich die Eingaben wiederholen, die seit Beginn der letzten Transaktion erfolgt sind. Auch Programme laufen nach Bidschirmeingabe des Anwenders normal weiter
Datenbanksystem SESAM/UDS setzt Transaktion zurück	Im Dialog-Modus wird eine Meldung ausgegeben. Im Programm-Modus wird auf der Anweisung hinter dem letzten vorangegangenen COMMIT WORK innerhalb des Programms wieder aufgesetzt. Wurde bei einem gesicherten Programm noch kein COMMIT WORK durchlaufen, so erfolgt ein Programmabbruch, es sei denn, das Programm wurde als Folgeprogramm von einem transaktionsgesicherten Programm aufgerufen. Das Rücksetzen bewirkt auch ein internes Rücksetzen von Speicherbereichen. Erfolgt das Rücksetzen innerhalb der 1. UTM-Transaktion , so wird der DRIVE-Vorgang abgebrochen.
Datenbanksystem SESAM/UDS setzt Vorgang zurück durch Administration oder Betriebsmittelengpaß	Im Dialog- und Programm-Modus wird, abhängig vom SQLCODE oder SQLSTATE, eine Meldung ausgegeben. Der Anwender kann darauf entsprechend reagieren: Programmabbruch erzwingen oder normal ab dem letzten COMMIT WORK weiterarbeiten, vorausgesetzt es wurde bereits ein COMMIT WORK durchlaufen und die evtl. benötigten Nachdefinitionen temporärer SQL-Objekte werden vom Datenbanksystem akzeptiert.

10.2.3 Auswirkungen der Transaktionssicherung auf Debug-Anweisungen

Wird eine SQL-Transaktion systemseitig oder durch die Anweisung ROLLBACK WORK WITH RESET zurückgesetzt, setzt DRIVE/WINDOWS im Debug-Modus den Programmlauf auf die letzte COMMIT WORK-Anweisung zurück oder auf den End-Haltepunkt, falls noch kein COMMIT WORK durchlaufen wurde.

Wird auf einen COMMIT WORK-Stand zurückgesetzt, befindet sich der Debugger in dem Zustand, den er nach der Anweisung COMMIT WORK hatte. D.h. die Wirkung von AT- und REMOVE-Anweisungen, die in der zurückgesetzten Transaktion gegeben wurden (Testpunkte und Aktionen vereinbaren und löschen), geht verloren, und Änderungen von Durchlaufzählern werden rückgängig gemacht. Veränderungen von DRIVE-Variablen durch SET-Anweisungen im Debugger werden wie im Programm-Modus zurückgesetzt.

Eine einfache ROLLBACK WORK-Anweisung hat keine Auswirkungen auf Debug-Objekte und DRIVE-Variablen.

10.2.4 Auswirkungen der Transaktionssicherung auf Dateibearbeitung

Die Dateibearbeitung unterliegt nicht der Transaktionssicherung (siehe auch Kapitel „Dateien bearbeiten“ auf Seite 389).

10.3 Auswirkungen der Transaktionssteuerung auf Definitionen

Die von DRIVE/WINDOWS unterstützten Datenbanksysteme sind transaktionsfähig im TIAM- und UTM-Betrieb. Auf alle Datenbanken kann im TIAM- und UTM-Betrieb zugegriffen werden. Für Auswirkungen der Transaktionssteuerung auf Definitionen in verteilten Anwendungen siehe Kapitel „Verteilte Anwendungen“ auf Seite 299 und Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339 .

Bei den Datenbanksystemen SESAM V1 und V2 und UDS unterliegen alle deklarativen Anweisungen, d.h. DECLARE CURSOR und CREATE TEMPORARY VIEW für temporäre SQL-Objekte, der Transaktionssicherung. Bei SESAM V2 unterliegen die DDL-Anweisungen zur Manipulation von persistenten SQL-Objekten der Transaktionssicherung, d.h. CREATE und ALTER TABLE, DROP TABLE und VIEW, CREATE SCHEMA und VIEW.

Transaktionen können aus einem internen oder externen Anlaß zurückgesetzt werden. Bei einem internen Rücksetzen wird die Transaktion vom Datenbanksystem zurückgesetzt, z.B. wegen eines Verarbeitungsfehlers. Bei einem externen Rücksetzen hat der Programmierer explizit die Anweisung ROLLBACK WORK gegeben.

Die Reaktionen von DRIVE/WINDOWS auf ein Rücksetzen hängen von mehreren Faktoren ab:

– Handelt es sich um ein internes oder externes Rücksetzen?

– War das letzte COMMIT WORK im Dialog- oder Programm-Modus?

– Läuft DRIVE/WINDOWS gerade im Dialog- oder Programm-Modus?

– Läuft DRIVE/WINDOWS gerade im TIAM- oder UTM-Betrieb?

– Ist das Speichermedium für die Wiederanlaufinformationen transaktionsfähig oder nicht?

Die Reaktionen auf die verschiedenen Faktoren werden unten ausführlich dargestellt. Allgemein läßt sich das DRIVE-Verhalten folgendermaßen beschreiben:

Dialog-Modus

Bei externem Rücksetzen zieht DRIVE/WINDOWS keine Definitionen nach. Der Anwender hat die Kontrolle und kann Definitionen explizit geben und freigeben. Ein internes Rücksetzen wird durch eine Meldung angezeigt. Alle Definitionen seit dem letzten COMMIT WORK-sind vergessen.

Programm-Modus

Die Definitionen für temporäre SQL-Objekte sollen außerhalb der Transaktionssicherung liegen. DRIVE/WINDOWS wiederholt deshalb für die Datenbanksysteme SESAM V1 und V2 und UDS bei externem Rücksetzen alle Definitionen seit dem letzten COMMIT WORK. Dabei kann es sich um Definitionen handeln, die zu mehreren Programmen gehören, z.B. bei Durchlaufen einer Aufrufhierarchie seit dem letzten COMMIT WORK. Die Anweisung ROLLBACK WORK wird im Programm-Modus mit der Klausel WITH RESET angeboten. D.h. die Programmsteuerung wird auf die Anweisung nach dem letzten Sicherungspunkt (letzter COMMIT WORK) zurückgesetzt, und die Definitionen von SQL-Objekten werden nur in bezug auf diesen Sicherungsstand nachgezogen.

Im Programm-Modus sind keine expliziten DROP-Anweisungen zugelassen. Ausnahmen: DROP wird für dynamische SQL-Objekte über eine EXECUTE-Anweisung ausgeführt oder es wird die aktuelle Belegung einer variablen Cursordeklaration gelöscht.

DRIVE/WINDOWS gibt alle SQL-Objekte beim Übergang in den Dialog-Modus frei.

Reaktionen bei internem Rücksetzen der Transaktion

Bei einem internen Rücksetzen durch SESAM oder UDS wird, je nach Situation,

– in den Dialog-Modus verzweigt oder

– die Programmsteuerung hinter die letzte COMMIT WORK-Anweisung zurückgesetzt.

Es hängt dann von der Programmlogik und den Eingabedaten ab, ob anschließend wieder dieselben oder andere Anweisungen durchlaufen werden wie in der zurückgesetzten Transaktion.

Reaktionen bei internem Rücksetzen der Transaktion durch SESAM oder UDS

	Letzter COMMIT im Dialog-Modus	**Letzter COMMIT im Programm-Modus**
TIAM-Betrieb	Noch im Dialog: Meldung ”TA ZURUECKGESETZT”; weiter im Dialog-Modus. Definitionen seit letztem COMMIT WORK sind weg. Schon im Programm: Meldung ”TA ZURUECKGESETZT, PROGRAMMABBRUCH”; weiter im Dialog-Modus. Definitionen seit letztem COMMIT WORK sind weg.	Noch im Programm: Meldung ”TA ZURUECKGESETZT”; nach der Bestätigung setzt die Programmsteuerung auf der Anweisung auf, die dem letzten COMMIT WORK folgt. Schon im Dialog: Meldung ”TA ZURUECKGESETZT”; weiter im Dialog-Modus. Definitionen seit Programmende sind weg.
UTM-Betrieb	Noch im Dialog: Meldung ”TA ZURUECKGESETZT”, weiter im Dialog-Modus mit letztem Bildschirm; Definitionen seit letztem COMMIT WORK sind weg. Schon im Programm: Meldung ”TA ZURUECKGESETZT, PROGRAMMABBRUCH”; weiter im Dialog-Modus mit letztem Bildschirm. Definitionen seit letztem COMMIT WORK sind weg.	Noch im Programm: Meldung ”TA ZURUECKGESETZT”; nach der Bestätigung erfolgt die Ausgabe des letzten Bildschirms vom vorherigen Sicherungspunkt; dies ist bei einfachem COMMIT WORK der 1. Bildschirm der abgebrochenen Transaktion (beim Bildschirmwiederanlauf) u. bei komplexem COMMIT WORK der mit dieser Anweisung verbundene Bildschirm. Nach der Bestätigung setzt die Programmsteuerung auf der Anweisung auf, die dem letzten COMMIT WORK folgt. Schon im Dialog: Meldung ”TA ZURUECKGESETZT”; weiter im Dialog-Modus; Definitionen seit Programmende sind weg.

Reaktionen bei ROLLBACK WORK

	Letzter COMMIT im Dialog-Modus	Letzter COMMIT im Programm-Modus
TIAM-Betrieb	Noch im Dialog: Meldung ”ANWEISUNG AUSGEFUEHRT”; weiter im Dialog-Modus. Definitionen seit letztem COMMIT sind weg. Schon im Programm: Programm-Steuerung auf der nächsten Anweisung. DRIVE/WINDOWS zieht intern Definitionen nach für alle statischen SQL-Objekte, deren Definitionen noch nicht durch ein COMMIT WORK festgeschrieben sind	Noch im Programm: Programm-Steuerung auf der nächsten Anweisung. Schon im Dialog: Meldung ”ANWEISUNG AUSGEFUEHRT”; weiter im Dialog-Modus. Definitionen seit Programmende sind weg.
UTM-Betrieb	Noch im Dialog: Meldung ”ANWEISUNG AUSGEFUEHRT”; weiter im Dialog-Modus. Definitionen seit letztem COMMIT WORK sind weg. Schon im Programm: Programm-Steuerung auf der nächsten Anweisung; (Interne DRIVE-Zwischenspeicherung in taskspezifischen Bereichen, dann RESET-Aufruf an UTM) DRIVE/WINDOWS zieht intern Definitionen nach für alle statischen SQL-Objekte, deren Definitionen noch nicht durch ein COMMIT WORK festgeschrieben sind	Noch im Programm: Programm-Steuerung auf der nächsten Anweisung; (Interne DRIVE-Zwischenspeicherung in taskspezifischen Bereichen, dann RESET-Aufruf an UTM) Schon im Dialog: Meldung ”ANWEISUNG AUSGEFUEHRT”; weiter im Dialog-Modus. Definitionen seit Programmende sind weg.

Reaktionen bei ROLLBACK WORK WITH RESET

	Letzter COMMIT im Dialog-Modus	Letzter COMMIT im Programm-Modus
TIAM-Betrieb	Meldung ”TA ZURUECKGESETZT, PROGRAMMABBRUCH”; weiter im Dialog-Modus. Definitionen seit letztem COMMIT WORK sind weg.	Die Programm-Steuerung setzt auf die Anweisung auf, die dem letzten COMMIT WORK folgt; weiter im Programm-Modus.
UTM-Betrieb	Meldung ”TA ZURUECKGESETZT, PROGRAMMABBRUCH”; weiter im Dialog-Modus mit letztem Bildschirm. Definitionen seit letztem COMMIT WORK sind weg.	Letzte Bildschirmausgabe vom vorherigen Sicherungspunkt. Nach der Bestätigung setzt die Programm-Steuerung auf die Anweisung auf, die dem letzten COMMIT WORK folgt

Gültigkeitsbereich und Lebensdauer von Definitionen

Zu den Objektklassen, die in DRIVE/WINDOWS mit CREATE oder DECLARE angelegt werden können, gehören Variablen, Konstanten, Views, Cursor, Bildschirme (Screens) sowie DRIVE-Bildschirm- und Listenformate.

Anweisung	Objekttyp	DB-System	Betriebsmodus	Definitionsarten
TEMP VIEW	SQL	SESAM V1 u. V2/UDS	alle	statisch/dynamisch
CURSOR	SQL	SESAM V1 u. V2/ UDS	alle	statisch/dynamisch
VARIABLE	DRIVE		Programm/Debug	statisch
CONSTANT	DRIVE		Programm/Debug	statisch
LIST	DRIVE		Programm/Debug	statisch/dynamisch
SCREEN	DRIVE		Programm/Debug	statisch
FORM	DRIVE		Programm/Debug	statisch/dynamisch

Ein Objekt heißt dynamisch definiert, wenn es im Rahmen einer EXECUTE-Anweisung definiert wurde, andernfalls heißt es statisch definiert. Dynamisch definierte Objekte sind DRIVE/WINDOWS und dem ggf. zugehörigen Datenbanksystem zum Übersetzungszeitpunkt nicht bekannt.

Persistente SQL-Objekte werden von den Datenbanksystemen transaktionsgesichert. Reports und Dateien sind weder SQL- noch DRIVE-Objekte, sondern externe Objekte, die nicht der Transaktionssicherung unterliegen.

Variable Cursor sind statische DRIVE-Objekte, die zwischen der dynamischen Spezifikation einer FOR-Klausel und der nächsten DROP-Anweisung zu dynamischen SQL-Objekten werden.

Gültigkeitsbereiche statisch und dynamisch definierter Objekte

Der **Gültigkeitsbereich** eines Objektes reicht im Programm-Modus von der Deklaration bis zum Ende (oder Abbruch) des Programms, in dem das Objekt deklariert wurde. Im Dialog-Modus reicht der Gültigkeitsbereich von Objekten von der Deklaration bis zur DROP-Anweisung oder bis zum Ende der DRIVE-Sitzung.

– Objekte werden entweder für den Programm/Debug-Modus oder für den Dialog-Modus angelegt und sind nur im jeweiligen Modus ansprechbar. Es können also keine Objekte angelegt werden, die im Programm/Debug-Modus und im Dialog-Modus gelten.

– Jedes Programm hat seine eigene Deklarationsumgebung. Das heißt, daß auch Objekte gleichen Namens, die in verschiedenen Programmen definiert sind, unterschiedliche Objekte bezeichnen.

– In einem Programm kann **nicht** auf Objekte eines anderen Programms zugegriffen werden. Bei einer Verzweigung mit CALL von Programm A in B wird die Definitionsumgebung des aufrufenden Programms A ungültig und diejenige des aufgerufenen Programms B gültig. Nach dem Rücksprung ist wieder die Definitionsumgebung des aufrufenden Programms A gültig und diejenige des aufgerufenen Programms B ungültig.

– Innerhalb eines Programms dürfen sich dynamische Definitionen auf statische des gleichen Programms beziehen. Statische Definitionen können sich **nicht** auf dynamische beziehen.

Lebensdauer temporärer und permanenter Objekte

Für ein Programm-Objekt ist mit dem Verlassen seines Gültigkeitsbereichs, nicht notwendig auch seine **Lebensdaue**r beendet, d.h. der Zeitraum, in dem es DRIVE/WINDOWS oder auch dem jeweiligen Datenbanksystem bekannt ist. Im Dialog-Modus dagegen sind Gültigkeitsbereich und Lebensdauer von Objekten gleich.

Temporäre Programm-Views werden von DRIVE/WINDOWS beim Übergang in den Dialog-Modus gelöscht. Dies gilt bei den Datenbanksystemen SESAM V1 und UDS uneingeschränkt, bei SESAM V2 nur für dynamische temporäre Programm-Views, weil SESAM V2 statische temporäre Programm-Views selbst verwaltet. Dynamische temporäre Dialog-Views und dynamische temporäre Programm-Views können vom Anwender gelöscht werden. Bei SESAM V2 ist dabei zu beachten, daß

1. jeder dynamische temporäre View einen Eigentümer hat, nämlich den zum Zeitpunkt seiner Definition aktuellen Berechtigungsschlüssel

2. nur der Eigentümer einen dynamischen temporären View löschen kann, so daß zum Zeitpunkt des Löschens derselbe Berechtigungsschlüssel aktuell sein muß wie zum Zeitpunkt der Definition.

Solange ein Objekt lebt, sind ihm immer auch ein oder mehrere objektklassenspezifische „Werte“ zugeordnet, z.B.

Objektklasse	**Werte**
temporärer View	Viewtabelle
Cursor	Cursortabelle, Cursorposition
Variable	Datenwert, NULL-Wert-Anzeige
DRIVE-Bildschirmformat	Eingabepuffer, Ausgabepuffer
Listenformat	Ausgabepuffer
FHS- Bildschirmformat	Eingabepuffer, Ausgabepuffer

Bei folgenden Objektklassen können die Lebensdauer von Objekten und damit auch die zugeordneten Werte durch die Angabe von PERMANENT oder TEMPORARY (Standard-Vorbelegung) bei der DECLARE-Anweisung beeinflußt werden:

Objektklasse	**beeinflußbare Werte**
Cursor	Cursorposition
Variable	Datenwert, NULL-Wert-Anzeige
DRIVE-Bildschirmformat	Eingabepuffer, Ausgabepuffer
DRIVE-Listenformat	Ausgabepuffer

Bei Dialog-Cursorn hat die Angabe PERMANENT keinen Effekt. Bei Variablen ist die Angabe PERMANENT nur auf der obersten Stufe erlaubt.

Die Angabe PERMANENT in DRIVE/WINDOWS verlängert die Lebensdauer für das angelegte Objekt (Stufennummer 1) über das Programmende hinaus. Permanente SQL-Objekte hingegen sind in der Datenbank festgeschrieben und existieren nicht nur in anwendungsspezifischen Speicherbereichen. Der Begriff permanent bedeutet also bei DRIVE und SQL etwas anderes, z.B. sind permanente Cursor temporäre SQL-Objekte. Deshalb heißen bei DRIVE permanente SQL-Objekte persistent.

Regeln zur Lebensdauer bei der Angabe PERMANENT

1. Statisch definierte Objekte
   - DRIVE-Objekte, d.h. Variablen oder DRIVE-Formate

     Für die Werte von Variablen und DRIVE-Formaten gilt die Lebensdauer auch nach Verlassen des Programms, wenn es mit CALL aufgerufen wurde. Die Werte bleiben erhalten und können bei einem weiteren Aufruf wieder verwendet werden.

Beispiel: Definieren einer permanenten Variablen

```
DCL VAR 1 &ARTIKEL PERMANENT,
           2 ARTNR CHAR (06),
           2 ARTNAME CHAR (15),
           2 ARTPREIS NUM (7,2);
```

Die Variable &ARTIKEL wird nur beim ersten CALL-Aufruf initialisiert. Bei folgenden CALL-Aufrufen und bei Unterprogrammende bleiben die Werte erhalten.

Die PERMANENT-Angabe ist nur auf der obersten Stufe einer Variablen erlaubt.

```
PROC DDD USING &G INT,
            1 &H1,
              2 H2 PERM,    → Fehler: nicht oberste Stufe
                3 H3 CHAR;
```

– SQL-Objekte, d.h. Cursor

Für Cursor gilt die Lebensdauer bis zum Ende der DRIVE-Anwendung (Übergang in den Dialog-Modus), bei dem sie automatisch gelöscht werden. Der Anwender kann statisch definierte Cursor nicht explizit mit DROP-Anweisungen löschen. Die Cursorposition bleibt bis Transaktionsende oder bis zum Schließen des Cursor erhalten (siehe „DRIVE-Anweisungen und -Klauseln für Cursor“ auf Seite 256). Die Cursorposition bleibt über Programmende hinaus erhalten, wenn vor Verlassen des Programms kein CLOSE gegeben wird und weder im gerufenen Programm noch im rufenden Programm zwischen den CALL-Aufrufen ein COMMIT WORK durchlaufen wird. Die Cursorposition geht auch außerhalb des Programms verloren durch Transaktionsende oder Rückkehr in den Dialog-Modus. Bei Rückkehr in den Dialog-Modus wird der Cursor implizit mit DROP freigegeben.

2. Dynamisch definierte Objekte

   – DRIVE-Objekte, d.h. Variablen oder DRIVE-Formate

   Für dynamisch definierte DRIVE-Objekte gilt die gleiche Regelung wie für statische.

   – SQL-Objekte, d.h. Cursor

   Für Cursor gelten Lebensdauer und Cursorposition bis zur expliziten Freigabe über dynamische DROP- oder CLOSE-Anweisungen in dem Programm, in dem diese definiert wurden oder bis zum automatischen Löschen beim Ende der DRIVE-Anwendung (Übergang in den Dialog-Modus).

Regeln zur Lebensdauer bei der Angabe TEMPORARY

1. Statisch definierte Objekte
   - DRIVE-Objekte, d.h. Variablen oder DRIVE-Formate

     Die variablen Werte dieser DRIVE-Objekte werden bei jedem Programmaufruf initialisiert (Standardvorbelegung oder INIT-Werte).
   - SQL-Objekte, d.h Cursor

     Für Cursor gilt die Lebensdauer bis zum Transaktionsende auf der nächst höheren Programmstufe oder bis zum automatischen Löschen beim Ende der DRIVE-Anwendung. Der Anwender kann diese Objekte nicht explizit mit DROP-Anweisungen freigeben. Bei Ende oder Abbruch des Programms, in dem die Cursor deklariert wurden, werden noch offene Cursor automatisch geschlossen. Daher sind nach Verlassen des Programms Cursorpositionen nicht mehr vorhanden.
2. Dynamisch definierte Objekte
   - DRIVE-Objekte, d.h. DRIVE-Formate

     Für DRIVE-Objekte gilt die gleiche Regelung wie für statische.
   - SQL-Objekte, d.h. Cursor

     Lebensdauer und Cursorposition gelten bis zur expliziten Freigabe über dynamische DROP- und CLOSE-Anweisungen in dem Programm, in dem sie deklariert wurden oder bis zum Transaktionsende auf der nächst höheren Programmstufe oder bis zum automatischen Löschen beim Ende der DRIVE-Anwendung.

Regeln zur Lebensdauer für variable Cursor

Entscheidend ist der Zeitraum, in dem der Cursor beim Datenbanksystem bekannt ist, d.h. variable Cursor verhalten sich wie dynamische. Es gilt also folgendes:

- permanente variable Cursor

  Lebensdauer und Cursorposition gelten bis zur expliziten Freigabe über DROP- oder CLOSE-Anweisungen in dem Programm, in dem der Cursor deklariert wurde oder bis zum automatischen Löschen beim Ende der DRIVE-Anwendung.
- temporäre variable Cursor

  Lebensdauer und Cursorposition gelten bis zur expliziten Freigabe über DROP- oder CLOSE-Anweisungen in dem Programm, in dem der Cursor deklariert wurde oder bis zum Transaktionsende auf der nächst höheren Programmstufe oder bis zum automatischen Löschen beim Ende der DRIVE-Anwendung.

Im BS2000 entspricht das Ende der DRIVE-Anwendung dem Übergang in den Dialog-Modus.

Die folgende Tabelle faßt die Regeln zur Lebensdauer von Objekten zusammen:

Definitionsart	**Lebensdauer von**	**Lebensdauer bis**
Dialog-Cursor Werte	DECLARE OPEN/vgl. Hinweis	DROP/Session-Ende CLOSE/TA-Ende/vgl. Hinweis
permanent statisch DRIVE-Werte SQL-Werte	erstem Ansprung vgl. Hinweis OPEN	Ende DRIVE-Anwendung vgl. Hinweis CLOSE/TA-Ende
temporär statisch SQL-Werte	jeder Ansprung OPEN	Rücksprung (DRIVE-Objekte) TA-Ende höhere Stufe (SQL-Objekte) Ende DRIVE-Anwendung CLOSE/Rücksprung/TA-Ende höhere Stufe
permanent dynamisch SQL-Werte	EXEC DECLARE EXEC OPEN	EXEC DROP (SQL-Objekte)/Ende DRIVE-Anwendung EXEC CLOSE/TA-Ende
temporär dynamisch SQL-Werte	EXEC DECLARE EXEC OPEN	EXEC DROP (SQL-Objekte)/TA-Ende höhere Stufe Ende DRIVE-Anwendung EXEC CLOSE/TA-Ende
permanent variabel SQL-Werte	EXEC DECLARE OPEN	DROP/Ende DRIVE-Anwendung CLOSE/TA-Ende
temporär variabel SQL-Werte	EXEC DECLARE OPEN	DROP/TA-Ende höhere Stufe/Ende DRIVE-Anwendung CLOSE/TA-Ende
Hinweis: Zu Anweisungen, die die Cursorposition verändern, siehe Seite 259. DRIVE-Werte werden durch verschiedene Anweisungen verändert, z.B. SET und FILL. Die Lebensdauer der Werte von DRIVE-Bildschirmformaten und von Listenformaten reicht bis zum ersten FILL nach (ggf. implizitem) DISPLAY.		

Die folgende Tabelle gibt einen Überblick zur Lebensdauer von Definitionen und ihren Werten in Abhängigkeit von Transaktions-Ereignissen. Mit Werten sind hier gemeint: Inhalte, Positionen, Datenwerte. Mit FORM und LIST sind die bereits mit DISPLAY ausgegebenen Formate bzw. Listen gemeint.

Definition/Wert	**CREATE**	**DECLARE**				**WERT**		
TA-Ereignis	**VIEW**	**CUR-SOR**	**FORM/ LIST**	**VAR/ CONST**	**WIN-DOW**	**VAR**	**FORM/ LIST**	**Cursor**
COMMIT WORK	F	F	F	F	F	F	F	EN
Ext. ROLLBACK mit RESET u. int. ROLLBACK	R	R	N	N	R	R	N	EN
Ext. ROLLBACK ohne RESET	N	R *	N	N	N	R*	N	EN
Programmende/-abbruch PERM TEMP	 N EN	 N G	 N EN	 N EN	 N EN	 N EN	 N EN	 N EN
Anwendungsende/-abbruch DRIVE-Ende	EN	EN	EN	EN	EN	EN	EN	EN
Übergang Programm-M. → Dialog-M.	EN	EN	EN	EN	-	EN	EN	EN
Übergang Dialog-M. → Programm-M.	N	N	-	-	EN	-	-	EN
Rücksetzen in 1. UTM-TA	VA	VA	VA	VA		VA	VA	VA

Legende

R zurückgesetzt

F festgeschrieben

N keine Wirkung

EN existiert nicht mehr

* im Programm-Modus wiederholt (Nachdefinition), falls Definition nicht schon durch COMMIT WORK festgeschrieben

10.4 Programmierhinweise und -regeln für OLTP-Programme

Die folgenden Programmierhinweise gelten für den lokalen Betrieb. Programmierregeln für eine Multi-Datenbank-Anwendung finden Sie im Kapitel „Datenbank-Unterstützung“ auf Seite 245, Programmierregeln für Verteilte Anwendungen im Kapitel „Verteilte Anwendungen“ auf Seite 299 und im Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339.

– Ein Programm darf nur im transaktionslosen Zustand übersetzt, getestet oder aufgerufen werden (Anweisungen COMPILE, DEBUG, DO).

– Vor dem Verlassen eines mit DO/DEBUG aufgerufenen Programms durch END PROCEDURE, BREAK, STOP oder Folge-DO muß ein COMMIT WORK oder ROLLBACK WORK durchlaufen worden sein, wenn die DRIVE-Anwendung SQL-Anweisungen enthält. Andernfalls wird das Programm abgebrochen und die Transaktion zurückgesetzt. Die SQL-Objekte des Programms (Cursor, temporäre Views) werden beim Verlassen in jedem Fall gelöscht.

– Ein DO in Programmen (Folge-DO) hat die gleiche Wirkung wie das Verlassen des Programm-Modus und Aufrufen des Programms mit DO aus dem Dialog-Modus: DRIVE/WINDOWS führt die Anweisung END PROCEDURE aus, baut die Definitionsumgebung der zuvor durchlaufenen Programme ab und startet das Folge-Programm.

– „Problem der ersten Transaktion“:

  DRIVE/WINDOWS bricht nach einem externen oder internen Rücksetzen die Anwendung oder den Vorgang stets dann ab, wenn keine Wiederanlaufinformation vorhanden ist, d.h. noch kein COMMIT WORK durchlaufen wurde.

  In Programmen mit Zugriff auf SESAM V1 oder UDS sollte unmittelbar hinter den Definitionen ein COMMIT WORK gesetzt werden. Damit sind die Definitionen der statischen SQL-Objekte im Datenbanksystem festgeschrieben.

– Anweisungen zur Transaktionssteuerung auf verschiedenen Programmstufen sind zwar erlaubt, es müssen aber die Regeln zur Lebensdauer von DRIVE- und SQL-Objekten beachtet werden.

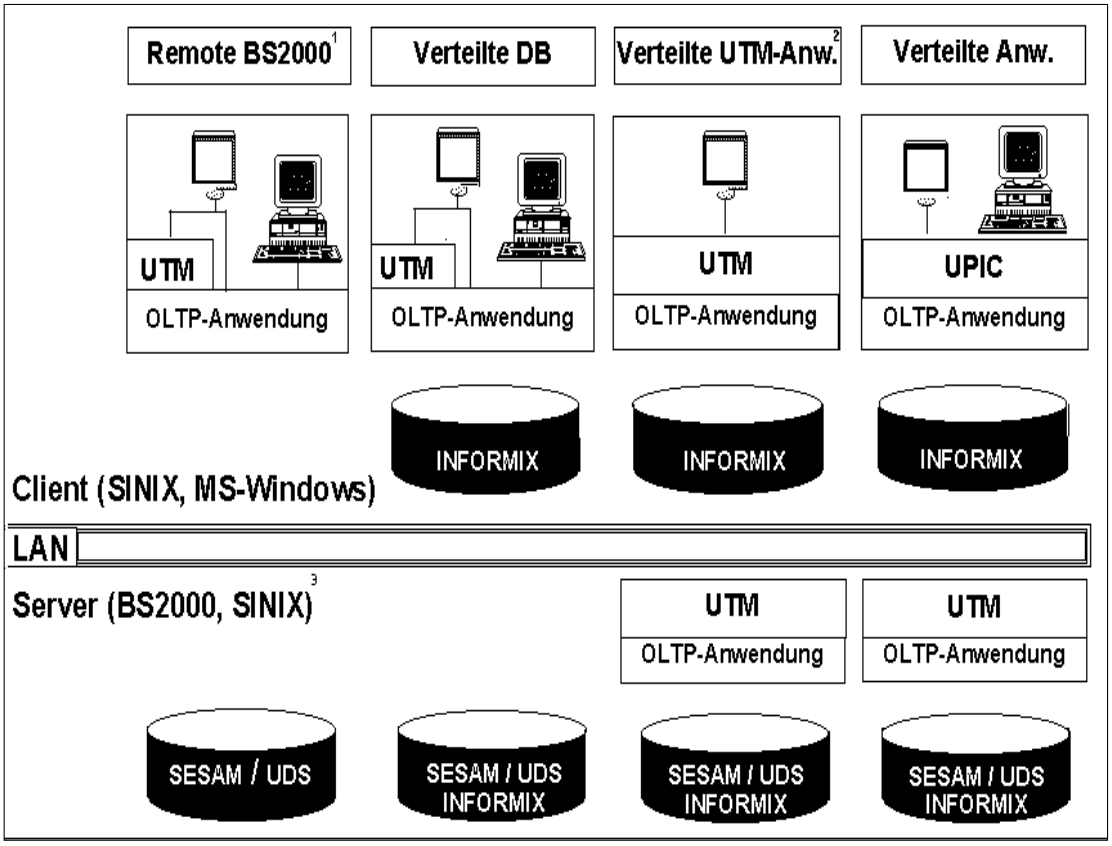
– Das „DRIVE-SQL-Lexikon für SESAM/SQL 2“ [5] enthält Konzepte und Programmierempfehlungen, die zu übersichtlichen Transaktionsprofilen je Programmstufe und zwischen Programmstufen führen (statische Programme, dynamische Programme und TA-übergreifende Programmkommunikation).

11 Client-Server-Konzepte

Die drei Komponenten von OLTP-Anwendungen - Präsentation, Verarbeitung, Datenbanken - können mit DRIVE/WINDOWS auf verschiedene Rechner verteilt werden, auf denen verschiedene Betriebssysteme laufen können. Als Client können MS-Windows oder SINIX eingesetzt werden, als Server SINIX oder BS2000. Die drei Komponenten können so verteilt werden, daß die Betriebsmittel optimal eingesetzt sind. Dieses Kapitel stellt die Client-Server-Konzepte, die DRIVE/WINDOWS unterstützt, plattformübergreifend dar.

Für die Präsentation gibt es eine grafische Bedienoberfläche unter MS-Windows und SINIX (OSF/Motif). Unter SINIX ist außerdem eine alphanumerische Bedienoberfläche mit FORMANT möglich. Als Datenbanksysteme gibt es im BS2000 SESAM V1, SESAM V2 und UDS, in SINIX und MS-WINDOWS INFORMIX-SE und INFORMIX-OnLine.

Je nachdem, wie die drei Komponenten Präsentation, Verarbeitung und Datenbanken verteilt sind, ergeben sich folgende Client-Server-Architekturen: Remote-Zugriff, Verteilte Datenbanken, Verteilte Verarbeitung. Die folgende Grafik zeigt diese Client-Server-Architekturen in vereinfachter Form:



1 Remote BS2000 ist beim Client MS-Windows nicht mit TP-Monitor (UTM) möglich, sondern nur über UPIC. Beim Client SINIX ist Remote BS2000 über UPIC (Grafik- und Alpha-Bedienoberfläche) oder mit TP-Monitor (UTM, nur Alpha-Bedienoberfläche) möglich.

2 Verteilte UTM-Anwendungen sind nur mit dem Client SINIX möglich.

3 Auf Server-Seite sind im BS2000 die lokalen Datenbanksysteme SESAM und UDS möglich, im SINIX das lokale Datenbanksystem INFORMIX.

11.1 Remote-Zugriff auf BS2000-Datenbanken

Sie können vom lokalen DRIVE-Anwendungsprogramm in SINIX oder MS-Windows auf Daten in einer entfernten BS2000-Datenbank (SESAM V1, SESAM V2 oder UDS) zugreifen. So stellen Sie entfernte Daten für eine lokale Verarbeitung bereit. Die Bedienoberfläche und die Verarbeitung laufen auf dem Client ab, die BS2000-Datenbanken auf dem Server.

Die DRIVE-Anwendung läuft unter MS-Windows ohne TP-Monitor. In SINIX kann sie im Betriebsmodus mit und ohne TP-Monitor laufen. Ohne TP-Monitor ist eine Grafik- und Alpha-Bedienoberfläche möglich, mit TP-Monitor nur eine Alpha-Bedienoberfläche. Der Server ist eine UTM-Anwendung, die mit der eingebundenen DRIVE-Netzkomponente DRIVE/WINDOWS-NET auf die Serverdaten zugreift. DRIVE/WINDOWS synchronisiert die Transaktionen des DRIVE-Anwendungsprogramms mit den Datenbank-Transaktionen.

Dabei haben die DRIVE-Anweisungen zur Transaktionssteuerung bei Remote-Zugriff dieselbe Wirkung wie lokal und werden im Kapitel „Transaktionskonzept“ dargestellt.

Die Einsatzvorbereitung für den Remote-Zugriff ist in der „DRIVE-SPU“ im Kapitel „OLTP-Anwendungen mit Remote-Zugriffen auf BS2000-Datenbanken“ beschrieben.

Mit welchen DRIVE-Anweisungen Sie das Datenbanksystem festlegen, auf das zugegriffen werden soll, ist im Kapitel „Datenbank-Unterstützung“ auf Seite 245 beschrieben. In einer DRIVE-Session kann nur auf ein BS2000-Datenbanksystem zugegriffen werden (SESAM V1 oder V2 oder UDS).

Mit dieser Client-Server-Architektur können Sie z.B.:

– in bestehenden DRIVE-Anwendungen die Alpha-Bedienoberfläche durch eine grafische Bedienoberfläche unter MS-Windows oder SINIX ersetzen, ohne daß sich der Datenbankzugriff ändert oder Daten migriert werden müssen.

– DRIVE-Anwendungen auf Rechner mit dem Betriebssystem MS-Windows oder SINIX auslagern und weiterhin den zentralen Datenbestand nutzen

11.2 Verteilte Datenbanken

Das jeweilige Datenbanksystem steuert Zugriffe auf die verteilten Datenbanken. Diese können sowohl auf dem Client und dem Server liegen als auch auf verschiedenen Servern. Für den DRIVE-Anwender bleibt diese Verteilung verborgen. Das jeweilige Datenbanksystem muß mit seiner Netzkomponente konfiguriert sein, also SESAM-DCN, UDS-D bzw. INFORMIX-STAR (SE)/INFORMIX-NET (OnLine) bei V5 (bei V6 integriert).

Bei INFORMIX können Sie auf verteilte Datenbanken nur im Betrieb ohne TP-Monitor zugreifen. Die Datenbanken können auf einen oder mehrere Rechner verteilt sein, müssen aber von dem gleichen INFORMIX-Backend-Typ (SE oder OnLine) verwaltet werden. Für den lesenden Zugriff können Sie entfernte Tabellen mit dem INFORMIX-System- und Datenbanknamen qualifizieren (siehe DRIVE-SQL-Lexikon für INFORMIX DATABASE-Anweisung). Für den schreibenden Zugriff müssen Sie die entfernte Datenbank über eine OPTION DATABASE-Angabe eröffnen.

Weitere Informationen finden Sie in den jeweiligen Handbüchern zur Netzkomponente des Datenbanksystems (siehe „INFORMIX NET/STAR“ [42])

Mit dieser Client-Server-Architektur können Sie z.B. Anwendungen mit Down- und Upload-Funktionen realisieren. Außerdem können Sie den Remote-Zugriff auf BS2000-Datenbanken (s.o.) um einen lokalen INFORMIX-Anschluß auf dem Client (MS-Windows oder SINIX) erweitern. In einer DRIVE-Anwendung mit mindestens zwei Programmen ist dann ein Zugriff auf INFORMIX und eines der folgenden BS2000-Datenbanksysteme möglich: SESAM V1 oder V2 oder UDS.

11.3 Verteilte Verarbeitung

Verteilt werden die Anwendungen an die Orte, an denen die Daten liegen. Damit wird der Datentransport minimiert und die Performance verbessert. Abhängig vom Betriebsmodus mit oder ohne TP-Monitor kann man im BS2000 und SINIX zwischen verteilten UTM-Anwendungen und verteilten Anwendungen unterscheiden.

Unter MS-Windows werden nur verteilte Anwendungen unterstützt. Auf der Server-Seite können Sie aber wiederum ein DRIVE-Programm in einer UTM-Anwendung im BS2000 oder SINIX aufrufen und die Funktionalität der verteilten UTM-Anwendung auf dieser Plattform nutzen.

Mit verteilter Verarbeitung können Sie auch Oldstyle-Prozeduren mit Zugriff auf SESAM oder LEASY aus Newstyle-Programmen im BS2000-Server aufrufen. Oldstyle-Prozeduren können dabei mit CALL aufgerufen und Parameter übergeben werden (siehe Kapitel „Integration von Old-Style-Prozeduren“ auf Seite 391).

11.3.1 Verteilte UTM-Anwendungen

Laufen Client- und Server-Anwendung (Auftraggeber und Auftragnehmer) im Betriebsmodus mit TP-Monitor, sprechen wir von verteilten UTM-Anwendungen oder Verteilter Transaktionsverarbeitung (VTV). Diese bietet DRIVE/WINDOWS im BS2000 und in SINIX. Unter MS-Windows können Sie über „Verteilte Anwendungen“ (s.u.) eine Server-Anwendung im BS2000 oder SINIX aufrufen, die als Auftraggeber dann die Funktionalität der „Verteilten UTM-Anwendungen“ nutzt.

Lokale Anwendungen geben Verarbeitungsaufträge an entfernte Anwendungen. Diese greifen auf lokale Daten zu, um ihren Verarbeitungsauftrag auszuführen. Die Anwendungen können auf Client- und Server-Seite mit voller Transaktionssicherheit verteilt werden. D.h. es gibt anwendungsübergreifende Transaktionen (Verteilte Transaktion), so daß sich die Daten in allen beteiligten Anwendungen und den von ihnen angesprochenen Datenbanksystemen in einem konsistenten Zustand befinden. Zu Besonderheiten der Transaktionssteuerung siehe Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339 und Kapitel „Verteilte Anwendungen“ auf Seite 299). Hier finden Sie auch die unterschiedlichen Einsatzvorbereitungen im Client und Server für die VTV.

DRIVE/WINDOWS unterstützt auch eine mehrstufige Hierarchie von Clients und Servern. D.h. ein DRIVE-Auftragnehmer (Server) kann selbst wieder Aufträge geben an weitere Auftragnehmer. DRIVE/WINDOWS übernimmt vollständig die Kommunikation zwischen Auftraggeber (Client) und Auftragnehmer (Server).

Performancevorteile sind möglich durch paralleles Aufrufen mehrerer Auftragnehmer-Programme im Auftraggeber (Client) (siehe „DRIVE-Lexikon“ [3], DISPATCH-Anweisung).

Sie können auch DRIVE-Anwendungen und Anwendungen in C bzw. COBOL miteinander kombinieren, müssen dann aber die UTM-Kommunikation der 3GL-Programme selbst programmieren.

Anwendungen in SINIX können auf das lokale Datenbanksystem INFORMIX zugreifen sowie über Remote-Zugriff (mit TP-Monitor) auf die BS2000-Datenbanksysteme SESAM V1 oder V2 und UDS, Anwendungen im BS2000 auf die lokalen Datenbanksysteme SESAM V1 und V2 und UDS.

Verteilte Transaktionsverarbeitung unterstützt nur Alpha-Bedienoberflächen.

11.3.2 Verteilte Anwendungen

Läuft die Client-Anwendung im Betriebsmodus ohne TP-Monitor (der Server ist immer eine UTM-Anwendung!), sprechen wir von verteilten Anwendungen. Mit dieser Client-Server-Architektur können Sie z.B. Daten von SESAM V1 oder V2 und UDS lokal und performant im BS2000 verarbeiten und mit einer modernen grafischen Bedienoberfläche unter MS-Windows oder SINIX (OSF/Motif) präsentieren (Trennung von Präsentation und Verarbeitung). DRIVE/WINDOWS als Client bietet einfache und leistungsfähige Window-Anweisungen für die Grafik-Programmierung.

Auf dem Server gibt es die Datenbanksysteme INFORMIX (SINIX) und SESAM V1 oder V2 sowie UDS (BS2000). Auf Client-Seite (MS-Windows oder SINIX) ist ebenfalls ein Datenbanksystem möglich, nämlich INFORMIX. Die lokalen Transaktionen in Client und Server werden aber nicht synchronisiert, d.h. es gibt keine Verteilte Transaktion wie bei VTV.

Auf der Client-Seite haben Sie zwei Möglichkeiten, mit einer DRIVE-Anwendung auf ein INFORMIX-Datenbanksystem zuzugreifen:
Über die datenbankinterne Verteilung durch INFORMIX-STAR (SE) oder -NET (OnLine) und über Verteilte Anwendungen durch DRIVE-Sprachmittel.
Beide Zugriffsarten können dieselbe Datenbank ansprechen. Das INFORMIX-Datenbanksystem verwaltet die beiden Zugriffe als zwei verschiedene Benutzer, die konkurrierend auf die Datenbank zugreifen.

Die DRIVE-Sprachmittel zum Verteilen der Anwendung, Besonderheiten bei der Transaktionssicherung sowie die unterschiedlichen Einsatzvorbereitungen im Client und Server sind im Kapitel „Verteilte Anwendungen“ auf Seite 299 beschrieben.

Beide Client/Server-Konzepte, Remote-Zugriff und verteilte Anwendungen, werden auf der Client-Seite mit UPIC realisiert. Das gleichzeitige Adressieren von zwei verschiedenen Servern über UPIC ist nicht möglich. Daher können Sie innerhalb einer DRIVE-Session nicht gleichzeitig einen Remote-Zugriff auf die BS2000-Datenbanken SESAM V1 oder V2 oder UDS ausführen und über einen Remote-CALL eine Anwendung im (BS2000- oder SINIX) Server aufrufen. Denn der Datenbank-Server bleibt bei allen Aufträgen vom Client offen und beendet sich erst bei Session-Ende des Client. DRIVE/WINDOWS gibt sonst eine Fehlermeldung aus.

Da zu einem Zeitpunkt nur eine Verbindung zwischen Client und Server möglich ist, können Serveraufträge nicht wie bei VTV parallelisiert werden.

In einer DRIVE-Session kann nur entweder mit einem SESAM V1- oder einem SESAM V2- oder UDS-Server gearbeitet werden. Innerhalb eines DRIVE-Programms kann nur auf ein Datenbanksystem zugegriffen werden, in einer DRIVE-Anwendung mit mindestens zwei Programmen kann auf INFORMIX und SESAM V1 oder V2 oder auf INFORMIX und UDS zugegriffen werden. Die im Kapitel „Datenbank-Unterstützung“ beschriebenen Regeln beim Wechsel von Programmen, insbesondere Vererbungsmechanismen und Einschränkungen bei offener Transaktion, gelten nicht für Remote-CALL-Aufrufe.

12 Verteilte Anwendungen

Dieses Kapitel beschreibt:

- die unterschiedlichen Anwendungen, die DRIVE/WINDOWS unterstützt
- die DRIVE-Anweisungen im Client-Programm (ab Seite 301)
- Auswirkungen von DRIVE-Anweisungen auf Server-Zustände und die logische Verbindung (ab Seite 328)
- Transaktionssicherung bei verteilten Anwendungen (ab Seite 307)
- Fehlersituationen und daraus folgende Programmierempfehlungen (ab Seite 312)
- die Datenübergabe zwischen DRIVE/WINDOWS und anwendereigenen UTM-Teilprogrammen in C/COBOL (ab Seite 316)
- die KDCDEF-Steueranweisungen und TNS-Einträge, die für die Kommunikation der Partneranwendungen auf beiden Seiten nötig sind (ab Seite 333).

Mit der Client-Server-Architektur „Verteilte Anwendungen“ können Sie die grafischen Möglichkeiten des Client-Systems MS-Windows, moderne Bedienoberflächen mit Fenstertechnik zu präsentieren, verbinden mit den leistungsfähigen Datenbanksystemen auf Server-Seite INFORMIX-ONLINE im SINIX und SESAM und UDS im BS2000. Dabei ist der lokale Datenbankzugriff im Server schnell und ohne Netzbelastung möglich. Übertragen werden müssen nur die ggf. geprüften Eingabedaten des Anwenders sowie die bearbeiteten Ausgabedaten.

Der Client ist eine DRIVE-Anwendung im Betriebsmodus ohne TP-Monitor. Diese DRIVE-Anwendung realisiert eine grafische Bedienoberfläche mit Hilfe des Dialog Builder (MS-WINDOWS). Die Schnittstelle im Client ist die X/OPEN-Schnittstelle CPIC wie sie über das Produkt UTM-UPIC realisiert wird. Der Client kann auch eine C-UPIC-Anwendung sein (Kopplungsmöglichkeiten siehe unten).

Der Server ist entweder eine DRIVE-Anwendung im Betriebsmodus mit TP-Monitor oder ein C/COBOL-UTM-Teilprogramm im SINIX oder BS2000. Der Server hat die Rolle des Auftragsnehmers in der Verteilten Verarbeitung mit UTM-D. Als Transportsystem wird TCP/IP verwendet.

Eine lokale Datenbank im Client ist ebenfalls möglich (INFORMIX-SE in MS-Windows). Es gibt aber nur lokale Transaktionen im Client und lokale Transaktionen im Server, die nicht synchronisiert werden. Eine verteilte, anwendungsübergreifende Transaktion wie bei Verteilten UTM-Anwendungen (VTV) gibt es nicht. D.h., Transaktionsanweisungen im Client sind von den Server-Transaktionen entkoppelt.

Beide Client/ Server-Architekturen, Remote-Zugriff auf die BS2000-Datenbanken SESAM und UDS und verteilte Anwendungen, werden auf der Client-Seite mit UPIC realisiert. Das gleichzeitige Adressieren von zwei verschiedenen Servern über UPIC ist nicht möglich. Daher können Sie innerhalb einer DRIVE-Session nicht gleichzeitig einen Remote-Zugriff auf die BS2000-Datenbanken SESAM oder UDS ausführen und über einen Remote-CALL eine Anwendung im BS2000- Server (DRIVE-Anwendung oder C/COBOL-Teilprogramm) aufrufen. Denn der Datenbank-Server bleibt bei allen Aufträgen vom Client offen und beendet sich erst bei Session-Ende des Client. DRIVE/WINDOWS gibt sonst eine Fehlermeldung aus.

Auf der Server-Seite können Sie DRIVE-Programme und C/COBOL-Teilprogramme mit der im BS2000 und SINIX möglichen Client/Server-Architektur der Verteilten UTM-Anwendungen (VTV) kombinieren. D.h. die Server-Anwendung als Auftragnehmer wird selbst wieder zum Auftraggeber in einer mehrstufigen Auftraggeber-Auftragnehmer-Hierarchie (siehe Kapitel „Verteilte Transaktionsverarbeitung (VTV)“ auf Seite 339). Über diese Client/Server-Architektur können Sie dann auch Oldstyle-Programme, die auf LEASY-Daten zugreifen, einbinden und so LEASY-Datenhaltung mit grafischen Bedienoberflächen kombinieren (siehe Kapitel „Integration von Old-Style-Prozeduren“ auf Seite 391).

DRIVE/WINDOWS unterstützt für Verteilte Anwendungen die folgenden Kopplungsmöglichkeiten:

Client	**Server**
DRIVE-Programm	DRIVE-Programm
DRIVE-Programm	C/COBOL-Teilprogramm
C-UPIC-Anwendung	DRIVE-Programm

In DRIVE-Programmen können Sie verteilte Anwendungen mit den DRIVE-Sprachmitteln realisieren. Bei C-UPIC-Anwendungen müssen Sie die entsprechenden UPIC-Aufrufe selbst programmieren (siehe Abschnitt „Ablauf der Kommunikation mit einem 3GL-Programm (für C-Programmierer)“ auf Seite 334), bei Anbindung an C-Teilprogrammme im Server die KDCS-Aufrufe.

Softwarevoraussetzungen allgemein (genaue Versionsangaben finden Sie in der Freigabemitteilung):

– DRIVE/WNDOWS V2.1

– UPIC-UTM

– UTM (Server-Seite in BS2000/SINIX)

12.1 Client-Seite

Die folgenden Abschnitte gelten für DRIVE-Programme als Client.

12.1.1 Verteilung vorbereiten

Verteilte Verarbeitung einstellen

Mit der Übersetzungsoption OPTION DISTRIBUTION wählen Sie in einem DRIVE-Programm zwischen lokaler und verteilter Verarbeitung. Sie können diese Option auch in der SPU über das Menü **Optionen** einstellen. Die Option im Programm hat für das aktuelle Programm Vorrang gegenüber der Menüeinstellung. Bei Programmende gilt wieder die Menüeinstellung.

Der Operand OFF bewirkt, daß jede CALL-Anweisung lokal verarbeitet wird.
Der Operand ON bewirkt, daß DRIVE/WINDOWS bei CALL-Aufrufen prüft, ob Verteilungsinformationen gesetzt worden sind. Sind keine Verteilungsinformationen gesetzt, wird der CALL- Aufruf lokal verarbeitet.

Sind Verteilungsinformationen gesetzt, versucht DRIVE/WINDOWS zunächst, eine Verteilungsinformation mit einem passenden Eintrag zu finden und das gerufene Programm entfernt zu verarbeiten. Ist kein passender Eintrag vorhanden, wird das Programm lokal verarbeitet.
Ein Eintrag ist passend, wenn die Werte für Bibliothek und Element mit denen bei der Anweisung CALL übereinstimmen.

Ist als Server ein DRIVE-Interpreter-Programm angegeben und nur der Elementname ohne Bibliothek, so wird dem Server-Vorgang nicht die im DRIVE-Programm als Client mit PARAMETER DYNAMIC LIBRARY eingestellte Bibliothek mitgeteilt. Der Server-Vorgang löst die Angabe vielmehr mit seiner eigenen PARAMETER-Voreinstellung auf (ggf. als DRIVE-Startparameter in der entfernten UTM-Anwendung siehe Abschnitt „Gültigkeit von Definitionen“ auf Seite 331).

Beim Aufruf eines anwendereigenen UTM-Teilprogramms mit CALL TAC werden die Werte in der Verteilungsinformation für den Transaktionscode geprüft. Der Operandenwert darf kein DRIVE-Interpreter-TAC oder DRIVE-Compiler-TAC sein. Beim Aufruf eines anwendereigenen UTM-Teilprogramms mit CALL muß zusätzlich die Programmiersprache, in der es geschrieben ist, angegeben werden.

Verteilungsinformationen angeben

Die Verteilungsinformationen werden mit der Anweisung PARAMETER DISTRIBUTION angegeben oder in der SPU über das Menü **Optionen**. Im folgenden wird zur Vereinfachung nur von der Anweisung gesprochen. Es gilt die jeweils letzte Einstellung, da die Parameter-Einstellungen im Gegensatz zu den Übersetzungsoptionen im Menü **Optionen** nicht programmspezifisch gelten. Als PARAMETER-Einstellungen sind die Verteilungsinformationen global in allen Programmstufen der aktuellen DRIVE-Session gültig. Mit jeder Anweisung werden die Verteilungsinformationen für genau ein Server-Programm festgelegt. Die Anweisung ist nur erlaubt, wenn kein Server adressiert ist. Für die RTS-Variante empfiehlt es sich daher, die Anweisung in die Startdatei von DRIVE/WINDOWS zu schreiben (siehe „DRIVE-SPU“ [7], Kapitel „Das Runtime-System“).

Mit der Anweisung PARAMETER DISTRIBUTION können folgende Angaben gemacht werden (siehe auch „DRIVE-Lexikon“ [3]):

– Bibliothek und Elementname.
– Typangabe CODE/OBJECT für ein DRIVE-Interpreter- bzw. DRIVE-Compiler-Programm.
– Bei OBJECT wird die Angabe einer Bibliothek ignoriert.
– Transaktionscode für den Server-Auftrag, der ein anwendereigenes C/COBOL-Teilprogramm ist.

Diese Angaben werden dann beim Aufrufen eines Programms mit den Angaben bei CALL verglichen und müssen genau übereinstimmen. Wird z.B. bei Bibliothek ein Pfadname angegeben, muß dieser Pfadname auch beim CALL-Aufruf angegeben werden. Auch Groß-/Kleinschreibung muß beachtet werden.

APPLICATION-Klausel. Der Symbolic Destination Name verweist auf einen Eintrag in der Side-Information Datei (`Upicfile`), die die Umgebung von UPIC-Anwendungen bestimmt (siehe „DRIVE-Programmiersystem“ [1], Abschnitt Verteilte Anwendungen generieren). Fehlt die APPLICATION-Klausel, so wird der Symbolic Destination Name aus der Anweisung PARAMETER STATIC SIDEINFO oder dem Menü **Optionen**/statische Parameter verwendet (siehe „DRIVE-Lexikon“ [3]).

Die Angaben für Bibliothek, Elementname und Transaktionscode sind immer eindeutig, da eine Verteilungsinformation mit denselben Operandenwerten die alten Werte überschreibt. Diese Werte müssen also über alle Server-Anwendungen, die vom aktuellen Client adre-

siert werden, eindeutig sein. Andernfalls muß vor dem CALL eines gleichnamigen Programms in einer anderen Serveranwendung eine entsprechende PARAMETER DISTRIBUTION-Anweisung durchlaufen werden.

Die Side-Information Datei (`Upicfile`)

Diese Datei muß auf der Client-Seite vorhanden sein. Sie enthält die Informationen, die UPIC zum Adressieren der UTM-Anwendung auf Server-Seite braucht. Für jede Server-Anwendung müssen Sie teils über Menü, teils mit einem Editor in der Datei `Upicfile` folgendes festlegen:

– Kennzeichen für ASCII-EBCDIC-Konvertierung

  DRIVE/WINDOWS setzt eine automatische Code-Konvertierung zwischen ASCII und EBCDIC durch das Kommunikationssystem voraus.

  Bei HD führt UPIC eine automatische Code-Konvertierung der Daten durch, die zwischen Client und Server ausgetauscht werden. Dies gilt beim Senden (nach EBCDIC) und beim Empfangen (nach ASCII).

  Bei HD führt UPIC die Code-Konvertierung **immer** durch, auch wenn der Server im SINIX läuft. D.h., Sie müssen das Betriebssystem des Server-Rechners berücksichtigen, wenn Sie das Konvertierungskennzeichen angeben:

  – HD, wenn der Server-Rechner BS2000 ist (für automatische Code-Konvertierung)

  – SD, wenn der Server-Rechner SINIX ist (keine automatische Code-Konvertierung).

– Symbolic Destination Name

  Dieser Name wird aus der Anweisung PARAMETER DISTRIBUTION APPLICATION oder PARAMETER STATIC SIDEINFO oder aus dem Menü **Optionen** von DRIVE/WINDOWS übernommen. Über den Symbolic Destination Name greift UPIC auf die Einträge in der `Upicfile` zu.

– Name der Server-Anwendung

  Mit dieser Angabe adressiert UPIC die UTM-Anwendung im Server (siehe Abschnitt Verteilte Anwendungen generieren in DRIVE-Programmiersystem [1]).

– Transaktionscode des Server-Vorgangs (optional)

  Diese Angabe kann ganz entfallen. DRIVE/WINDOWS übergibt als Client automatisch an UPIC die entsprechende Angabe aus dem CALL-Aufruf:

– Bei Aufruf eines DRIVE-Programms als Server mit Angabe von Bibliothek/Elementnamen ist der Server-Transaktionscode für DRIVE-Interpreter- bzw. DRIVE-Compiler-Betrieb fest vergeben (SQLRMT bzw. DRT#SC## siehe DRIVE-Programmiersystem[1], Abschnitt Verteilte Anwendungen generieren). Sie wählen zwischen Interpreter- und Compilerbetrieb mit der TYPE-Klausel der Anweisung PARAMETER DISTRIBUTION (siehe „DRIVE-Lexikon“ [3]).

– Bei Aufruf eines C/COBOL-Teilprogramms als Server mit CALL TAC geben Sie bei diesem Aufruf den Transaktionscode an (siehe „DRIVE-Lexikon“ [3]).

CALL-Aufruf

In einem DRIVE-Programm als Client rufen Sie Server-Programme mit CALL auf:

– CALL mit Bibliothek/Elementnamen für ein DRIVE-Programm als Server

– CALL TAC für ein anwendereigenes C/COBOL-Teilprogramm.

Beim CALL-Aufruf selbst brauchen Sie nicht zu unterscheiden, ob Sie das Programm lokal oder remote ablaufen lassen wollen. DRIVE/WINDOWS prüft intern, ob Verteilte Anwendungen eingestellt ist (OPTION DISTRIBUTION) und wertet bei gesetzter Option die Verteilungsinformationen aus (PARAMETER DISTRIBUTION). CALL TAC kann im Client-Betrieb, d.h. ohne TP-Monitor, nur remote ausgeführt werden. Fehlt die Verteilungsinformation, so daß das Teilprogramm lokal ausgeführt werden müßte, wird der CALL TAC-Aufruf zum Ablaufzeitpunkt ignoriert.

12.1.2 DRIVE-Anweisungen und Server-Zustände

Dieser Abschnitt beschreibt, welche Server-Zustände durch welche DRIVE-Anweisungen erreicht werden. Für Programmierer, die C-Programme im Client (siehe Seite 333) oder C/COBOL-Teilprogramme im Server anschließen, wird **Spezialinformation** zu den jeweiligen Server-PEND-Aufrufen in eckigen Klammern gegeben.

DRIVE/WINDOWS kann folgende Zustände des Servers unterscheiden:

1. der Server hat sich noch nicht beendet und kann weitere Aufträge empfangen
2. der Server hat sich beendet und kann keine weiteren Aufträge empfangen
3. der Server wurde durch einen Programm-Fehler von UTM beendet.

Der Server hat sich noch nicht beendet und kann weitere Aufträge empfangen

Dieser Zustand wird erreicht, wenn

– im Server-DRIVE-Programm die Anweisung END PROCEDURE auf oberster Server-Programmstufe erreicht oder die Anweisung COMMIT WORK auf beliebiger Programmstufe durchlaufen wurde.

– [**Spezialinformation**: im C/COBOL-Teilprogramm PEND KP oder PEND RE ausgelöst wurde.]

DRIVE/WINDOWS versorgt intern in Server-Umgebung die RETURN-Parameter, die in der USING-Klausel der CALL-Anweisung angegeben wurden, mit den Rückgabewerten des Servers. Der Programmlauf wird im Client-DRIVE-Programm mit der Anweisung fortgesetzt, die auf die CALL-Anweisung folgt.

Ein weiterer Remote-CALL auf den gleichen Server wird über die gleiche logische Verbindung und den noch offenen UTM-Vorgang bearbeitet (UPIC-Aufrufe Send Data und Receive). Ein Remote-CALL auf einen anderen Server weist DRIVE/WINDOWS mit einer Fehlermeldung ab, da über UPIC zu einem Zeitpunkt nur ein einziger Server-Vorgang adressiert werden kann.

Der Server hat sich beendet und kann keine weiteren Aufträge empfangen

Dieser Zustand wird erreicht, wenn

– im DRIVE-Programm die Anweisung STOP oder COMMIT WORK WITH STOP durchlaufen wurde

– das Server-DRIVE-Programm fehlerhaft abgebrochen und eine entsprechende Fehlermeldung an den Client zurückgegeben wurde (siehe WHENEVER...'DIS ERROR', Fehlerklasse D0001 Seite 312).

– [**Spezialinformation**: das Server-C/COBOL-Teilprogramm PEND FI erreicht hat.]

Wurde das Server-Programm fehlerfrei beendet, versorgt DRIVE/WINDOWS intern in Server-Umgebung die RETURN-Parameter, die in der USING-Klausel der CALL-Anweisung angegeben wurden, mit den Rückgabewerten des Servers. Der Programmlauf wird im Client-DRIVE-Programm mit der Anweisung fortgesetzt, die auf die CALL-Anweisung folgt.

Meldet das Server-Programm einen Fehler (z.B. CHECK ERROR im DRIVE-Programm), beendet sich das Server-Programm, die Datenbank wird von DRIVE/WINDOWS zurückgesetzt und die logische Verbindung abgebaut. Im Client-DRIVE-Programm kann der Fehler mit WHENEVER...'DIS ERROR', Fehlerklasse D0001 behandelt werden [**Spezialinformation**: die C-UPIC-Anwendung erhält im Header-Feld Aufruf-Code den Wert E oder P (siehe Seite 316)].

Bei einem weiteren Remote-CALL an denselben oder einen anderen Server wird die logische Verbindung zwischen Client und Server wieder aufgebaut.

Der Server wurde durch einen Programmfehler von UTM beendet

Wird ein Programmfehler von UTM erreicht [**Spezialinformation:** PEND ER oder PEND FR im Server], beendet sich der Server, die logische Verbindung wird abgebaut. Bei einem weiteren Remote-CALL an denselben oder einen anderen Server wird die logische Verbindung zwischen Client und Server wieder aufgebaut. Im Client-DRIVE-Programm kann der Fehler mit WHENEVER...'DIS ERROR' behandelt werden
[**Spezialinformation:** das UPIC-Ergebnis ist DEALLOCATE_ABEND].

Gültigkeit der logischen Verbindung

Ein Client kann über die UPIC-Schnittstelle zu einem Zeitpunkt nur eine einzige Server-Anwendung adressieren. Es wird eine eindeutige logische Verbindung aufgebaut, die erhalten bleibt, bis sich der Server-Vorgang (fehlerfrei oder fehlerhaft) beendet hat oder DRIVE/WINDOWS als Client die logische Verbindung wegen Fehler vorzeitig abbauen muß.

Die logische Verbindung zwischen Client und Server ist bei DRIVE/WINDOWS programmübergreifend gültig. So können zwei verschiedene DRIVE-Programme, die im Client aufgerufen werden, Remote-CALL-Aufträge über die gleiche logische Verbindung an den gleichen Server senden, wenn der Server sich bei den Aufrufen nicht beendet.

Beispiel:

Ruft das DRIVE-Programm A zwei lokale DRIVE-Programme B und C mit CALL auf, können in den Programmen B und C Remote-CALL-Aufrufe des gleichen Servers über die gleiche logische Verbindung gehen, wenn der Server nach dem ersten Aufruf offen bleibt. Dies gilt unabhängig von der Aufrufhierarchie z.B. auch, wenn das Programm C aus B aufgerufen wird.

Ist der Server ein DRIVE-Programm, können permanente Objekte über mehrere Aufrufe weiterbenutzt werden.

DRIVE-Anweisungen und Ende der logischen Verbindung

Die folgenden DRIVE-Anweisungen können Sie im Client-Programm nur angeben, wenn die logische Verbindung zum Server wieder abgebaut ist:

– END PROCEDURE auf der obersten Programmstufe im Client

– STOP, STOP WITH...

– COMMIT WORK WITH STOP [WITH]...

– DO PROCEDURE

Andernfalls setzt DRIVE/WINDOWS intern die logische Verbindung und damit den Server zurück und beendet ihn
[**Spezialinformation**: UPIC-Aufruf Deallocate].
Danach bricht DRIVE/WINDOWS mit Fehler ab.

12.1.3 Transaktionssicherung

Von MS-Windows aus sind folgende Datenbankzugriffe möglich:

1. auf INFORMIX
   - lokal auf INFORMIX-SE
   - remote über datenbankinterne Verteilung von INFORMIX-NET bzw. STAR auf INFORMIX-SE (MS-Windows oder SINIX) und ONLINE im SINIX. Es gibt keine Transaktionssynchronisation zwischen möglicher lokaler und remoter INFORMIX-Datenbank.
   - über die Client-Server-Architektur der verteilten Anwendungen auf INFORMIX-SE oder ONLINE im SINIX. Die Anwendung auf SINIX-Seite ist technisch eine UTM-Anwendung, da das Konzept intern von DRIVE/WINDOWS über die Schnittstellen UPIC und UTM realisiert wird. Dem Programmierer steht die Funktionalität des Betriebsmodus mit TP-Monitor im Server zur Verfügung. Neben der Datenbank im SINIX-Server ist auch eine lokale Datenbank unter MS-Windows möglich. Es gibt aber keine Transaktionssynchronisation zwischen den beiden Datenbanken.
2. auf die BS2000-Datenbanken SESAM und UDS
   - Remote-Zugriff auf die Datenbanken, d.h. die Datenbankzugriffe gehen über das Netz, die gesamte DRIVE-Anwendung liegt auf MS-Windows. Realisiert wird der Remote-Zugriff über die Netzkomponente DRIVE-NET. Neben der remote Datenbank im BS2000 ist auch eine lokale Datenbank unter MS-Windows möglich. Es gibt aber keine Transaktionssynchronisation zwischen den beiden Datenbanken.
   - über die Client- Server-Architektur der verteilten Anwendungen. Die Anwendung auf BS2000-Seite ist technisch eine UTM-Anwendung, da das Konzept intern von DRIVE/WINDOWS über die Schnittstellen UPIC und UTM realisiert wird. Dem Programmierer steht die Funktionalität des UTM-Betriebs im Server zur Verfügung. Neben der Datenbank im BS2000-Server ist auch eine lokale Datenbank unter MS-Windows möglich. Es gibt aber keine Transaktionssynchronisation zwischen den beiden Datenbanken.

Bei verteilten Anwendungen gelten die Sicherungspunkte nur für die jeweiligen lokalen Anwendungen, also z.B. für die DRIVE-SESAM-UTM- oder DRIVE-UDS-UTM-Anwendung im BS2000. Im Client kann DRIVE/WINDOWS die Datenbanktransaktion zusammen mit den DRIVE-Variablen und der Fensteroberfläche mit Inhalten sowie den DRIVE-Programmlauf auf einen gemeinsamen Sicherungspunkt zurücksetzen (Anweisung ROLL-

BACK WORK WITH RESET und PARAMETER STATIC RESTART=ON). Diese Transaktionssteuerung gilt auch für einen Remote-Zugriff auf ein INFORMIX-Datenbanksystem (da dieser intern/transparent über INFORMIX-NET bzw. -STAR verteilt wird).

Zu beachten ist, daß beim internen Rücksetzen einer INFORMIX-Transaktion der dazugehörige Backend-Prozeß beendet wird und DRIVE/WINDOWS mit einem Programmabbruch reagiert.

Die jeweils lokalen Transaktionen von Client und Server werden aber nicht synchronisiert. Bei Verbindungsverlust können in den verschiedenen verteilten Anwendungen inkonsistente Daten vorliegen. Daher bietet der Transaktionsmonitor UTM im BS2000 keinen Vorgangswiederanlauf bei Kommunikation mit UPIC-Anwendungen. Realisiert ist also verteilte Verarbeitung ohne anwendungsübergreifende Transaktionssicherung („cooperative processing“). Diese Form von verteilter Verarbeitung ist z.B. dann sinnvoll, wenn bei strikter Trennung von Präsentation (mit Dialogverarbeitung) und (Daten-) Verarbeitung der Datenbestand in nur einer Anwendung, nämlich auf Server-Seite, verändert wird. Zu Datenverlust von Eingabedaten auf Client-Seite bzw. doppelten Auftragsdaten auf Server-Seite bei wiederholtem Remote-CALL sowie Dateninkonsistenz zwischen Datenbanken auf Client- und Server-Seite siehe Seite 309.

Programmierempfehlungen

Die folgenden Programmierempfehlungen sichern im allgemeinen die Datenkonsistenz zwischen Client und Server.

1. Client- und Server-Programme sollten die jeweiligen COMMIT-Zeitpunkte abstimmen, wenn eine lokale Datenbank angeschlossen ist.

   Hat der Server COMMIT WORK ausgeführt, sollte der Client möglichst unmittelbar nach dem Remote-CALL ebenfalls COMMIT WORK ausführen.

2. Bei Fehlersituationen im Server (z.B. Programmabbruch wegen CHECK ERROR) setzt DRIVE/WINDOWS die Server-Transaktion zurück und meldet die Fehlersituation dem Client (D0001 oder Returncode E im Header siehe Seite 316). Das Client-Programm kann mit WHENEVER ...'DIS ERROR' reagieren, es sollte kein COMMIT WORK ausführen. Bei lokalem Datenbankanschluß und synchronisiertem COMMIT (siehe Punkt 1) sollte ein ROLLBACK WORK ausgeführt werden.

3. Konnte der Server z.B. wegen Netzproblemen nicht adressiert werden, kann der Client diese Fehlersituation mit WHENEVER...'DIS ERROR'abfragen (D0003). Der Client sollte dann kein COMMIT WORK ausführen. Bei lokalem Datenbankanschluß und synchronisiertem COMMIT (siehe Punkt 1) sollte ein ROLLBACK WORK ausgeführt werden.

4. Im Client sollte die Anweisung ROLLBACK WORK WITH RESET (Voraussetzung: PARAMETER STATIC RESTART=ON) möglichst vermieden werden, damit durch das Rücksetzen des Programmlaufs im Client auf den letzen Sicherungspunkt Remote-CALLs nicht mehrmals ausgeführt werden (siehe unten).
5. Datenverlust von Eingabedaten auf Client-Seite möglich, da möglichst kein ROLLBACK WORK WITH RESET gegeben werden soll, d.h. dann kein Bildschirm-Wiederanlauf der Fenster durch DRIVE-Transaktionen stattfindet (siehe Kapitel „Transaktionskonzept“ auf Seite 261).

Dateninkonsistenz zwischen Client und Server in Sonderfällen

In folgenden Sonderfällen ist die Datenkonsistenz zwischen Client und Server nicht gesichert:

1. Tritt nach einem Remote-CALL im Client eine Fehlersituation auf ohne lokalen Datenbankanschluß, so können die Daten im Server bei einem dortigen COMMIT bereits festgeschrieben sein. Der Endanwender kann die eingegebenen Daten nur durch eine neue Abfrage der Datenbank überprüfen.

   Bei lokalem (oder datenbankintern verteiltem remote) Datenbanksystem INFORMIX wird im Client die Datenbank-Transaktion zurückgesetzt. Damit beendet sich auch das INFORMIX-Backend und DRIVE/WINDOWS reagiert mit Programmabbruch. Der Server hat jedoch seine lokalen Daten bereits mit COMMIT WORK festgeschrieben. Mit der ersten Programmierempfehlung, die jeweiligen COMMIT-Zeitpunkte in Client und Server abzustimmen, können Sie dieses Problem umgehen.

2. An der UPIC-Schnittstelle können folgende Situationen nicht unterschieden werden:

   – UTM-Server-Vorgang oder UTM-Anwendung wurde wegen eines Systemfehlers beendet (PEND ER/FR)

   – Verbindungsabbau durch das Transportsystem.

   Sie können diese beiden Fälle im Client mit WHENEVER...'DIS ERROR', Fehlerklasse D0005 abfragen. In der Regel hat der Server-Vorgang seine lokale Datenbank-Transaktion nicht festgeschrieben und wurde von UTM zurückgesetzt. Im ungünstigsten Fall eines Verbindungsabbaus durch das Transportsystem, nachdem der Server bereits COMMIT WORK ausgeführt hat (also quasi Netzausfall beim Rücksenden der Server-Quittung an den Client), ist die lokale Datenbank-Transaktion des Servers festgeschrieben.

Transaktionsanweisungen im Client

Da die jeweiligen lokalen Transaktionen im Client und Server nicht synchronisiert werden, wirken sich Transaktionsanweisungen im Client nicht auf Server-Transaktionen aus.

Anweisungen COMMIT WORK [WITH STOP] [WITH send message]

Diese Anweisungen beenden die Transaktion im Client. DRIVE/WINDOWS prüft nicht den Zustand der Server-Transaktion.

Die Anweisung COMMIT WORK WITH STOP beendet neben der lokalen Datenbank-Transaktion auch die aktuelle DRIVE-Session. Es darf kein Server mehr adressiert sein.

Anweisung ROLLBACK WORK

Diese Anweisung setzt die lokale Datenbank-Transaktion im Client zurück. DRIVE/WINDOWS prüft nicht den Zustand der Server-Transaktion, diese wird nicht zurückgesetzt.

Beispiel 1

Im Client wird ein COMMIT WORK ausgeführt und anschließend durch SQL-Anweisungen eine neue lokale Datenbank-Transaktion eröffnet. Dann wird ein Server mit Remote-CALL aufgerufen. Der Server öffnet ebenfalls eine lokale Datenbank-Transaktion, schließt diese mit COMMIT WORK ab und beendet seinen Programmlauf. Die Server-Transaktion ist damit bereits festgeschrieben.
Im Client wird ein ROLLBACK WORK ausgeführt, d.h. die (lokale oder datenbankintern verteilte remote) INFORMIX-Transaktion auf den letzten Sicherungspunkt im Client zurückgesetzt.
Die Server-Transaktion kann jedoch nicht mehr zurückgesetzt werden.
Es wird daher empfohlen, die COMMIT WORK-Zeitpunkte von Client und Server zu koordinieren (siehe Programmierempfehlungen Seite 308).

Beispiel 2

Der Client ruft per Remote-CALL einen Server und wartet im UPIC-Aufruf Receive. Tritt beim Server oder im Netz ein Fehler auf, schickt UPIC die Rückmeldung DEALLOCATE_ABEND an den Client. Mit dieser Meldung werden folgende Situationen zusammengefaßt:

- Ungültiger Server-Transaktionscode
- UTM-Vorgang oder UTM-Anwendung wurde beendet
- Verbindung wurde von UTM-Administration abgebaut
- Verbindung wurde vom Transportsystem abgebaut.

In den ersten drei Fällen wird der Server-Auftrag nicht mit Transaktionsende abgeschlossen, d.h. Änderungen lokaler Daten werden nicht festgeschrieben. Im vierten Fall ist auf Client-Seite nicht klar, ob die Netzprobleme vor dem Aktivieren des Servers oder erst beim Rücksenden der Server-Quittung, also nach dem COMMIT WORK im Server aufgetreten sind und der Server die Änderung seiner lokalen Daten festschreiben konnte.

Anweisung ROLLBACK WORK WITH RESET

Wie im lokalen/nicht verteilten Betrieb setzt diese Anweisung die (lokale oder datenbankintern verteilte remote) Datenbank-Transaktion bei INFORMIX zurück sowie die DRIVE-Variablen, Fenster der Bedienoberfläche und Inhalte. Voraussetzung: PARAMETER STATIC RESTART=ON ist gesetzt. Der Programmlauf setzt beim letzten Sicherungspunkt (COMMIT WORK) fort.

Setzt INFORMIX eine Transaktion intern zurück, wird auch der dazugehörige Backend-Prozeß beendet und DRIVE/WINDOWS muß mit einem Programmabbruch reagieren. D.h. die Situation TA_CANCELLED ist im Client bei INFORMIX-Anschluß nicht möglich.

Folgende Fälle sind für die Client-Server-Kommunikation zu unterscheiden je nachdem, welche Server adressiert sind zum Zeitpunkt des ROLLBACK WORK WITH RESET im Client und des COMMIT WORK im Server:

1. Fall

RESET-Zeitpunkt	es ist kein Server adressiert
COMMIT-Zeitpunkt	es ist kein Server adressiert

Das Client-Programm wird hinter COMMIT WORK fortgesetzt.

2. Fall

RESET-Zeitpunkt	es ist Server 1 adressiert
COMMIT-Zeitpunkt	es ist kein Server adressiert

Server1 wird durch den UPIC-Aufruf Deallocate zurückgesetzt und beendet. Das Client-Programm wird hinter COMMIT WORK fortgesetzt.

3. Fall

RESET-Zeitpunkt	es ist kein Server adressiert
COMMIT-Zeitpunkt	es ist Server 1 adressiert

Da Server1 nicht in seinem ursprünglichen Zustand wiederhergestellt werden kann, wird das Client-Programm abgebrochen (Meldung Wiederanlauf nicht möglich).

4. Fall

RESET-Zeitpunkt	es ist Server 1 adressiert
COMMIT-Zeitpunkt	es ist Server 1 adressiert

Das Client-Programm wird hinter COMMIT WORK fortgesetzt.

5. Fall

RESET-Zeitpunkt	es ist Server 1 adressiert
COMMIT-Zeitpunkt	es ist Server 2 adressiert

Da Server 2 nicht in seinem ursprünglichen Zustand wiederhergestellt werden kann, wird das Client-Programm abgebrochen (Meldung Wiederanlauf nicht möglich).

In den Fällen 1, 2 und 4 kann das Rücksetzen des Programmlaufs im Client ein mehrmaliges Ausführen des gleichen Remote-CALLs bewirken, d.h. Auftragsvervielfältigung. Im Client-Betrieb sollte daher die Anweisung ROLLBACK WORK WITH RESET möglichst vermieden werden (siehe Programmierempfehlungen Seite 308).

12.1.4 Fehlersituationen

Bei einem fehlerhaften Client-DRIVE-Programm (z.B. CHECK ERROR ohne WHENEVER...CONTINUE oder CALL) wird ein noch adressierter Server intern von DRIVE/WINDOWS zurückgesetzt und beendet [**Spezialinformation**: UPIC-Aufruf Deallocate].

Bei Fehler im Client außerhalb des Anwenderprogramms (z.B.Warmstart) kann der Server-Vorgang entweder durch Timer zurückgesetzt werden oder durch einen Administrator-Eingriff, um den Server-Vorgang für denselben User zugänglich zu machen. Administrations-Maßnahme:

– `KDCINF USER, L=KDCCON`

  zeigt die User an, die mit der Anwendung verbunden sind

– `KDCLTERM ACT=DIS, LTERM=<lterm-name>`

  baut die logische Verbindung zwischen dem Client und der UTM-Anwendung ab.

Die Systemvariable &DISTRIBUTION_STATE wird nur belegt, wenn beim Ausführen eines Remote-CALL ein Fehler auftritt. Bei fehlerfrei ausgeführten Remote-CALLs wird diese Systemvariable inhaltlich nicht verändert. Bei lokalen CALL-Anweisungen wird die Fehlerbehandlung mit WHENEVER...'DIS ERROR' nicht ausgewertet.

Bei Fehlersituationen im Server wird im Client die Systemvariable &DISTRIBUTION_STATE.DIS_ERROR mit dem entsprechenden Inhalt der Server-Systemvariable &ERROR_STATE.ERROR versorgt. Entsprechendes giilt für die Komponenten DIS_DML_STATE bis DIS_SATEMENT. Detaillierte Informationen werden in der Systemvariablen &ERROR_STATE.SUB_CODE und &DISTRIBUTION_STATE hinterlegt (siehe Abschnitt „Systemvariablen“ auf Seite 20). Sie können also im Client-DRIVE-Programm mit der Anweisung WHENEVER...'DIS ERROR' diese Fehlersituationen abfragen und entsprechend reagieren, falls der Server eigene Fehlersituationen nicht selbst abfängt.

In der Systemvariablen **&ERROR_STATE.SUB_CODE** wird die entsprechende Fehler-Unterklasse (Dxxxx) hinterlegt:

- D0001 - Server meldet einen Programm-Fehler zurück

  Mögliche Ursachen:

  Bei einem DRIVE-Programm z.B. CHECK ERROR, bei einem C/COBOL-Teilprogramm der Returncode E im Header (siehe Seite 316).

  Maßnahme:

  Systemvariable &DISTRIBUTION_STATE auswerten

- D0002 - Einträge in der `Upicfile` nicht korrekt

  Mögliche Ursachen:

  – Symbolic Destination Name ungültig (siehe APPLICATION-Klausel bei PARAMETER DISTRIBUTION oder PARAMETER STATIC SIDEINFO oder Menü **Parameter**)

  – falsches Konvertierungskennzeichen (SD bei SINIX-Server, HD bei BS2000-Server siehe Seite 303)

  – Partner-Name nicht im TNS generiert (siehe DRIVE-Programmiersystem [1], Abschnitt Verteilte Anwendungen generieren)

  Maßnahme:

  Entsprechende Einträge in der `Upicfile` machen.

- D0003 - Verbindung zum Server konnte nicht aufgebaut werden

  Maßnahmen:

  – UTM-Anwendung des Server hochfahren

  – PTERM-Generierung beim UTM-Server mit TNS-Einträgen im Client in der Datei DIR1 für jeden User (PARAMETER STATIC USER) vergleichen (siehe DRIVE-Programmiersystem [1], Abschnitt Verteilte Anwendungen generieren)

- D0004 - Vorübergehender Betriebsmittelengpaß bei der Kommunikation

  Mögliche Ursachen:

  – Verbindung zum Server ist nicht zustande gekommen

  – Verbindung zum Server wurde beendet

  Maßnahme:

  Nochmaliger Remote-CALL-Aufruf auf den gleichen Server-Vorgang ist möglich. Zu beachten ist jedoch, daß der Server-Vorgang ggf. von UTM zurückgesetzt und beendet wurde.

- D0005 - Abbruch der Verbindung zum Server

  Mögliche Ursachen:

  – Server-Anwendung wurde durch UTM beendet

  – Verbindung zum Server wurde durch die UTM-Administration abgebaut

  – Verbindung zum Server wurde durch das Transportsystem abgebaut

  – PEND RS/ER/FR im UTM-Server-Vorgang

  – ungültiger Transaktionscode beim Senden

  Bedeutung: Der Server-Vorgang wurde zurückgesetzt und beendet.

  Maßnahmen:

  – Abbruch-Ursache im Server überprüfen

  – TAC-Angabe bei CALL TAC oder Generierung der Server-Anwendung überprüfen

DRIVE/WINDOWS hinterlegt in der Systemvariablen **&DISTRIBUTION_STATE** Informationen über den Server, der den Fehler verursacht hat. Die Variable ist folgendermaßen deklariert:

```
1 &DISTRIBUTION_STATE PERMANENT,
  2 DIS_ERROR            CHAR(16)    INIT 'OK',
  2 DIS_DML_STATE        CHAR(16)    INIT 'OK',
  2 DIS_SQL_CODE         INTEGER     INIT 0,
  2 DIS_SQL_STATE,
     3 DIS_SQL_CLASS     CHAR(2)     INIT '00'
     3 DIS_SUB_CLASS  CHAR(3)     INIT '000',
  2 DIS_STATEMENT        CHAR(16)    INIT '',
```

```
  2 DIS_SUB_CODE         CHAR(8)     INIT '',
  2 DIS_LIBRARY          CHAR(54)    INIT '',
  2 DIS_ELEMENT          CHAR(31)    INIT '',
  2 DIS_TACNAME          CHAR(8)     INIT '',
  2 DIS_APPLICATION      CHAR(8)     INIT '';
```

Sind Client und Server DRIVE-Programme, werden die Komponenten DIS_ERROR, DIS_DML_STATE, DIS_SQL_CODE, DIS_SUB_CODE, DIS_SQL_CLASS sowie DIS_STATEMENT und DIS_SUB_CLASS bei Auftreten der Fehlerklasse D0001 mit den Werten der Server-Systemvariablen &ERROR_STATE versorgt. Bei den Fehlerklassen D0002, D0003, D0004 und D0005 bleiben diese Komponenten inhaltlich unverändert.

Die Komponenten DIS_LIBRARY, DIS_ELEMENT, DIS_TACNAME, DIS_APPLICATION werden bei jeder Fehlerklasse mit den entsprechenden Werten der Verteilungsinformationen für den fehlerhaften Server versorgt, wie sie mit PARAMETER DISTRIBUTION gesetzt wurden (Seite 301).

12.1.5 Parameter übergeben zwischen Client und Server

Sind Client und Server DRIVE-Programme (Aufruf CALL bibliothek/elemname), können folgende USING-Parameter übergeben werden (siehe auch DRIVE-Lexikon [3]):

– Literale

– Variablen (einfache Variablen, Vektoren, Matrizen, Strukturen)

– Ausdrücke

– Aggregate

Das DRIVE-RTS (Runtime-System) übernimmt vollständig die Ablage und Auswertung der Daten. Die Ablageform und die Art der Konvertierung wird über interne Formate geregelt. Der USING-Bereich darf max. 32000 Bytes groß sein (USING-Daten und Beschreibungsdaten).

Bei der Kopplung eines DRIVE-Programms mit einem C/COBOL-Teilprogramm (Aufruf CALL TAC) können als USING-Paramter nur Variablen übergeben werden. Diese dürfen beliebig strukturiert sein (einfache Variablen, Vektoren, Matrizen, Strukturen). Die Daten werden in netzinvarianter Form übergeben. Allen Nachrichten, die zwischen Client und Server ausgetauscht werden, wird ein Header vorgeschaltet (siehe Seite 316).

12.1.6 Datenübergabe bei UTM-Teilprogrammen in C/COBOL im Server (für C/COBOL-Programmierer)

Zum Nachrichtenaustausch zwischen einem anwendereigenen UTM-Teilprogramm in C/COBOL und einem DRIVE-Programm wird ein Übergabebereich verwendet. Mögliche Kopplungen sind ein DRIVE-Client mit einem C/COBOL-Server oder ein C-Client mit einem DRIVE-Server (siehe Abschnitt „Datenkonvertierung bei C-Programmen als Client und DRIVE als Server“ auf Seite 336).

Die Daten im Übergabebereich liegen in netzinvarianter Form (abdruckbar) vor. Beim Nachrichtenaustausch zwischen BS2000- und MS-Windows-Anwendungen setzt UPIC zwischen EBCDIC und ASCII um und umgekehrt.
In den anwendereigenen UTM-Teilprogrammen in C/COBOL müssen Sie bei MPUT-Anweisungen immer das Formatkennzeichen mit Leerzeichen belegen.

Der Übergabebereich besteht aus zwei Teilen:

– Header-Informationen, die immer vorhanden sind und eine feste Länge haben

– USING-Daten beim Übergeben von Parametern bzw. Diagnosemeldungen.

12.1.6.1 Header-Informationen im Datenübergabebereich

Der Header hat eine feste Länge von 144 Byte. Alle Informationen im Header werden in abdruckbarer Form hinterlegt. Dies gilt auch für alle numerischen Informationen (Längenfelder). Die folgende Tabelle zeigt das Layout des Header und die Länge der einzelnen Header-Felder für die Kommunikation von DRIVE/WINDOWS mit einem anwendereigenen UTM-Teilprogramm in C/COBOL.

Header-Feld	**Längen in Byte bei C**	**Längen in Byte bei COBOL**
1) Versionskennzeichen	9	9
2) Name des Folgetac bei PEND PA [1]	9	9
3) Aufruf-Code	1	1
4) Direktive (DRIVE/Benutzer)	1	1
5) Gesamtlänge des Übergabereichs	7	7
6) Länge des Bibliotheksnamens	7	7
7) Bibliotheksname	55	55
8) Länge des Elementnamens	7	7

Header-Feld	**Längen in Byte bei C**	**Längen in Byte bei COBOL**
9) Elementname	32	32
10) Reservefeld	16	16
[1] Für die Client-Seite nicht relevant		

Bei der Kommunikation von DRIVE/WINDOWS mit einem UTM-Teilprogramm in C/COBOL ist der Wert in jedem Header-Feld mit Ausnahme von Aufruf-Code (3) und Direktive (4) mit einem Nullbyte abgeschlossen.

Bei den Längenfeldern 5), 6) und 8) wird der numerische Längenwert abgelegt wie beim DRIVE-Datentyp SMALLINT für USING-Daten im Übergabebereich (siehe Abschnitt „Datenkonvertierung bei DRIVE als Client und C/COBOL-Teilprogrammen als Server“ auf Seite 325 und Abschnitt „Datenkonvertierung bei C-Programmen als Client und DRIVE als Server“ auf Seite 336).

Bedeutung der Header-Felder:

1) Versionskennzeichen muß versorgt werden mit DRIVTV01

2) für die Client-Seite nicht relevant

3) Aufruf-Code

   – Für die Verbindung vom Client zum Server ist der Wert S als Standard-Aufruf erlaubt, um normale Programmverarbeitung im Server-Vorgang zu starten

   – Für die Verbindung vom Server zum Client sind folgende Werte erlaubt:

     – O: Aufruf wurde fehlerfrei beendet.

     – E: Aufruf wurde mit Fehlern im Programmablauf beendet.

     – P: Der Header wurde nicht korrekt versorgt (Parameterfehler), der Aufruf ist fehlerhaft.

4) Direktive

   Das Feld gibt an, mit welchem Programmtyp DRIVE/WINDOWS kommuniziert und welches Layout der Übergabebereich hat. Der Wert ist C für C-Programme und S für COBOL-Programme oder C-Programme, die USING-Daten in fester Länge verarbeiten (siehe Abschnitt „USING-Daten im Datenübergabebereich“ auf Seite 318).

5) Gesamtlänge des Übergabebereichs

Die Gesamtlänge des Übergabebereichs schließt die Länge des Header mit ein und ist abhängig von der Generierung der Server-UTM-Anwendung:
Länge des SPAB minus 50 Byte für interne Verwaltung (siehe MAX-Anweisung bei KDCDEF-Steueranweisungen im Beispiel für KDCDEF-Rahmen in DRIVE-Programmiersystem [1], Abschnitt Verteilte Anwendungen generieren).

Ist die Gesamtlänge des Übergabebereichs größer als die Länge des Header, so schließen sich im Übergabebereich folgende Daten unmittelbar an den Header an in der Länge, die sich aus der Differenz von Gesamtlänge minus Headerlänge ergibt:

– bei Aufruf-Code S: USING-Daten des Clients an den Server
– bei Aufruf-Code O: USING-Daten für RETURN-Parameter des Servers zurück an den Client
– bei Aufruf-Code E oder P: Diagnosemeldungen des Servers an den Client

Die folgenden Felder sind nur bei Aufruf-Code S relevant, wenn ein DRIVE-Programm als Server aufgerufen werden soll.

6) Länge des Bibliotheksnamens

Gibt die Länge des Bibliotheksnamens an und darf Werte von 0 bis 54 enthalten.

7) Bibliotheksname

Name der DRIVE-Bibliothek. Die Länge des Namens muß im Feld 6 linksbündig hinterlegt sein.

8) Länge des Elementnamens

Gibt die Länge des Elementnamens an und darf Werte von 1 bis 31 enthalten.

9) Elementname

Name des Elements für das DRIVE-Programm. Die Länge des Namens muß im Feld 8 linksbündig hinterlegt sein.

10) Reservefeld, wird zur Zeit nicht verwendet.

12.1.7 USING-Daten im Datenübergabebereich

Zwischen DRIVE/WINDOWS und einem anwendereigenen UTM-Teilprogramm in C/COBOL können nur Variablen, und zwar auch Vektoren, Matrizen und strukturierte Variablen als USING-Parameter übergeben werden.

Die USING-Daten des Übergabebereichs sind nur erforderlich, falls USING-Parameter angegeben werden. Datenwerte werden zwischen DRIVE/WINDOWS und anwendereigenen UTM-Teilprogrammen in C/COBOL grundsätzlich in abdruckbarer Form ausgetauscht. Dies gilt auch für numerische Werte.

DRIVE/WINDOWS übergibt Dezimalzeichen an ein anwendereigenes UTM-Teilprogramm in C/COBOL immer als „.“. In umgekehrter Richtung können anwendereigene UTM-Teilprogramme in C/COBOL Dezimalzeichen an DRIVE/WINDOWS entweder als „.“ oder als „,“ übergeben.

Verbindung DRIVE-Programme - anwendereigene UTM-Teilprogrammen in C und umgekehrt:

Datenwerte werden abhängig vom Wert in signifikanter Länge übergeben bzw. erwartet und grundsätzlich mit einem Nullbyte (X’00’) abgeschlossen. Numerische Werte werden bei Zielsprache C mit Vorzeichen linksbündig, Datenwert linksbündig, unmittelbar gefolgt von einem Nullbyte übergeben. Das Vorzeichen wird nur bei negativen Werten übergeben. Indikatorwerte werden mit einem Nullbyte übergeben. Umgekehrt müssen Sie bei Zielsprache DRIVE/WINDOWS Indikatorwerte mit Nullbyte abschließen. Bei der Indikatoranzeige für den NULL-Wert wird kein Datenwert übergeben: Auf den Indikatorwert abgeschlossen mit Nullbyte folgt unmittelbar ein Nullbyte für den leeren Wert.

Da zwischen DRIVE/WINDOWS und anwendereigenen UTM-Teilprogrammen in C der Datenaustausch immer mit Nullbyte abgeschlossen wird, können Sie bei allen Datentypen die C-Bibliotheksfunktionen einsetzen.

Verbindung DRIVE-Programme - anwendereigene UTM-Teilprogramme in COBOL und umgekehrt:

Die Datenwerte werden abhängig vom Datentyp in fester Länge übergeben bzw. erwartet. Alle Daten folgen unmittelbar aufeinander, sie sind nicht durch Nullbyte getrennt. Die Datenwerte werden mit Vorzeichen linksbündig, Datenwert rechtsbündig (ggf. mit führenden Nullen aufgefüllt) übergeben und erwartet. Das Vorzeichen wird immer übergeben. Auch bei der Indikatoranzeige für den NULL-Wert wird, abhängig vom Datentyp, in fester Länge Platz reserviert. Der eigentliche Datenwert ist undefiniert.

Übergabeform und -länge

Die Übergabeform und -länge von Datenwerten ist abhängig vom Datentyp und der Programmiersprache des anwendereigenen UTM-Teilprogramms (C oder COBOL). Die folgende Tabelle zeigt Übergabeform und -länge am Beispiel, eine ausführliche Darstellung finden Sie in Abschnitt „Datenkonvertierung bei DRIVE als Client und C/COBOL-Teilprogrammen als Server“ auf Seite 325 und Abschnitt „Datenkonvertierung bei C-Programmen als Client und DRIVE als Server“ auf Seite 336)

.

DRIVE-Datentyp Beispiel-Wert	Übergabelänge in Byte (ohne Indikatorwertlänge)	Übergabe ohne/ mit Indikator	Übergabeformat des Beispielwerts
SMALLINT 47	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	"47#" "00#47#" "-1##"
	COBOL: 6	ohne Indikator mit Indikator: NOT NULL NULL	"+00047" "00+00047" "-1....."
INTEGER 47	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	"47#" "00#47#" "-1##"
	COBOL: 11	ohne Indikator mit Indikator: NOT NULL NULL	"+0000000047" "00+0000000047" "-1....."
REAL 2345.67	C: wertabhängig (wegen Vorzeichen)	ohne Indikator mit Indikator: NOT NULL NULL	"2.345670E+003#" Mantisse 7-stellig "00#2.345670E+003#" Exponent -stellig "-1##"
	COBOL: 13	ohne Indikator mit Indikator: NOT NULL NULL	"+2.345670E+003" "00+2.345670E+003" "-1....."
FLOAT 2345.67123	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	"+2.34567123000000E+003#" Mantisse 15-stellig "00#2.34567123000000E+003# " Exponent -stellig "-1##"
	COBOL: 21	ohne Indikator mit Indikator: NOT NULL NULL	"+2.34567123000000E+003" "00+2.34567123000000E+003" "-1....."

DRIVE-Datentyp Beispiel-Wert	Übergabelänge in Byte (ohne Indikatorwertlänge)	Übergabe ohne/ mit Indikator	Übergabeformat des Beispielwerts
NUM/ DEC/ XDEC 234 in DEC(7,0) 234.56 in DEC(7,3)	C: wertabhängig	s = 0 ohne Indikator mit Indikator: NOT NULL NULL	 "234#" "00#234#" " -1##"
		s > 0 ohne Indikator mit Indikator: NOT NULL NULL	 "234.560#" "00#234.560#" "-1##"
		p = s ohne Indikator mit Indikator: NOT NULL NULL	 "0.12345#" "00#0.12345" "-1##"
	COBOL: p + 1 bei s = 0	s = 0 ohne Indikator mit Indikator: NOT NULL NULL	 "+0000234" "00+0000234" "-1........"
0,12345 in DEC(5,5) Bsp. DEC(5,5) Wert: 0,12345	p + 2 bei s > 0	s > 0 ohne Indikator mit Indikator: NOT NULL NULL	 "+234.560" "00+0234.560" "-1........."
	p + 3 bei p = s p = s Typen, die nur Nachkommastellen enthalten	p = s ohne Indikator mit Indikator: NOT NULL NULL	 "+0.12345" "00+0.12345" "-1........"

DRIVE-Datentyp Beispiel-Wert	Übergabelänge in Byte (ohne Indikatorwertlänge)	Übergabe ohne/ mit Indikator	Übergabeformat des Beispielwerts
CHAR(n) 'abc' in CHAR(5)	C: n+1 *	ohne Indikator mit Indikator: NOT NULL NULL	"abc␣␣ #" "00abc␣␣ #" "-1##"
	COBOL: n	ohne Indikator mit Indikator: NOT NULL NULL	"abc␣␣" "00abc␣␣" "-1....."
VARCHAR(n) 'abc'	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	"abc#" "00#abc#" "-1##"
	COBOL: akt. Länge + 6 bei COBOL-Verbindung ist immer ein Längenfeld vorhanden oder notwendig	ohne Indikator mit Indikator: NOT NULL NULL	"+00003abc" "00+00003abc" "-1+00000"
DATE 19.01.96	C: 11 *	ohne Indikator mit Indikator: NOT NULL NULL	"1996-01-19#" "00#1996-01-19#" "-1##"
	COBOL: 10	ohne Indikator mit Indikator: NOT NULL NULL	"1996-01-19" "001996-01-19" "-1....."
TIME 1 Sek. vor Mit- ternacht	C: 9 *	ohne Indikator mit Indikator: NOT NULL NULL	"23:59:59#" "00#23:59:59#" "-1##"
	COBOL: 8	ohne Indikator mit Indikator: NOT NULL NULL	"23:59:59" "0023:59:59" "-1....."

DRIVE-Datentyp Beispiel-Wert	Übergabelänge in Byte (ohne Indikatorwertlänge)	Übergabe ohne/ mit Indikator	Übergabeformat des Beispielwerts
TIME(3) 1Tausendstel- Sekunde vor Mitternacht	C: 13 *	ohne Indikator mit Indikator: NOT NULL NULL	"23:59:59:999#" "00#23:59:59:999#" "-1##"
	COBOL: 12	ohne Indikator mit Indikator: NOT NULL NULL	"23:59:59:999" "0023:59:59:999" "-1....."
TIMESTAMP(3) Datum 19.1.96 Zeit 19.45.33 50 Tausendstel- Sekunden	C: 24 *	ohne Indikator mit Indikator: NOT NULL NULL	"1996-01-19 19:45:33.050#" "00#1996-01-19 19:45:33.050#"
	COBOL: 23	ohne Indikator mit Indikator: NOT NULL NULL	"1996-01-19 19:45:33.050" "001996-01-19 19:45:33.050" "- 1....."
INTERVAL		Übergabeformat wie bei INTEGER [1]	

Legende
- p = stellenanzahl
- s = nachkommastellen
- # = Nullbyte-Inhalt
- ... = undefinierter Inhalt
- * Werte vom Datentyp CHAR, DATE, TIME, TIME(3) u. TIMESTAMP(3) werden bei Verbindung mit C immer in fester Länge übergeben bzw. erwartet. Eventuelle Nullbytes in den Datenwerten werden übertragen (bei CHAR) oder führen zu CONVERSION ERROR.
- [1] d.h. Maximallänge 11 Zeichen. Im Client-Programm wird der Datentyp INTERVAL als XDEC-Datentyp abgelegt, so daß ihm ein Wert > max. INTEGER zugewiesen werden kann. Die Daten werden aber an das Server-Programm im Format INTEGER übergeben so wie bei Datenübergabe an lokale 3-GL-Unterprogramme.

Indikatorwerte werden bei C und COBOL in fester Länge von 2 Byte abgelegt.

12.1.7.1 Aufbau der Übergabeparameter

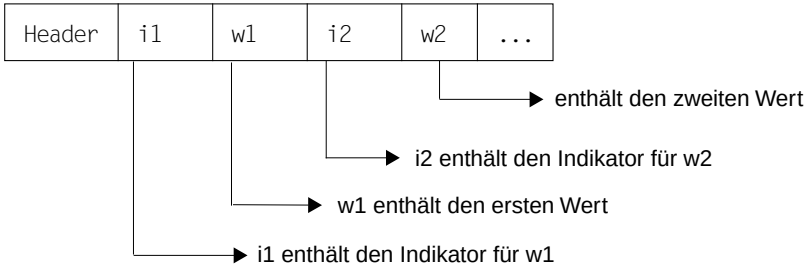
Jeder Übergabebereich für die Verbindung zwischen DRIVE/WINDOWS und einem anwendereigenen UTM-Teilprogramm in C/COBOL und umgekehrt beginnt mit dem Header. Unmittelbar hinter diesem liegen die eigentlichen Datenwerte und ggf. Indikatoranzeigen für den NULL-Wert. Die Datenwerte sind in abdruckbarer Form abgelegt (zur Ablageform der einzelnen Datentypen siehe Seite 325 und Abschnitt „Datenkonvertierung bei C-Programmen als Client und DRIVE als Server“ auf Seite 336).

Ablageform der Indikatorwerte

Indikatorwerte zeigen an, ob der folgende Datenwert ein NULL-Wert ist oder nicht. Sie werden in 2-Byte-Feldern in abdruckbarer Form unmittelbar vor dem eigentlichen Datenwert abgelegt. Dies gilt auch bei Vektoren, Matrizen und Strukturen: Der Indikatorwert für jede Komponente wird unmittelbar vor dem Komponentenwert abgelegt. Der Wert C’-1’ zeigt den NULL-Wert an, d.h. das unmittelbar folgende Feld enthält keinen signifikanten Datenwert. Der Indikatorwert C’00’ zeigt an, daß das folgende Feld keinen NULL-Wert , sondern einen signifikaten Datenwert enthält. Ist ein Indikatorfeld vorhanden, muß es mit einem dieser beiden Werte versorgt sein, sonst meldet DRIVE/WINDOWS „CONVERSION ERROR“.

Ist der Client ein DRIVE-Programm, so können Sie beim Aufruf eines anwendereigenen UTM-Teilprogramms in C/COBOL mit CALL TAC die Indikator-Klausel angeben (siehe „DRIVE-Lexikon“ [3] und Abschnitt „Daten an Unterprogramme in anderen Programmiersprachen übergeben“ auf Seite 112). DRIVE/WINDOWS übergibt Indikatorwerte nur für so spezifizierte USING-Parameter und erwartet beim Rücksprung Indikatorwerte auch nur bei den entsprechenden RETURN-Parametern.

Beispiel



Sind RETURN-Parameter angegeben, so müssen Sie beim Rücksprung vom Server zum Client die USING-Daten für die RETURN-Parameter versorgen, wenn der Server ein UTM-Teilprogramm in C/COBOL ist. Ist der Server ein DRIVE-Programm, versorgt DRIVE/WINDOWS die USING-Daten für die RETURN-Parameter automatisch.

12.1.7.2 Datenkonvertierung bei DRIVE als Client und C/COBOL-Teilprogrammen als Server

```
CALL  ... TAC ... USING &r1,
                        RETURN &r2,
```

Ist das DRIVE-Programm Client und ein UTM-Teilprogramm in C/COBOL Server, konvertiert DRIVE/WINDOWS die numerischen Werte in abdruckbare Form. Mit der INDIKATOR-Klausel wird für jeden USING-Parameter festgelegt, ob vor dem Datenwert (bei Vektoren, Matrizen und Strukturen vor jedem Komponentenwert) ein Indikatorwert abgelegt wird. Das UTM-Teilprogramm kann ggf. die abdruckbaren Werte in interne Rechenformate übertragen.

Auf dem Rückweg muß das UTM-Teilprogramm RETURN-Werte aus evtl. internen Rechenformaten in abdruckbare Form konvertieren. Bei jedem RETURN-Parameter, der mit INDIKATOR-Klausel in der DRIVE-CALL-Anweisung angegeben ist, erwartet DRIVE/WINDOWS vor dem Datenwert (bei Vektoren, Matrizen und Strukturen vor dem Komponentenwert) einen Indikatorwert. Das DRIVE-RTS (Runtime-System) kennt die Zieldatentypen der RETURN-Werte und konvertiert die abdruckbare Form in die interne DRIVE-Form.

Ablageform der einzelnen Datentypen

Die Übergabeform und -länge von Datenwerten ist abhängig vom Datentyp und der Programmiersprache des anwendereigenen UTM-Teilprogramms (C oder COBOL).

CHAR(n)

wird bei COBOL in Länge n übertragen, bei C in Länge (n+1).

VARCHAR(n)

wird in signifikanter Länge übertragen. Die Ablagefrom ist abhängig von der Programmiersprache:

C: String in signifikanter Länge, mit Nullbyte abgeschlossen

COBOL: Längenfeld (abdruckbar, Format wie bei SMALLINT), gefolgt von Wert. Sie können also Datenmengen in variabler Länge übertragen.

Verwenden Sie für COBOL-Anwendungen, die USING-Daten mit fester Länge erwarten, CHAR-Felder statt VARCHAR-Felder. Verwenden Sie für C-Anwendungen VARCHAR-Felder. Damit können Sie den Pointer jeweils um die C-String-Länge (strlen) +1 erhöhen.

INTEGER, INTERVAL

wird als Zahl übergeben, die aus Vorzeichen und einer ganzzahligen Ziffernfolge besteht.

SMALLINT

wird als Zahl übergeben, die aus Vorzeichen und einer ganzzahligen Ziffernfolge besteht.

REAL

wird als Zahl übergeben, die aus Vorzeichen und einer 7-stelligen Ziffernfolge besteht mit Dezimalpunkt, gefolgt von e oder E, gefolgt von einem Vorzeichen und abschließend einer ganzen Zahl.

FLOAT

wird als Zahl übergeben, die aus Vorzeichen und einer 15-stelligen Ziffernfolge besteht, gefolgt von e oder E, gefolgt von einem Vorzeichen und abschließend einer ganzen Zahl.

DEC(p,s), NUM(p,s), XDEC(p,s)

p = stellenanzahl; s = nachkommastellen

Der Daten-Wert kann ein Vorzeichen, einen Dezimalpunkt und Zahlen rechts davon haben.

DATE

Der Datumswert wird abdruckbar übergeben in der Form JJJJ-MM-TT (Jahr, Monat, Tag) .

TIME

Der Zeitwert wird abdruckbar übergeben in der Form HH:MM:SS (Stunden, Minuten, Sekunden).

TIME(3)

Der Zeitwert wird abdruckbar übergeben in der Form HH:MM:SS.FFF (Stunden, Minuten, Sekunden, Tausendstel-Sekunden).

TIMESTAMP(3)

Der Zeitwert wird abdruckbar übergeben in der Form JJJJ-MO-TT HH:MM:SS.FFF (Jahr, Monat, Tag, Stunden, Minuten, Sekunden, Tausendstel-Sekunden).

Reaktionen bei fehlerhaften USING-Parametern

Ist DRIVE/WINDOWS der Server und werden Datenwerte in ungültiger Form angeliefert oder sind sie z.B. zu groß für die Zielvariable, wird dies bereits beim Aufruf des DRIVE-Programms erkannt und der Übergabebereich mit der entsprechenden Fehlermeldung versorgt.

Reaktionen bei fehlerhaften RETURN-Parametern

Werden RETURN-Parameter fehlerhaft übergeben oder verletzen sie die Datentypbeschreibung im Zielprogramm (ist z.B. der übergebene Wert zu groß für den Zieldatentyp), gibt DRIVE/WINDOWS die Meldung CONVERSION_ERROR aus und ruft eine WHENEVER-Fehlerbehandlung auf, wenn sie vorhanden ist. Im Client wird ein DISTRIBUTION_STATE erzeugt mit DIS_SUB_CODE='D0001' und DIS_ERROR='CONVERSION_ERROR'.

12.1.8 Diagnosebereich

Bei fehlerhaftem Ansprung des Servers können im Diagnosebereich Diagnosemeldungen an den Client zurückgegeben werden. Bei Aufruf eines DRIVE-Servers aus einem C-Client ist dies immer dann der Fall, wenn im Header das Feld Aufruf-Code beim Rücksprung mit den Werten E oder P belegt ist. Jede Meldung muß in einer festen Länge von 79 Byte abgelegt und ggf. mit Leerzeichen aufgefüllt werden.

Kommuniziert DRIVE/WINDOWS mit einem COBOLProgramm, wird unmittelbar hinter der Diagnosemeldung ein Filler der Länge 1 Byte vorgesehen oder erwartet. Kommuniziert DRIVE/WINDOWS mit einem C-Programm, ist jede Diagnosemeldung mit einem Nullbyte abgeschlossen oder abzuschließen. Für jede Diagnosemeldung im Diagnosebereich müssen Sie also 80 Byte reservieren.

Die Länge des gesamten Diagnosebereichs wird durch die Differenz bestimmt zwischen der Gesamtlänge des Übergabebereichs (siehe Header-Feld 5) und der festen Headerlänge von 144 Byte. Die Anzahl der Diagnosemeldungen ergibt sich also aus der ganzzahligen Division der Länge des gesamten Diagnosebereichs durch die feste Länge der Diagnosemeldungen von 80 Byte. Zur Zeit werden von DRIVE/WINDOWS maximal 32 Diagnosemeldungen verarbeitet. Diese Anzahl kann geringer werden je nach der maximalen Größe des Übergabebereichs per Generierung.

12.2 Server-Seite

Ein DRIVE-Programm als Server bietet die gleiche Funktionalität an DRIVE-Variablen sowie temporären und permanenten SQL-Objekten wie ein lokal mit CALL gerufenes DRIVE-Programm. Dies gilt insbesondere für Gültigkeitsbereich und -dauer dieser Objekte und der globablen PARAMETER-Einstellungen, d.h. auch der mit PARAMETER DISTRIBUTION festgelegten Verteilungsinformationen (zu Gültigkeitsbereich und Lebensdauer von Definitionen siehe Seite 331).

Die DRIVE-Betriebsmittel gelten vorgangs-spezifisch. Ein erneuter Aufruf eines DRIVE-Programms kann also auf die zuvor permanent und global definierten und belegten Betriebsmittel wieder zugreifen, wenn der Server-Vorgang noch offen ist (siehe Kapitel „Transaktionskonzept“ auf Seite 261).

Ein adressierter Server beendet sich ordnungsgemäß mit

– COMMIT WORK WITH STOP oder STOP im letzten Server-DRIVE-Programm (d.h. oberste Server-Programmstufe)

– PEND FI im letzten C/COBOL-Teilprogrammlauf.

DRIVE-Server-Programme im SINIX und BS2000 können über die Client/ Server-Architektur der Verteilten UTM-Anwendungen (VTV) Auftraggeber einer mehrstufigen Auftraggeber-Auftragnehmer-Hierarchie sein. Es gelten die im Handbuch „DRIVE-Programmiersystem für SINIX/BS2000“ [12, 5] beschriebenen Regelungen. Unterschiede zum Auftragnehmer-Betrieb bei VTV ergeben sich in beliebiger Hierarchiestufe bei der Anweisung ROLLBACK WORK WITH RESET. Bei dieser Anweisung im Server bricht DRIVE/WINDOWS das Server-Programm mit Fehler ab, da die UPIC-UTM-Kopplung keinen Vorgangswiederanlauf unterstützt.

12.2.1 Wirkung von DRIVE-Anweisungen im Server

Anweisung END PROCEDURE

END PROCEDURE auf oberster Server-Programmstufe bewirkt , daß RETURN-Parameter an das rufende Programm im Client übergeben werden und ein Rücksprung zum Client erfolgt. Das Client-Programm wird hinter der CALL-Anweisung fortgesetzt.

Anweisung STOP

Da es keine Transaktionssynchronisation zwischen Client und Server in einer Verteilten Transaktion gibt, wird bei der Anweisung STOP der Server-Vorgang von UTM sofort beendet. Der Server gibt RETURN-Parameter an den Client zurük. Das Client-Programm wird hinter der CALL-Anweisung fortgesetzt.

Anweisung COMMIT WORK

Da es keine Transaktionssynchronisation zwischen Client und Server in einer Verteilten Transaktion gibt, werden mit der Anweisung COMMIT WORK die lokale Server-Transaktion und das Server-Programm sofort von UTM beendet. Der Server gibt RETURN-Parameter an den Client zurück. Wird COMMIT WORK im Server in einer tieferen lokalen Programmstufe erreicht, werden alle RETURN-Parameter der lokalen Server-Programmstufen bis zur obersten Programmstufe übertragen. Die Steuerung der Anwendung geht an den Client zurück, das Client-Programm wird hinter der CALL-Anweisung fortgesetzt. Der Server-Vorgang wird nicht wie bei Verteilten UTM-Anwendungen (VTV) gesperrt und kann sofort neue Aufträge vom Client verarbeiten.

Anweisung ROLLBACK WORK WITH RESET oder Meldung TA_CANCELLED

Die UPIC-UTM-Kopplung unterstützt keinen Vorgangswiederanlauf. Bei der Anweisung ROLLBACK WORK WITH RESET und der Meldung TA_CANCELLED des Datenbanksystems bricht DRIVE/WINDOWS daher das Server-Programm mit dem lokalen Datenbanksystem mit Fehler ab. Ist der Client ein DRIVE-Programm, gibt DRIVE/WINDOWS die Information TA_CANCELLED an den Client als &DML_STATE zurück (siehe Tabelle unten).

Fehlerhafter Programmabbruch

Ist der Client ein DRIVE-Programm, gibt DRIVE/WINDOWS die Inhalte der Systemvariablen &ERROR_STATE an den Client zurück:

Systemvariable	**Inhalt**
&ERROR	Literale wie bei WHENEVER-Anweisung 'OK', d.h. Fehler beim Datenbanksystem (siehe &DML_STATE) 'DRIVE ERROR' bei sonstigen Fehlern (z.B. Speicherengpaß)
&DML_STATE	Literale wie bei WHENEVER-Anweisung 'TA CANCELLED' 'SESSION CANCEL' 'OK' oder Blanks. d.h sonstiger Fehler (siehe &ERROR)
&SQL_CODE	Returncode des Datenbanksystems
&SQL_STATE	Returncode von SESAM V2

Systemvariable	Inhalt
&STATEMENT	Fehlerhafte Anweisung
&SUB_CODE	Zusätzlicher Fehlercode

Bei Programmfehler in einem DRIVE-Programm als Client wird ein DIS ERROR mit der Fehlerklasse D0001 erzeugt und die oben genannten Komponten der Systemvariablen ERROR_STATE in den Komponenten der Systemvariablen DISTRIBUTION_STATE des Client abgelegt.

Da es keine Transaktionssynchronisation zwischen Client und Server in einer Verteilten Transaktion gibt, wird bei fehlerhaftem Programmabbruch der Server-Vorgang sofort von UTM beendet.

12.2.2 DRIVE-Anweisungen, die im Server nicht erlaubt sind

In Server-Programmen, die mit Remote-CALL aufgerufen werden, sind folgende Anweisungen nicht erlaubt (siehe DRIVE-Lexikon [3]):

– FILL
– DISPLAY
– SEND MESSAGE
– COMMIT WORK WITH DISPLAY
– COMMIT WORK WITH SEND MESSAGE
– STOP WITH DISPLAY
– STOP WITH charausdruck
– CALL TAC (lokal)
– DO
– ROLLBACK WORK WITH RESET (siehe Seite 328).

12.2.3 Datenübergabe

Wie beim lokalen CALL-Aufruf eines DRIVE-Programms werden Parameter in der USING-Klausel der PROCEDURE-Anweisung definiert. Sind im DRIVE-Programm der obersten Programmstufe im Server (also im DRIVE-Programm, das vom Client direkt aufgerufen wurde) USING-Parameter mit RETURN definiert, übergibt DRIVE/WINDOWS die entspre-

chenden RETURN-Werte bei fehlerfreiem Rücksprung zum Client. RETURN-Werte werden bei der Anweisung END PROC im Server-Programm übergeben sowie bei COMMIT WORK, COMMIT WORK WITH STOP und STOP.

Bei der Anweisung COMMIT WORK auf einer unteren Programmstufe in Server-Umgebung (also einem DRIVE-Programm, das nicht direkt vom Client aufgerufen wurde) werden die RETURN-Parameterwerte aller Programmstufen zurückgegeben, die zwischen dem aktuellen und dem obersten Auftragnehmer-Programm im Server liegen, das direkt vom Client aufgerufen wurde. D.h., die RETURN-Parameter werden jeweils vom gerufenen zum rufenden DRIVE-Programm zurückgegeben.

Zwischen DRIVE-Programmen übernimmt das DRIVE-Run-Time-System (RTS) vollständig den Austausch von Parameterwerten, deren Ablage und Auswertung.

Verwenden Sie die REDEFINES-Klausel möglichst nicht, da z.B. das Redefinieren von INTERVAL-Feldern durch CHAR-Felder zu unterschiedlichen Datenergebnissen führt. Verwenden Sie Redefinitionen von unterschiedlichen Datentyp-Feldern möglichst nur in homogener Betriebssystemumgebung, da es bei heterogenen Systemen in der Regel zu „CONVERSION ERROR“ kommt.

Für den Nachrichten- und Datenaustausch **zwischen DRIVE/WINDOWS und einem anwendereigenen UTM-Teilprogramm in C/COBOL** wird ein Datenübergabebereich verwendet (siehe Abschnitt „Datenübergabe bei UTM-Teilprogrammen in C/COBOL im Server (für C/COBOL-Programmierer)“ auf Seite 316).

Im DRIVE-Programm der obersten Server-Programmstufe, also in dem DRIVE-Programm, das vom Client direkt aufgerufen wurde, können folgende USING-Parameter übergeben werden:

– einfache Variablen
– Vektoren
– Matrizen
– Strukturen

Diese Parameterübergabe ist unabhängig davon, ob der Client ein DRIVE-Programm oder eine C-UPIC-Anwendung ist.

12.2.4 Gültigkeit von Definitionen

Ein DRIVE-Programm als Server bietet die gleiche Funktionalität an DRIVE-Variablen sowie temporären und permanenten SQL-Objekten wie ein lokal mit CALL gerufenes DRIVE-Programm. Dies gilt insbesondere für Gültigkeitsbereich und -dauer dieser Objekte und der globablen PARAMETER-Einstellungen, d.h. auch der mit PARAMETER DISTRIBU-

TION festgelegten Verteilungsinformationen (zu Gültigkeitsbereich und Lebensdauer von Definitionen siehe Abschnitt „Auswirkungen der Transaktionssteuerung auf Definitionen“ auf Seite 281).

Die DRIVE-Betriebsmittel gelten nur vorgangs-spezifisch. Ein erneuter Aufruf eines DRIVE-Programms in derselben Server-Anwendung kann also nur dann auf die zuvor permanent und global definierten und belegten Betriebsmittel wieder zugreifen, wenn dieser Server-Vorgang nicht beendet wurde.

12.3 Allgemeine Programmierhinweise

Die folgenden Hinweise zum Programmieren von Client-Server-Anwendungen greifen einzelne Gesichtspunkte auf ohne Anspruch auf Vollständigkeit.

- Anwendung verteilen

  Beim Verteilen der Anwendungsprogramme, die eine reine Verarbeitung ausführen, sollten Sie überprüfen, ob die häufigere Kommunikationsbeziehung zum Programm des Präsentationsteils (Client) oder zum Programm mit Datenbankzugriff (Server) liegt und entsprechend verteilen.

- COMMIT WORK im Server

  Beim COMMIT WORK im Server beendet UTM die lokale Server-Transaktion und das Server-Programm. Es werden RETURN-Parameter an den Client übergeben, und die Steuerung geht an den Client zurück. Daraus folgt z.B.:

  – Das Präsentationsprogramm (Client) sollte **einen** Auftrag an den Server geben, der **ein** Ausgabeergebnis zurückliefert. D.h. evtl. Datenbankabfragen, die die Datenbankergebnisse auswertend steuern (z.B. Menge der Treffersätze ermitteln, um die Abfrage zu optimieren), erfolgen im Server-Programm. Dieses enthält keinen COMMIT WORK, da sonst ein Rücksprung zum Client erfolgt.

  – Das Server-Programm kann als Auftraggeber einer Auftraggeber-Auftragnehmer-Hierarchie in Verteilter Transaktionsverarbeitung (VTV) fungieren. Wird in einem Programm der VTV-Hierarchie im Server ein COMMIT WORK gesetzt, gibt es in der Verteilten Transaktion einen abschließenden COMMIT WORK im obersten Server-Programm und damit einen Rücksprung zum Client.

  – Bei Datenbankzugriff im Server mit variablem oder dynamischen Cursor dürfen DROP-Anweisungen nicht nach dem COMMIT WORK stehen, da sie nicht mehr durchlaufen werden. Sie müssen entweder vor dem COMMIT WORK stehen oder es darf keine geänderte Cursordeklaration notwendig sein.

- Werden in einem Dialogschritt mit dem Anwender mehrere Fenster ausgegeben, sollte pro Dialogschritt nur ein Netzzugriff erfolgen, um die Performance zu gewährleisten.
- Tritt ein „CONVERSION ERROR“ bei Parameterübergabe zwischen Client und Server auf, sollte dieser Fehler nicht nur im Server mit WHENEVER abgefragt werden, sondern auch im Client, um einen Programmabbruch im Client zu verhindern (siehe Seite 327).
- Bei DRIVE-Programmen als Client und Server garantiert DRIVE/WINDOWS die Konvertierung von Übergabeparametern zwischen ASCII und EBCDIC (siehe Seite 303). Dabei hinterlegt DRIVE/WINDOWS Metainformationen zu den Daten, die dadurch umfangreicher werden. Außerdem werden z.B. INTEGER-Felder durch Entpacken der Zahlen erheblich größer. Bei Strukturen als Übergabeparametern können Sie mit ungefähr 25 Byte zusätzlicher Information pro Komponente rechnen (zur Abschätzung des Speicherbedarfs für die reinen Datenwerte siehe C-Datentypen Seite 316).

  D.h., die Übergabemenge von Parametern in DRIVE-Variablen für die Netzübertragung muß u.U. reduziert werden, auch wenn eine zusätzliche Client-Server-Kommunikation nötig ist, bevor alle Daten bereitstehen.
- Beendet sich der Client, ohne daß zuvor im Server STOP erreicht wurde, beendet DRIVE/WINDOWS den Server und bricht das Client-Programm mit Fehlermeldung ab.

12.4 Generierung

Zur Generierung von verteilten Anwendungen siehe „DRIVE-Programmiersystem“ [1], Kapitel DRIVE/WINDOWS für UTM-Betrieb generieren.

12.5 Spezialinformationen zu C-UPIC-Programmen als Client

Sie können als Client auch eine C-UPIC-Anwendung einsetzen. In diesem Fall müssen Sie die UPIC-Aufrufe selbst programmieren und die Side-Information Datei `upicfile` einrichten (siehe Seite 303).

12.5.1 Ablauf der Kommunikation mit einem 3GL-Programm (für C-Programmierer)

Dieser Abschnitt beschreibt, in welchen Schritten ein UPIC-Programm abläuft. Bei einer DRIVE- zu DRIVE-Kommunikation werden diese Schritte intern von DRIVE/WINDOWS ausgeführt. Haben Sie als Client eine C-UPIC-Anwendung, müssen Sie die folgenden UPIC-Aufrufe selbst programmieren.

An- und Abmelden an der UPIC-Schnittstelle

Sie müssen die C-UPIC-Anwendung als Client intern an der UPIC-Schnittstelle anmelden mit dem UPIC-Aufruf Enable_UTM_UPIC. Aus Sicht des Clients macht sich die UPIC-Anwendung gegenüber der UTM-Anwendung im Server als User bekannt. Bei allen Server-UTM-Anwendungen, die von diesem Client adressiert werden, muß der Benutzername als PTERM generiert sein (siehe „DRIVE-Programmiersystem“ [1], Abschnitt „Verteilte Anwendungen generieren“). Damit wird auf der Server-UTM-Seite der Kommunikationspartner festgelegt.

Am Anwendungsende Ihres Clients melden Sie sich von der UPIC-Schnittstelle ab mit dem UPIC-Aufruf Disable_UTM_UPIC. Das Ende einer DRIVE-Client-Session wird erreicht durch die DRIVE-Anweisung STOP oder durch einen internen Fehler beim UPIC-Aufruf Initialize_Conversation (siehe auch WHENEVER 'DIS ERROR', Fehlerklasse D0002 Seite 312).

Logische Verbindung

Ein UPIC-Client kann zu einem Zeitpunkt nur eine einzige Server-Anwendung adressieren. UPIC vergibt für die logische Verbindung, die zum Server aufgebaut wird, eine eindeutige „Conversation_ID“. Mit ihr wird die Server-UTM-Anwendung adressiert, um einen UTM-Vorgang auszuführen. Diese „Conversation_ID“ und damit die logische Verbindung bleibt erhalten bis

– der Server-Vorgang sich beendet hat, und zwar fehlerfrei oder fehlerhaft (PEND FI oder ER/FR) oder

– die C-UPIC-Anwendung als Client die logische Verbindung wegen Fehler vorzeitig abbauen muß (siehe UPIC-Aufruf Deallocate).

Um die logische Verbindung (Conversation) zum Server und den Server-Vorgang zu starten, müssen die folgenden UPIC-Aufrufe ausgeführt werden:

– Initialize_Conversation

  Initialisiert die Charakteristika der Conversation, indem die Werte aus der `upicfile` ausgewertet werden. UPIC legt die Conversation_ID fest.

– SET_TP_NAME

  Gibt den Transaktionscode für das Server-Teilprogramm an. Alternativ können Sie den Transaktionscode auch in der `upicfile` angeben.

  Bei Aufruf eines DRIVE-Programms als Server wird der fest vorgegebene Transaktionscode für Interpreter- oder Compiler-Betrieb im Server eingesetzt (SQLRMT oder DRT#SC##).

– Allocate

  Richtet die Conversation ein.

– Send_Data und Receive

  Rufen den Server auf mit Datenübergabe und empfangen dessen Antwort.

  Im Client wird der Programmlauf an dieser Stelle unterbrochen, bis die Antwort des Servers eingetroffen ist. Am Returncode des UPIC-Aufrufs Receive können Sie unterscheiden, ob der Server noch offen ist und weitere Aufträge empfangen kann oder ob er sich ordnungsgemäß oder fehlerhaft beendet hat und keine Aufträge empfangen kann. Hat sich der Server-Vorgang fehlerhaft beendet (z.B. PEND FR), kann es sein, daß UPIC das Ergebnis DEALLOCATED_ABEND erst beim nächsten Aufruf liefert. In diesem Fall gibt UPIC bei Receive im Client nicht das Senderecht (Status NO_STATUS_RECEIVED). DRIVE/WINDOWS führt dann implizit einen weiteren Receive-Aufruf an UPIC aus.

Welche PEND-Aufrufe durch DRIVE-Anweisungen ausgelöst werden, finden Sie als Spezialinformation gekennzeichnet in Abschnitt „DRIVE-Anweisungen und Server-Zustände“ auf Seite 304.

12.5.2 Datenübergabe

Zum Nachrichtenaustausch zwischen einer C-UPIC-Anwendung und einem DRIVE-Programm wird ein Übergabebereich verwendet (siehe Abschnitt „Datenübergabe bei UTM-Teilprogrammen in C/COBOL im Server (für C/COBOL-Programmierer)“ auf Seite 316.

Aufbau der Übergabeparameter

Ist der Client eine anwendereigene C-UPIC-Anwendung und der Server ein DRIVE-Programm, müssen Sie zu **allen** USING-Parameter Indikatorwerte übergeben. Beim Rücksprung zum Client übergibt DRIVE/WINDOWS für alle RETURN-Parameter Indikatorwerte.

12.5.2.1 Datenkonvertierung bei C-Programmen als Client und DRIVE als Server

```
PROC USING &r1 DEC(5,2)
          RETURN &r2 INTEGER
```

Das UTM-Teilprogramm in C/COBOL als Client muß numerische Daten aus evtl. internen Rechenformaten in abdruckbare Form konvertieren. DRIVE/WINDOWS erwartet numerische Daten in abdruckbarer Form und konvertiert automatisch in die interne DRIVE-Form. Vor **jedem** Datenwert (bei Vektoren, Matrizen und Strukturen vor jedem Komponentenwert) erwartet DRIVE/WINDOWS einen Indikatorwert. Im DRIVE-Programm werden die Daten in der Form deklariert, in der sie verarbeitet werden sollen. Auf dem Rückweg konvertiert DRIVE/WINDOWS numerische RETURN-Werte aus dem internen DRIVE-Format in abdruckbare Form. Vor j**edem** Datenwert (bei Vektoren, Matrizen und Strukturen vor jedem Komponentenwert) legt DRIVE/WINDOWS einen Indikatorwert ab. Zur Weiterverarbeitung numerischer Daten durch ein UTM-Teilprogramm konvertiert dieses ggf. die abdruckbare Form in die interne Rechenform der entsprechenden Programmiersprache C oder COBOL.

Ablageform der einzelnen Datentypen

Das DRIVE-Programm erwartet die Datenwerte immer in fester Länge (abhängig vom Datentyp) in abdruckbarer Form.

CHAR(n)

DRIVE/WINDOWS übernimmt Zeichen in Länge n.

Insbesondere bei der Verbindung von UTM-Teilprogrammen in C mit DRIVE/WINDOWS und beim Übergeben von Strings vom Datentyp CHAR müssen Sie darauf achten, Literale ggf. mit Leerzeichen aufzufüllen, damit nicht eventuell vorhandene Endekennzeichen (X'00') in die DRIVE-Variable übertragen werden.

VARCHAR(n)

DRIVE/WINDOWS übernimmt Daten in signifikanter Länge. Die Ablageform ist abhängig von der Programmiersprache:

C: String in signifikanter Länge, mit Nullbyte abgeschlossen

COBOL: Längenfeld (abdruckbar, Format wie bei SMALLINT), gefolgt von Wert. Sie können also Datenmengen in variabler Länge übertragen.

Verwenden Sie für COBOL-Anwendungen, die USING-Daten mit fester Länge übergeben, CHAR-Felder statt VARCHAR-Feldern.

INTEGER, INTERVAL

DRIVE/WINDOWS übernimmt folgende Zeichen: eine Folge von Leerzeichen, Vorzeichen und eine ganzzahlige Ziffernfolge.

SMALLINT

DRIVE/WINDOWS übernimmt folgende Zeichen: eine Folge von Leerzeichen, Vorzeichen und eine ganzzahlige Ziffernfolge.

REAL, FLOAT

DRIVE/WINDOWS übernimmt folgende Zeichen: eine Folge von Leerzeichen, Vorzeichen, eine Ziffernfolge mit oder ohne Dezimalpunkt, gefolgt von e oder E, gefolgt von einem Vorzeichen und abschließend einer ganzen Zahl. Die Mantisse ist 7-stellig, der Exponent 2- oder 3-stellig.

DEC(p,s), NUM(p,s), XDEC(p,s)

p = stellenanzahl; s = nachkommastellen

Führende Leerzeichen werden überlesen. Der Daten-Wert kann ein Vorzeichen, einen Dezimalpunkt und Zahlen rechts davon haben.

DATE

Der Datenwert wird abdruckbar erwartet in der Form JJJJ-MM-TT (Jahr, Monat, Tag) . .

TIME

Der Zeitwert wird abdruckbar erwartet in der Form HH:MM:SS (Stunden, Minuten, Sekunden).

TIME(3)

Der Zeitwert wird abdruckbar erwartet in der Form HH:MM:SS.FFF (Stunden, Minuten, Sekunden, Tausendstel-Sekunden).

TIMESTAMP(3)

Der Zeitwert wird abdruckbar erwartet in der Form JJJJ-MO-TT HH:MM:SS.FFF (Jahr, Monat, Tag, Stunden, Minuten, Sekunden, Tausendstel-Sekunden).

Reaktionen bei fehlerhaften USING-Parametern

Wenn Datenwerte in ungültiger Form angeliefert werden oder z.B. zu groß sind für die Zielvariable, wird dies bereits beim Aufruf des DRIVE-Programms erkannt und der Übergabebereich mit der entsprechenden Fehlermeldung für den Rücksprung zum Client versorgt.

13 Verteilte Transaktionsverarbeitung (VTV)

Dieses Kapitel beschreibt:

- die unterschiedlichen Anwendungen, die DRIVE/WINDOWS unterstützt (ab Seite 340)
- die DRIVE-Anweisungen, die VTV im DRIVE-Programm als Auftraggeber ermöglichen (ab Seite 341)
- Auswirkungen von DRIVE-Anweisungen im Auftraggeber und Auftragnehmer auf die VTV (ab Seite 350 und ab Seite 356)
- die Datenübergabe zwischen DRIVE/WINDOWS und anwendereigenen UTM-Teilprogrammen in C/COBOL (ab Seite 316)

Werden DRIVE-Anwendungen, die im UTM-Betrieb laufen, verteilt, spricht man auch von Verteilter Transaktionsverarbeitung (VTV). Lokale Anwendungen mit Alpha-Bedienoberfläche geben als Auftraggeber Verarbeitungsaufträge an entfernte Anwendungen als Auftragnehmer. Diese greifen auf lokale Daten zu, um ihren Verarbeitungsauftrag zu erledigen. Es können unterschiedliche Anwendungen miteinander verbunden werden: DRIVE-Anwendungen und Anwendungen in C bzw. COBOL. Die Anwendungen können lokal in einem Rechner ablaufen oder verteilt in verschiedenen Rechnern. Dabei kann auf den Rechnern BS2000 oder SINIX als Betriebssystem installiert sein.

In einer BS2000-Anwendung, die aus mindestens zwei Programmen besteht, sind Datenbankzugriffe auf SESAM/SQL und UDS/SQL möglich. In einer SINIX-Anwendung, die aus mindestens zwei Programmen besteht, sind Datenbankzugriffe auf zwei verschiedene INFORMIX-Datenbanken möglich. Die Datenbankzugriffe erfolgen jeweils über das lokale Datenbanksystem.

DRIVE/WINDOWS garantiert volle Transaktionssicherung über alle beteiligten Anwendungen hinweg. Das Ende aller Transaktionen in lokalen und entfernten Anwendungen wird von UTM-D synchronisiert, d.h. beide Anwendungen setzen gleichzeitig Sicherungspunkte. Wenn Fehler auftreten, werden über UTM-D alle beteiligten Transaktionen zurückgesetzt. Die bei VTV synchronisierten Transaktionen bilden eine Einheit, die Verteilte Transaktion.

Aus der Sicht des Anwenders läuft eine Verteilte Transaktion genauso ab wie eine lokale. Eine Eingabe wird von UTM an die lokale Anwendung geleitet. Braucht diese zur Bearbeitung Leistungen einer entfernten Anwendung, schickt sie über UTM-D eine entsprechende Nachricht dorthin und startet damit einen neuen Vorgang: Der Auftraggeber-Vorgang beauftragt einen entfernten Auftragnehmer-Vorgang, eine Teilaufgabe durchzuführen. D.h.,

die Programme, die als UTM-Teilprogramme in einer Anwendung zusammenarbeiten, kennen die logische Verteilung und steuern sie. Von DRIVE/WINDOWS unterstützte Auftraggeber-/-nehmer-Verbindungen

Bei verteilten DRIVE-Anwendungen unter UTM-D kann der Auftraggeber sowohl ein DRIVE-Programm als auch ein anwendereigenes UTM-Teilprogramm in C/COBOL sein. Auftragnehmer können ebenfalls DRIVE-Programme oder anwendereigene UTM-Teilprogramme in C/COBOL sein. Daraus ergeben sich folgende Verbindungsmöglichkeiten.

Homogene Verbindungen

Auftraggeber	**Auftragnehmer**
DRIVE/WINDOWS (SINIX)	DRIVE/WINDOWS (SINIX)
DRIVE/WINDOWS (SINIX)	C/COBOL-Teilprogramm (SINIX)
C/COBOL-Teilprogramm (SINIX)	DRIVE/WINDOWS (SINIX)
DRIVE/WINDOWS (BS2000)	DRIVE/WINDOWS (BS2000)
DRIVE/WINDOWS (BS2000)	C/COBOL-Teilprogramm (BS2000)
C/COBOL-Teilprogramm (BS2000)	DRIVE/WINDOWS (BS2000)

Heterogene Verbindungen

Auftraggeber	**Auftragnehmer**
DRIVE/WINDOWS (SINIX)	DRIVE/WINDOWS(BS2000)
DRIVE/WINDOWS (SINIX)	C/COBOL-Teilprogramm (BS2000)
C/COBOL-Teilprogramm (SINIX)	DRIVE/WINDOWS (BS2000)
DRIVE/WINDOWS (BS2000)	DRIVE/WINDOWS (SINIX)
DRIVE/WINDOWS (BS2000)	C/COBOL-Teilprogramm (SINIX)
C/COBOL-Teilprogramm (BS2000)	DRIVE/WINDOWS (SINIX)

13.1 DRIVE-Programm als Auftraggeber

Die folgenden Abschnitte gelten für DRIVE-Programme, die in Auftraggeber-Umgebung ablaufen. Das sind prinzipiell auch solche Auftraggeber-Umgebungen, die selbst Auftragnehmer in einer mehrstufigen Auftraggeber-Auftragnehmer-Hierarchie sind. Denn ein Auftraggeber-Vorgang kann mehrere Auftragnehmer-Vorgänge starten. Diese können weitere Auftragnehmer-Vorgänge starten, also selbst wieder Auftraggeber werden, so daß eine Auftraggeber-Auftragnehmer-Hierarchie ensteht.
Regeln, die nur für DRIVE-Programme in der obersten Auftraggeber-Umgebung einer mehrstufigen Hierarchie gelten, werden gesondert aufgeführt.
Regeln, die aus der Auftragnehmer-Sicht von DRIVE-Programmen folgen, sind im Abschnitt Abschnitt „DRIVE-Programm als Auftragnehmer“ auf Seite 356 beschrieben.

Im DRIVE-Programm als Auftraggeber wird die Verteilte Transaktionsverarbeitung mit der Anweisung OPTION DISTRIBUTION eingestellt. Die Verteilungsinformationen werden mit der Anweisung PARAMETER DISTRIBUTION angegeben und parallele Aufträge mit der Anweisung DISPATCH gebündelt.

Ein DRIVE-Programm als Auftraggeber adressiert Auftragnehmer-Vorgänge automatisch nach der BOTTOM-UP-Strategie von UTM. Außerdem hält sich DRIVE/WINDOWS als Auftraggeber an die Programmierregeln von UTM für VTV, die sogenannten Vorgangs- und Transaktionsregeln. D.h.:

– DRIVE/WINDOWS überprüft im Auftraggeber-Vorgang, daß dieser nicht beendet wird, solange noch offene Auftragnehmer-Vorgänge vorhanden sind.

– DRIVE/WINDOWS überprüft im Auftraggeber-Vorgang, daß eine Transaktion erst beendet wird, wenn alle Auftragnehmer-Vorgänge Transaktionsende erreicht haben.

Näheres zur BOTTOM-UP-Strategie und zu den Programmierregeln siehe Handbuch UTM, UTM-D Programmieren [29].

13.1.1 Verteilte Verarbeitung einstellen mit OPTION DISTRIBUTION

Mit der Übersetzungsoption OPTION DISTRIBUTION wählen Sie in einem DRIVE-Programm zwischen lokaler und verteilter Verarbeitung.

Der Operand OFF bewirkt, daß jede CALL- bzw. ENTER-Anweisung lokal verarbeitet wird. Der Operand ON bewirkt, daß DRIVE/WINDOWS bei Programmaufrufen mit CALL und ENTER prüft, ob Verteilungsinformationen mit der Anweisung PARAMETER DISTRIBUTION gesetzt worden sind. Sind keine Verteilungsinformationen gesetzt, wird jede CALL- und ENTER-Anweisung lokal verarbeitet.

Sind Verteilungsinformationen gesetzt, versucht DRIVE/WINDOWS zunächst, eine Verteilungsinformation mit einem passenden Eintrag zu finden und das gerufene Programm

remote zu verarbeiten. Ist kein passender Eintrag vorhanden, wird das Programm lokal verarbeitet.
Im Dialog-Modus wird jede ENTER-Anweisung so verarbeitet, als ob OPTION DISTRIBUTION auf ON gesetzt wäre.

Ein Eintrag ist passend, wenn die Operandenwerte für Bibliothek und Element bei der Anweisung CALL oder ENTER mit den Operandenwerten übereinstimmen, die bei der Anweisung PARAMETER DISTRIBUTION für das DRIVE-Interpreter- oder Compiler-Programm gemacht wurden (siehe „DRIVE-Lexikon“ [3]).

Ist als Auftragnehmer ein DRIVE-Interpreter-Programm angegeben und nur der Elementname ohne Bibliothek, so wird dem Auftragnehmer-Vorgang nicht die im Auftraggeber-Vorgang mit PARAMETER DYNAMIC LIBRARY eingestellte Bibliothek mitgeteilt. Der Auftragnehmer-Vorgang löst die Angabe vielmehr mit seiner eigenen PARAMETER-Voreinstellung auf (ggf. als DRIVE-Startparameter in der entfernten UTM-Anwendung siehe Abschnitt „Verteilung vorbereiten“ auf Seite 302 und Abschnitt „Gültigkeit von Definitionen“ auf Seite 331).

Beim Aufruf eines anwendereigenen UTM-Teilprogramms mit CALL/ENTER TAC wird der Operandenwert für den Transaktionscode mit dem in der Anweisung PARAMETER DISTRIBUTION verglichen. Der Operandenwert darf kein DRIVE-Interpreter-TAC oder DRIVE-Compiler-TAC sein. Beim Aufruf eines anwendereigenen UTM-Teilprogramms mit CALL/ENTER muß die Programmiersprache, in der es geschrieben ist, angegeben werden.

Der Name der Auftragnehmer-Anwendung kann entweder mit der Anweisung PARAMETER DISTRIBUTION angegeben werden oder bei der Generierung der UTM-Anwendung (siehe DRIVE-Programmiersprache, Kapitel DRIVE/WINDOWS für den UTM-Betrieb generieren, Abschnitt „Auftragnehmer adressieren“ und „DRIVE-Lexikon“ [3])

13.1.2 Verteilungsinformationen angeben mit PARAMETER DISTRIBUTION

Mit der Anweisung PARAMETER DISTRIBUTION können Sie Verteilungsinformationen setzen. Pro Anweisung werden die Verteilungsinformationen zu genau einem Auftragnehmer-Programm gesetzt, einem DRIVE-Programm oder einem anwendereigenen UTM-Teilprogramm in C/COBOL (siehe Abschnitt „Gültigkeit von Definitionen“ auf Seite 331).

Die Anweisung PARAMETER DISTRIBUTION ist nur erlaubt, wenn kein Auftragnehmer-Vorgang offen ist bzw. wenn sich alle zuvor adressierten Auftragnehmer-Vorgänge wieder beendet haben. Der Auftraggeber weist nämlich dem Auftragnehmer beim Adressieren auch eine Vorgangsidentifikation (Vorgangs-ID) zu, bevor er einen Auftrag erteilt. Die logische Verbindung zwischen Auftraggeber und offenem Auftragnehmer-Vorgang über die Vorgangs-ID bleibt erhalten, solange der Vorgang

nicht beendet ist. Es empfiehlt sich daher, die PARAMETER DISTRIBUTION-Anweisungen als Startparameter beim Hochfahren der UTM-Anwendung anzugeben.

Mit der Anweisung PARAMETER DISTRIBUTION können folgende Angaben gemacht werden, die dann beim Aufrufen eines Programms mit CALL/ENTER und eingestellter Verteilter Verarbeitung (OPTION DISTRIBUTION) mit den Angaben bei CALL/ENTER verglichen werden (s.o.):

– Bibliothek und Elementname.

  CODE/OBJECT für ein DRIVE-Interpreter- bzw. DRIVE-Compiler-Programm.

  Bei OBJECT wird die Angabe einer Bibliothek ignoriert.

– Transaktionscode für den Auftragnehmer-Auftrag, der ein anwendereigenes UTM-Teilprogramm ist

– Name der Auftragnehmer-Anwendung (siehe „DRIVE-Programmiersprache“ [1], Kapitel DRIVE/WINDOWS für den UTM-Betrieb generieren, Abschnitt „Auftragnehmer adressieren“ und „DRIVE-Lexikon“ [3).

Die Angaben für Bibliothek, Elementname und Transaktionscode sind immer eindeutig, da eine Verteilungsinformation mit denselben Operandenwerten die alten Werte überschreibt. Diese Werte müssen also über alle UTM-Anwendungen, die an der VTV beteiligt sind, eindeutig sein.

13.1.3 Parallele Verteilte Verarbeitung anstoßen mit DISPATCH

Verteilte Transaktionsverarbeitung ermöglicht es auch, mehrere Verbindungen zwischen zwei UTM-Anwendungen aufzubauen und mehrere Dialogaufträge parallel abzuwickeln (siehe „DRIVE-Programmiersprache“ [1], Kapitel DRIVE/WINDOWS für den UTM-Betrieb generieren,Abschnitt „KDCDEF-Steueranweisungen“).

Mit den Anweisungen DISPATCH und END DISPATCH können Sie alle CALL-Anweisungen zur mehrstufigen Verteilten Transaktionsverarbeitung bündeln. D.h., alle zwischen-DISPATCH und END DISPATCH stehenden Remote-CALL-Anweisungen werden erst zum Zeitpunkt der END DISPATCH-Anweisung parallel ausgeführt. Die Verarabeitung im DRIVE-Programm nach END DISPATCH wird fortgesetzt, wenn alle remote aufgerufenen Programme beendet wurden (siehe „DRIVE-Lexikon“ [3]).

Lokale CALL-Anweisungen werden sofort ausgeführt. Für CALL-Anweisungen, die in einem DISPATCH-Block remote verarbeitet werden, baut DRIVE/WINDOWS zum Zeitpunkt der CALL-Anweisung einen Datenübergabebereich für die USING-Parameterwerte auf. Alle Remote- CALL-Anweisungen werden aber erst zum Zeitpunkt der END DISPATCH-Anweisung ausgeführt. D.h., erst zu diesem Zeitpunkt werden die Parameterwerte aus dem

Übergabebereich an das CALL-Programm übergeben. Anschließend werden die Rückgabewerte des gerufenen Programms in die Parameter übernommen, die bei der CALL-Anweisung mit RETURN gekennzeichnet wurden.

Wird also zwischen einer Remote-CALL-Anweisung und END DISPATCH der Wert eines RETURN-Parameters gelesen, so ist dieser noch unverändert.

Wird der Wert eines RETURN-Parameters zwischen der Remote-CALL-Anweisung und END DISPATCH verändert (z.B. durch SET), gilt der neue Datenwert nur bis END _DISPATCH. Dann wird er vom Rückgabewert des mit CALL gerufenen Programms überschrieben.

Werten Sie bei DISPATCH-Blöcken die RETURN-Parameterwerte aus Remote-CALL-Aufrufen erst nach END DISPATCH aus. Fragen Sie Datenwerte von DRIVE-Variablen, die als RETURN-Parameter verwendet und zwischen einer Remote-CALL-Anweisung und END DISPATCH z.B. durch SET geändert werden, nicht mehr nach END DISPATCH ab.

Beispiel

```
DISPATCH;
   CALL local_proc1      USING RETURN &a; ------------------------------------  (1)
   CALL remote_proc2     USING RETURN &a; ------------------------------------  (2)
   SET &b = &a;-----------------------------------------------------------------  (3)
   CALL TAC remote_tac   USING RETURN &b; ------------------------------------  (4)
END DISPATCH; ---------------------------------------------------------------  (5)
```

(1) Die lokale CALL-Anweisung wird sofort ausgeführt. Dadurch hat der RETURN-Parameter &a unmittelbar nach dem CALL-Aufruf den von local_proc1 zurrückgegebenen Datenwert. (Im DISPATCH-Block müssen nur die Namen der RETURN-Parameter von remote CALL-Aufrufen eindeutig sein.)

(2) Der von local_proc1 an den RETURN-Parameter &a zurückgegebene Datenwert wird in einem Übergabebereich für den verzögerten Aufruf von remote_proc2 gesichert.

(3) Der Variablen &b wird der Wert der Variablen &a zugewiesen. Dies ist weiterhin der vom Programm local_proc1 in (1) zurückgegebene Datenwert.

(4) Der in (3) an &b zugewiesene Datenwert wird in einem Übergabebereich für den verzögerten Aufruf des anwendereigenen UTM-Teilprogramms remote_tac gesichert.

(5) Die CALL-Anweisungen (2) und (4) werden angestoßen. Danach werden die jeweiligen RETURN-Paramterwerte an die Variablen &a und &b übertragen.

13.1.4 Transaktions- und Vorgangsende bei VTV

Transaktions- und Vorgangsende werden in DRIVE/WINDOWS bei VTV wie im lokalen Betrieb durch die Anweisungen COMMIT WORK, COMMIT WORK WITH STOP und STOP gesteuert. Die Aussagen zu COMMIT WORK umfassen im Abschnitt „Zulässigkeit von DRIVE-Anweisungen abhängig vom Auftragnehmer“ auch COMMIT WORK WITH display/send message (zu Unterschieden in den intern ausgelösten PEND-Varianten siehe Abschnitt „Interne PEND-Varianten von DRIVE-Anweisungen“ auf Seite 352).

- Ist eine lokale Datenbank-Transaktion offen, beenden die Anweisungen COMMIT WORK und COMMIT WORK WITH STOP wie im lokalen Betrieb die Transaktion. COMMIT WORK WITH STOP beendet außerdem den Vorgang. Die Anweisung STOP ist wie im lokalen Betrieb auch bei VTV nicht erlaubt, wenn eine lokale Datenbank-Transaktion offen ist.
- Ist keine lokale Datenbank-Transaktion offen, beenden die Anweisungen COMMIT WORK und COMMIT WORK WITH STOP wie im lokalen Betrieb die UTM-Transaktion. STOP und COMMIT WORK WITH STOP beenden den Vorgang.

13.1.5 Logische Verbindung zwischen Auftraggeber und Auftragnehmer

Bei Remote-Aufträgen mit CALL und ENTER werden intern logische Verbindungen zwischen dem aufrufenden Auftraggeber-Vorgang und dem aufgerufenen Auftragnehmer-Vorgang über UTM aufgebaut. Jede dieser Verbindungen ist eindeutig und wird von DRIVE/WINDOWS intern durch eine eindeutige Vorgangsidentifikation (Vorgangs-ID) verwaltet. Die möglichen Verbindungen, die ein DRIVE-Auftraggeber zu Auftragnehmer-Partnern haben kann, werden in den KDCDEF-Steueranweisungen der UTM-Generierung festgelegt (siehe „DRIVE-Programmiersprache“ [1], Kapitel DRIVE/WINDOWS für den UTM-Betrieb generieren, Abschnitt „KDCDEF-Steueranweisungen“).

Wichtig ist, daß eine logische Verbindung, die ein DRIVE-Auftraggeber über UTM zu seinem Auftragnehmer-Partner hat, DRIVE-Programm-übergreifend, nämlich vorgangsspezifisch gültig ist. Die Verbindung besteht also zwischen dem DRIVE-Auftraggeber-Vorgang und einem entfernten Auftragnehmer-Vorgang. So können zwei verschiedene DRIVE-Programme, die im Programm-Modus aufgerufen werden und im Auftraggeber-Vorgang nacheinander ablaufen, über die gleiche logische Verbindung mit dem gleichen Auftragnehmer-Vorgang kommunizieren, wenn die Verbindung über einen Auftrag hinaus offen bleibt (siehe „DRIVE-Programmiersprache“ [1], Kapitel DRIVE/WINDOWS für den UTM-Betrieb generieren, Abschnitt „Logische Verbindung zwischen Auftraggeber und Auftragnehmer“).

Beispiel:

Ruft das DRIVE-Programm A zwei lokale DRIVE-Programme B und C mit CALL auf, können in den Programmen B und C Remote-CALL-Aufrufe der gleichen Auftragnehmer-Anwendung über die gleiche logische Verbindung gehen, wenn diese Verbindung beim ersten Aufruf offen bleibt. Dies gilt unabhängig von der Aufrufhierarchie z.B. auch, wenn das Programm C aus B aufgerufen wird.

In der obersten DRIVE-Auftraggeber-Umgebung einer mehrstufigen Auftraggeber-Auftragnehmer-Hierarchie müssen vor dem Übergang vom Programm- in den Dialog-Modus sowie vor dem Ausführen einer DO-Anweisung im Programm-Modus alle logischen Verbindungen zu Auftragnehmer-Vorgängen abgebaut sein. Andernfalls bricht DRIVE/WINDOWS den Programm-Modus mit Fehler ab.

Logische Verbindung bei CALL-Aufrufen

Wird im DRIVE/WINDOWS als Auftraggeber zum erstenmal ein Auftragnehmer-Programm in einer entfernten UTM-Anwendung mit CALL aufgerufen, startet UTM einen neuen Auftragnehmer-Vorgang und baut damit eine logische Verbindung auf. Das Auftragnehmer-Programm legt mit der DRIVE-Anweisung (im DRIVE-Programm) oder der PEND-Variante (im UTM-Teilprogramm in C/COBOL), mit der es beendet wird, die weitere Verwendbarkeit der aufgebauten logischen Verbindung fest:

- Die Verbindung bleibt offen:
  Eine folgende Remote-CALL-Anweisung auf ein Programm derselben Auftragnehmer-Anwendung wird über die bestehende logische Verbindung in dem weiterhin offenen Auftragnehmer-Vorgang ausgeführt.
  Ist der Auftragnehmer ein DRIVE-Programm, können die zuvor definierten Betriebsmittel (globale PARAMETER-Einstellungen, permanente DRIVE-Programm-Objekte) weiterverwendet werden.

- Die Verbindung wird gesperrt bis zum Beenden der Verteilten Transaktion:
  Eine folgende Remote-CALL-Anweisung auf ein Programm derselben Auftragnehmer-Anwendung wird über eine neue logische Verbindung ausgeführt.
  Ist der Auftragnehmer ein DRIVE-Vorgang, werden in Auftragnehmer-Umgebung die globalen und permanten Betriebsmittel initialisiert (siehe auch Abschnitt „Verteilung vorbereiten“ auf Seite 302).
  Durch das Sperren von logischen Verbindungen können außerhalb von DISPATCH-Blöcken mehrere logische Verbindungen zu derselben Auftragnehmer-Anwendung aufgebaut werden und nach dem Beenden der Verteilten Transaktion offen sein. Bei weiteren Remote-CALL-Aufrufen werden diese offenen Verbindungen von DRIVE/WINDOWS in der Reihenfolge wiederverwendet, in der sie aufgebaut wurden.

- Die Verbindung wird beim Beenden der Verteilten Transaktion abgebaut:
  Eine folgende Remote-CALL-Anweisung auf ein Programm derselben Auftragnehmer-Anwendung wird über eine neue logische Verbindung ausgeführt.
  Ist der Auftragnehmer ein DRIVE-Vorgang, werden in Auftragnehmer-Umgebung die globalen und permanten Betriebsmittel initialisiert (siehe auch Abschnitt „Verteilung vorbereiten“ auf Seite 302).

Die DRIVE-Anweisungen, die das Auftragnehmer-Programm beenden, und ihre Auswirkungen auf die logische Verbindung und die VTV insgesamt sind im Abschnitt „Auswirkungen von DRIVE-Anweisungen auf VTV“ auf Seite 356 beschrieben. Die PEND-Varianten der UTM-Teilprogramme in C/COBOL und ihre Auswirkungen auf die VTV finden Sie im Abschnitt „Programmierregeln“ auf Seite 358.

Logische Verbindung bei CALL-Aufrufen im DISPATCH-Block

Für jeden Remote-Auftrag mit CALL in einem DISPATCH-Block wird eine eigene logische Verbindung mit einer eindeutigen internen Vorgangsidentifikation (Vorgangs-ID) verwendet. Bereits offene Verbindungen werden dabei in der Reihenfolge, in der sie zuvor aufgebaut wurden, wieder verwendet (siehe auch Abschnitt „Beispiel für Ablaufhierarchie von Transaktion, Vorgang und logischer Verbindung“ auf Seite 348).

Logische Verbindung bei asynchronen Aufrufen mit ENTER

Wird aus einem Auftraggeber-DRIVE-Programm ein asynchroner Remote-Auftrag mit ENTER gestartet, baut DRIVE/WINDOWS über UTM immer eine neue logische Verbindung auf und startet in der Auftragnehmer-Anwendung einen neuen Auftragnehmer-Vorgang. Unmittelbar nach dem ENTER-Aufruf gibt UTM die logische Verbindung wieder frei. Der asynchrone Remote-Vorgang kann also über diese logische Verbindung nicht noch einmal erreicht werden und beendet sich wie im lokalen Betrieb.

Beispiel für Ablaufhierarchie von Transaktion, Vorgang und logischer Verbindung

Das folgende Beispiel zeigt ein DRIVE-Programm als Auftraggeber in einer Auftraggeber-Auftragnehmer-Beziehung.

```
PAR DISTRIBUTION ELEMENT="abfrage" APPLICATION="appl" STATUS=ADD;   ┐
                                                                    ├──────────   (1)
PAR DISTRIBUTION ELEMENT="buchung" APPLICATION="appl" STATUS=ADD;   ┘
...
CALL "buchung" USING &kunden_satz; ──────────────────────────────────────────   (2)
...
CALL "abfrage" USING 'open', RETURN &kunden_satz; ───────────────────────────   (3)
CALL "buchung" USING &kunden_satz; ──────────────────────────────────────────   (4)
...
COMMIT WORK WITH DISPLAY FORM &kunden_satz; ─────────────────────────────────   (5)
...

DISPATCH; ───────────────────────────────────────────────────────────────────   (6)
CALL "abfrage" USING 'stop', RETURN &kunden_satz_1; ─────────────────────────   (7)
CALL "abfrage" USING 'stop', RETURN &kunden_satz_2; ─────────────────────────   (8)
END DISPATCH; ───────────────────────────────────────────────────────────────   (9)
...
COMMIT WORK WITH DISPLAY FORM &ende_meldung; ────────────────────────────────  (10)
```

Das Beispiel geht von folgenden Annahmen aus:

- das Auftragnehmer-DRIVE-Programm "abfrage" wird folgendermaßen beendet:
  - mit END PROC, also ohne Anfordern von Transaktions- und Vorgangsende, wenn der Parameterwert 'open' übergeben wird
  - mit STOP, also mit Anfordern von Transaktions- und Vorgangsende, wenn der Parameterwert 'stop' übergeben wird
- das Programm "buchung" beendet die Verarbeitung mit COMMIT WORK, also mit Anfordern von Transaktionsende ohne Vorgangsende.

Das Beispiel läuft folgendermaßen ab:

(1) Im Auftraggeber werden die Verteilungsinformationen festgelegt.

(2) Mit dem CALL- Aufruf wird eine erste logische Verbindung id1 aufgebaut und ein neuer Vorgang v1 in der Auftragnehmer-Anwendung "appl" gestartet. Die Betriebsmittel (globale PARAMETER-Einstellungen, permanente DRIVE-Programm-Objekte) werden initialisiert. v1 führt das DRIVE-Programm "buchung" aus.
Da "buchung" Transaktionsende anfordert, wird die Verbindung id1 von UTM bis

zum Beenden der Verteilten Transaktion gesperrt, d.h. bis das Auftraggeber-DRIVE-Programm COMMIT WORK durchläuft. Der Auftragnehmer-Vorgang v1 bleibt weiterhin offen.

(3) Mit dem zweiten CALL-Aufruf eines DRIVE-Programms in der Auftragnehmer-Anwendung "appl" wird eine freie logische Verbindung zu dieser Anwendung gesucht. Da id1 weiterhin gesperrt ist, wird eine zweite logische Verbindung id2 aufgebaut und ein zweiter Auftragnehmer-Vorgang v2 in der Auftragnehmer-Anwendung "appl" gestartet. Die Betriebsmittel (globale PARAMETER-Einstellungen, permanente DRIVE-Programm-Objekte) werden initialisiert. Dieser Vorgang v2 führt das DRIVE-Programm "abfrage" aus.
Da "abfrage" mit END PROC beendet wird, bleiben der Auftragnehmer-Vorgang v2 und die logische Verbindung id2 offen.

(4) Bei diesem dritten CALL-Aufruf eines DRIVE-Programms in der Auftragnehmer-Anwendung "appl" wird die zweite logische Verbindung id2 verwendet, da die erste v1 weiterhin gesperrt ist. Das DRIVE-Programm "buchung" wird im Auftragnehmer-Vorgang v2 ausgeführt, die zuvor definierten globalen und permanenten Betriebsmittel werden weiterverwendet.

Da "buchung" Transaktionsende anfordert, wird die logische Verbindung id2 von UTM bis zum Beenden der Verteilten Transaktion gesperrt, d.h. bis das Auftraggeber-DRIVE-Programm COMMIT WORK durchläuft. Der Auftragnehmer-Vorgang v2 bleibt weiterhin offen.

(5) Mit der COMMIT WORK-Anweisung im Auftraggeber-DRIVE-Programm wird die Verteilte Transaktion beendet, d.h. es werden alle lokalen Transaktionen in den Auftragnehmer-Vorgängen v1 und v2 sowie im Auftraggeber-Programm beendet.
Die Auftragnehmer-Vorgänge v1 und v2 bleiben offen. Die logischen Verbindungen id1 und id2 werden von UTM entsperrt und sind ebenfalls wieder offen.

(6) Der Verarbeitungsblock für Remote-CALL-Anweisungen beginnt.

(7),(8) Für die zwei CALL-Aufrufe des DRIVE-Programms "abfrage" werden zwei logische Verbindungen zu der Auftragnehmer-Anwendung gebraucht. Die Verarbeitung beginnt erst bei END DISPATCH.

(9) Die beiden CALL-Anweisungen werden parallel ausgeführt. Für CALL (7) wird die logische Verbindung id1 verwendet, für CALL (8) die Verbindung id2 zum Vorgang v2. Die zuvor definierten globalen und permanenten Betriebsmittel werden weiterverwendet.
Da das Progamm "abfrage" in beiden Fällen mit STOP beendet wird, wird für v1 und v2 Vorgangsende angefordert.

(10) Mit der COMMIT WORK-Anweisung im Auftraggeber-DRIVE-Programm wird die Verteilte Transaktion beendet, d.h. es werden alle lokalen Transaktionen in den Auftragnehmer-Vorgängen v1 und v2 sowie im Auftraggeber-Programm beendet.

Außerdem beendet UTM die Auftragnehmer-Vorgänge v1 und v2 und gibt die logischen Verbindungen id1 und id2 frei. Es gibt damit keine Verbindung mehr zwischen dem Auftraggeber-DRIVE-Programm und der Auftragnehmer-Anwendung "appl".

13.1.6 Zulässigkeit von DRIVE-Anweisungen abhängig vom Auftragnehmer

Abhängig vom Vorgangs- und Transaktionsstatus der zum jeweiligen Zeitpunkt adressierten Auftragnehmer sind bestimmte DRIVE-Anweisungen erlaubt oder führen zum Programmabbruch (siehe unten und UTM, UTM-D Programmieren [29]).

- Regeln, die nur für DRIVE-Programme auf der obersten Programmstufe in der obersten Auftraggeber-Umgebung einer mehrstufigen Hierarchie gelten (DO-Level):
  - Wird die Anweisung END PROCEDURE erreicht, müssen alle zuvor adressierten Auftragnehmer-Vorgänge beendet sein, und zwar:
    - durch COMMIT WORK WITH STOP oder STOP im letzten Auftragnehmer-DRIVE-Programm
    - durch PEND FI im letzten anwendereigenen UTM-Teilprogrammlauf.

    Andernfalls wird das DRIVE-Programm fehlerhaft abgebrochen.
  - Die Anweisung DO PROC ist nur auf dieser Programmstufe erlaubt. Wird diese Anweisung erreicht, müssen alle zuvor adressierten Auftragnehmer-Vorgänge beendet sein. Andernfalls wird das DRIVE-Programm fehlerhaft abgebrochen.
  - Wird das Programm in oberster Auftraggeber-Umgebung fehlerhaft abgebrochen, wird der Programm-Modus wie im lokalen Betrieb abgebrochen. DRIVE/WINDOWS schreibt Fehlermeldungen wie im lokalen Betrieb in die zentrale Druckdatei. Eventuell offene Auftragnehmer-Vorgänge werden ebenfalls beendet:

    Sind DRIVE-Auftragnehmer-Vorgänge offen, beendet DRIVE/WINDOWS diese intern automatisch.

    Sind Vorgänge anwendereigener UTM-Teilprogramme in C/COBOL offen, setzt DRIVE/WINDOWS intern für jeden dieser offenen Vorgänge einen Aufruf mit Aufrufcode „R“ oder „F“ ab (siehe Abschnitt „Header-Informationen im Datenübergabebereich“ auf Seite 316).
    Hat ein Auftragnehmer-Vorgang schon Transaktionsende angefordert, wird der Auftraggeber-Vorgang abgebrochen und damit die Verteilte Transaktion zurückgesetzt.

- Regeln, die für Auftraggeber-DRIVE-Programme unabhängig von der Auftraggeber-Auftragnehmer-Hierarchiestufe gelten:
  - STOP ist nur auf der obersten DRIVE-Programmmstufe des aktuellen Auftraggeber-Vorgangs erlaubt.
    Wird die Anweisung STOP oder COMMIT WORK WITH STOP erreicht, müssen alle zuvor adressierten Auftragnehmer-Vorgänge beendet sein, und zwar:
    - durch COMMIT WORK WITH STOP oder STOP im letzten Auftragnehmer-DRIVE-Programm
    - durch PEND FI im letzten anwendereigenen UTM-Teilprogrammlauf.

    Andernfalls wird das DRIVE-Programm fehlerhaft abgebrochen.

    Die Anweisung STOP ist wie im lokalen Betrieb nur erlaubt, wenn keine lokale Datenbank-Transaktion offen ist. Andernfalls können Sie die Anweisung COMMIT WORK WITH STOP verwenden.
  - Die Anweisung COMMIT WORK ist nur erlaubt, wenn alle zuvor adressierten Auftragnehmer-Vorgänge Transaktions- oder Vorgangsende angemeldet haben. D.h.:
    - Im letzten Auftragnehmer-DRIVE-Programm wurde COMMIT WORK, COMMIT WORK WITH STOP oder STOP erreicht.
    - Im letzten anwendereigenen UTM-Teilprogramm in C/COBOL wurde PEND RE oder PEND FI erreicht.

    Andernfalls wird das Auftraggeber-DRIVE-Programm fehlerhaft abgebrochen.
  - Mit der Anweisung ROLLBACK WORK werden alle Datenbank- und UTM-Transaktionen des aktuellen Auftraggeber-Vorgangs und der von ihm adressierten Auftragnehmer-Vorgänge zurückgesetzt.
    Sind DRIVE-Auftragnehmer-Vorgänge offen, setzt DRIVE/WINDOWS intern automatisch zurück. Sind Vorgänge mit anwendereigenen UTM-Teilprogrammen in C/COBOL offen, setzt DRIVE/WINDOWS intern für jeden offenen Vorgang einen Aufruf mit dem Aufruf-Code „K“ ab (siehe Abschnitt „Header-Informationen im Datenübergabebereich“ auf Seite 316).
    Anschließend wird die Verarbeitung im Auftraggeber mit der Anweisung fortgesetzt, die auf die ROLLBACK WORK-Anweisung folgt. Die DRIVE-Variablen werden nicht zurückgesetzt. Hat ein an der VTV beteiligter Auftragnehmer bereits Transaktionsende angefordert, wird Vorgangswiederanlauf ausgelöst und wie bei ROLLBACK WORK WITH RESET verfahren.
- Die Anweisung ROLLBACK WORK WITH RESET setzt alle Datenbank-, UTM- und Programmvorgänge, die an der VTV beteiligt sind, auf den letzten Sicherungspunkt zurück, d.h. auch DRIVE-Variablen. Aufgrund der BOTTOM-UP-Strategie setzt DRIVE/WINDOWS die Verarbeitung auf der obersten Programmstufe im obersten Auftragge-

ber (DO-Level) mit der Anweisung fort, die auf den letzten Sicherungspunkt folgt. Wurde noch kein COMMIT WORK durchlaufen, wird das Programm abgebrochen. Alle zu diesem Zeitpunkt offenen Auftragnehmer-Vorgänge werden ebenfalls abgebrochen.

Bei DRIVE-Programmen, die oberster Auftraggeber in einer mehrstufigen Auftraggeber-Auftragnehmer-Hierarchie sind, gilt:

Wurde die Anweisung COMMIT WORK beim letzten Sicherungspunkt mit der Klausel WITH display/send message, d.h. mit Bildschirmaus-/-eingabe verarbeitet (PEND RE siehe folgende Tabelle), wird dem Anwender das Rücksetzen gemeldet. UTM gibt den Bildschirm des letzten Sicherungspunktes aus, und DRIVE/WINDOWS führt die Bildschirmeingabe (Bestätigung) durch. Danach setzt DRIVE/WINDOWS die Verarbeitung wie allgemein in Auftraggeber-Umgebung fort, d.h., mit der Anweisung, die auf den letzten Sicherungspunkt folgt.

13.1.7 Interne PEND-Varianten von DRIVE-Anweisungen

Die Synchronisation von SQL-Transaktionen der Datenbanksysteme SESAM/SQL, UDS/SQL und INFORMIX mit UTM-Transaktionen bedeutet, daß UTM und das zugeordnete Datenbanksystem ihre Transaktionen gemeinsam beenden. Eine UTM-Transaktion enthält daher entweder keine oder genau eine (vollständige) SQL-Transaktion. Eine SQL-Transaktion wird beendet (COMMIT WORK) oder zurückgesetzt (ROLLBACK WORK), indem die zugehörige UTM-Transaktion beendet oder zurückgesetzt wird. DRIVE/WINDOWS setzt daher die SQL-Anweisungen COMMIT WORK und ROLLBACK WORK in die entsprechenden KDCS-Aufrufe um. Zum PEND SP siehe auch Abschnitt „Transaktionssicherung“ auf Seite 272.

Die folgende Tabelle zeigt, welche DRIVE-Anweisungen intern zum Ausführen welcher PEND-Variante führen. Je nach Vorgangs- (KCVGST) und Transaktionsstatus (KCTAST) der adressierten Auftragnehmer-Vorgänge sind die DRIVE-Anweisungen erlaubt oder führen zu Programmabbruch. Bei Remote-CALL ist nur der Vorgangs- und Transaktions-Status des jeweils zu adressierenden Auftragnehmers relevant [1].

Welchen Vorgangs- und Transaktionsstatus ein Auftragnehmer-Vorgang gegenüber seinem Auftraggeber-Vorgang hat, hängt davon ab, wie der letzte Auftragnehmer-Programmlauf beendet wurde (siehe Abschnitt „Programmierregeln“ auf Seite 358).

Der PEND SP wird von DRIVE/WINDOWS bei der Anweisung COMMIT WORK ohne WITH-Klausel in oberster Auftraggeber-Umgebung einer mehrstufigen Hierarchie unterstützt. D.h., die Transaktion wird beendet und der Programmlauf mit der nächsten Anweisung fortgesetzt.

[1] siehe Abschnitt „DRIVE-Programm als Auftraggeber“ auf Seite 341, Abschnitt „DRIVE-Programm als Auftragnehmer“ auf Seite 356 und UTM, UTM-D Programmieren [29].

Auftraggeber		**Auftragnehmer**		
PEND-Variante	**DRIVE-Anweisung**	**Vorgangs-status KCVGST**	**Transaktions-status KCTAST**	**Bedeutung**
KP	DISPLAY FILL-Überlauf SEND MESSAGE REMOTE CALL	O	O	erlaubt " " "
SP	COMMIT WORK			PRAB [1]
RE	COMMIT... WITH display/send ...			PRAB [1]
FI/FC	COMMIT WITH STOP STOP [WITH Folgetac]			PRAB [2] PRAB [2]
KP	DISPLAY FILL-Überlauf SEND MESSAGE REMOTE CALL	O	P	erlaubt " " erlaubt; neue Vorgangs-ID
SP	COMMIT WORK			erlaubt
RE	COMMIT ... WITH display/send...			erlaubt
FI/FC	COMMIT WITH STOP STOP [WITH Folgetac]			PRAB [2] PRAB [2]

Auftraggeber		Auftragnehmer		
PEND-Variante	**DRIVE-Anweisung**	**Vorgangs-status KCVGST**	**Transaktions-status KCTAST**	**Bedeutung**
KP	DISPLAY FILL-Überlauf SEND MESSAGE REMOTE CALL	O	C	erlaubt " " "
SP	COMMIT WORK			erlaubt
RE	COMMIT ... WITH display/send ...			erlaubt
FI/FC	COMMIT WITH STOP STOP [WITH Folgetac]			PRAB [2)] PRAB [2)]
KP	DISPLAY FILL-Überlauf SEND MESSAGE REMOTE CALL	C	P	erlaubt " " erlaubt; neue Vorgangs-ID
SP	COMMIT WORK			erlaubt
RE	COMMIT ... WITH display/send ...			erlaubt
FI/FC	COMMIT WITH STOP STOP [WITH Folgetac]			erlaubt erlaubt

Auftraggeber		**Auftragnehmer**		**Bedeutung**
PEND-Variante	**DRIVE-Anweisung**	**Vorgangs-status KCVGST**	**Transaktions-status KCTAST**	
KP	DISPLAY FILL-Überlauf SEND MESSAGE REMOTE CALL	C	C	erlaubt " " "
SP	COMMIT WORK			erlaubt
RE	COMMIT ... WITH display/send ...			erlaubt
FI/FC	COMMIT WITH STOP STOP [WITH Folgetac]			erlaubt
[1] Programmabbruch, Transaktionsregel verletzt [2] Programmabbruch, Vorgangsregel verletzt				

Zum Zeitpunkt der END PROCEDURE-Anweisung in einem mit DO gerufenen Programm dürfen keine Auftragnehmer-Vorgänge mit Vorgangsstatus „O“ bzw. „C“ und Transaktions-Status „O“ bzw. „P“ vorhanden sein. Andernfalls wurde das DRIVE-Programm fehlerhaft ausgeführt.

Sind bei fehlerhaften Programmabbrüchen noch offene Auftragnehmer-Vorgänge vorhanden, so veranlaßt DRIVE/WINDOWS diese mit einem Sonderaufruf zu einem PEND FI (siehe Abschnitt „Zulässigkeit von DRIVE-Anweisungen abhängig vom Auftragnehmer“ auf Seite 350 und Abschnitt „Header-Informationen im Datenübergabebereich“ auf Seite 316).

13.1.8 Datenübergabe

Bei Remote-Aufruf können **zwischen zwei DRIVE-Programmen** folgende USING-Parameter übergeben werden:

- Literale
- Variablen (einfache Variablen, Vektoren, Matrizen, Strukturen)
- Aggregate
- Ausdrücke

Zwischen DRIVE-Programmen übernimmt das DRIVE-Run-Time-System (RTS) vollständig den Austausch von Parameterwerten, deren Ablage und Auswertung. Insgesamt darf der USING-Übergabebereich, d.h. USING-Daten und die dazugehörigen Typbeschreibungen, maximal 32000Bytes groß sein.

Für den Nachrichten- und Datenaustausch **zwischen DRIVE/WINDOWS und einem anwendereigenen UTM-Teilprogramm in C/COBOL** wird ein Datenübergabebereich verwendet, der maximal 32000 Bytes groß sein darf (siehe Abschnitt „Datenübergabe bei UTM-Teilprogrammen in C/COBOL im Server (für C/COBOL-Programmierer)“ auf Seite 316). Als USING-Parameter können nur Variablen übergeben werden (einfache Variablen, Vektoren, Matrizen, beliebige Variablen-Strukturen).

13.2 DRIVE-Programm als Auftragnehmer

Ein DRIVE-Programm kann auch der Auftragnehmer eines DRIVE-Programms oder eines UTM-Teilprogramms in C/COBOL sein. Der folgende Abschnitt gilt für DRIVE-Programme, die in Auftragnehmer-Umgebung ablaufen. Das sind auch Auftragnehmer-Umgebungen, die selbst wieder Auftraggeber in einer mehrstufigen Hierarchie sind. Regeln, die aus der Auftraggeber-Sicht von DRIVE-Programmen folgen, sind im Abschnitt „DRIVE-Programm als Auftraggeber“ auf Seite 341 beschrieben.

13.2.1 Auswirkungen von DRIVE-Anweisungen auf VTV

Die folgenden DRIVE-Anweisungen beenden den Auftragnehmer-Programmlauf und wirken sich damit auf die VTV aus (siehe auch Tabelle in Abschnitt „Programmierregeln“ auf Seite 358):

END PROC (auf oberster Programmstufe, d.h. auf der Stufe, die vom Auftraggeber direkt aufgerufen wurde)

COMMIT WORK

COMMIT WORK WITH STOP

STOP

ROLLBACK WORK WITH RESET

– END PROCEDURE auf der obersten Programmstufe von DRIVE/WINDOWS als Auftragnehmer, d.h. auf der Stufe, die vom Auftraggeber direkt aufgerufen wurde, bewirkt Rücksprung zum Auftraggeber. Die Auftragnehmer-Transaktion wird nicht beendet. Der aktuelle Auftragnehmer-Vorgang kann weitere DRIVE-Programme ausführen (siehe auch Abschnitt „Logische Verbindung zwischen Auftraggeber und Auftragnehmer“ auf Seite 345 und Abschnitt „Programmierregeln“ auf Seite 358).

– STOP ist nur auf der obersten Programmstufe von DRIVE/WINDOWS als Auftragnehmer erlaubt, d.h. auf der Stufe, die vom Auftraggeber direkt aufgerufen wurde. Außerdem ist STOP wie im lokalen Betrieb nur erlaubt, wenn keine lokale Datenbank-Transaktion offen ist. Andernfalls können Sie die Anweisung COMMIT WORK WITH STOP verwenden.
DRIVE/WINDOWS meldet bei UTM das Ende des aktuellen Vorgangs an. Sobald die Verteilte Transaktion insgesamt beendet ist, wird auch der Auftragnehmer-Vorgang beendet und kann keine weiteren DRIVE-Programme ausführen.

– COMMIT WORK darf wie im lokalen Betrieb auf beliebiger Programmstufe im Auftragnehmer-DRIVE-Vorgang durchlaufen werden und bewirkt Rücksprung zum Auftraggeber. DRIVE/WINDOWS meldet bei UTM Transaktionsende an. Der Auftragnehmer-Vorgang wird gesperrt. Sobald die Verteilte Transaktion insgesamt beendet ist, wird die lokale Transaktion tatsächlich beendet. Der Auftragnehmer-Vorgang wird entsperrt und kann weitere DRIVE-Programme ausführen (siehe Abschnitt „Programmierregeln“ auf Seite 358 und Abschnitt „Logische Verbindung zwischen Auftraggeber und Auftragnehmer“ auf Seite 345).

– COMMIT WORK WITH STOP ist nur auf der obersten Programmstufe von DRIVE/WINDOWS als Auftragnehmer erlaubt, d.h. auf der Stufe, die vom Auftraggeber direkt aufgerufen wurde. DRIVE/WINDOWS meldet bei UTM das Ende des aktuellen Vorgangs an. Sobald die Verteilte Transaktion insgesamt beendet ist, wird auch der Auftragnehmer-Vorgang beendet und kann keine weiteren DRIVE-Programm ausführen.

– Die Anweisung ROLLBACK WORK WITH RESET setzt alle Datenbank-, UTM- und Programmvorgänge, die an der VTV beteiligt sind, auf den letzten Sicherungspunkt zurück, d.h. auch DRIVE-Variablen. Da DRIVE/WINDOWS bei den Programmierregeln die BOTTOM-UP-Strategie voraussetzt (siehe Abschnitt „DRIVE-Programm als Auftraggeber“ auf Seite 341 und Abschnitt „Programmierregeln“ auf Seite 358), wird immer auf den obersten Auftraggeber zurückgesetzt.

  Die Anweisung ROLLBACK WORK hingegen beendet nicht den Auftragnehmer-Programmlauf. Sie setzt die lokale Datenbank- und UTM-Transaktion zurück. Anschließend wird die Verarbeitung im Auftragnehmer mit der Anweisung fortgesetzt, die auf die ROLLBACK WORK-Anweisung folgt. Die DRIVE-Variablen werden nicht zurückgesetzt.

– Wird das Auftragnehmer-Programm fehlerhaft abgebrochen, wird wie im lokalen Betrieb der Programm-Modus beendet und eine offene lokale Transaktion zurückgesetzt. DRIVE/WINDOWS schreibt Fehlermeldungen wie im lokalen Betrieb in die zentrale Druckdatei. Bei Verbindung mit einem anwendereigenen UTM-Teilprogramm in C/COBOL versorgt DRIVE/WINDOWS außerdem das Header-Feld Aufruf-Code mit dem Wert „E“ und hinterlegt Diagnosemeldungen im Übergabebereich (siehe Abschnitt „Header-Informationen im Datenübergabebereich“ auf Seite 361).

DRIVE/WINDOWS meldet bei UTM das Ende des aktuellen Vorgangs an. Sobald die Verteilte Transaktion insgesamt beendet ist, wird der Auftragnehmer-Vorgang tatsächlich beendet und kann keine weiteren DRIVE-Programme ausführen.

In Auftragnehmer-Programmen, die mit Remote-CALL aufgerufen werden, sind folgende Anweisungen nicht erlaubt (siehe „DRIVE-Lexikon“ [3]):

FILL

DISPLAY

SEND MESSAGE

COMMIT WORK WITH DISPLAY

COMMIT WORK WITH SEND MESSAGE

STOP WITH DISPLAY

STOP WITH charausdruck

CALL TAC (lokal)

DO

In Auftragnehmer-Programmen, die mit Remote-ENTER aufgerufen werden, sind Remote-CALL-Anweisungen und lokale CALL TAC-Anweisungen nicht erlaubt. Ansonsten gelten die gleichen Einschränkungen wie für den lokalen Asynchron-Betrieb (siehe Abschnitt „UTM-Asynchronvorgänge“ auf Seite 16 und „DRIVE-Lexikon“ [3]).

13.2.2 Programmierregeln

Auch als Auftragnehmer setzt ein DRIVE-Programm die BOTTOM-UP-Strategie bei Partnervorgängen in anderen Programmiersprachen voraus, also bei anwendereigenen UTM-Teilprogrammen in C/COBOL) . D.h. ein Auftraggeber hat gegenüber einem DRIVE-Programm als Auftragnehmer immer den Vorgangs- und Transaktions-Status „O“. Damit sind im Auftragnehmer alle DRIVE-Anweisungen zulässig, die zu internem PEND führen mit Ausnahme von Bildschirmein- und -ausgaben (FILL, DISPLAY, SEND MESSAGE).

Die folgende Tabelle zeigt, welchen Vorgangs- und Transaktionsstatus ein Auftragnehmer-Vorgang abhängig davon hat, wie der letzte Auftragnehmer-Programmlauf beendet wurde.

Auftragnehmer				
letzte DRIVE-Anweisung	**PEND-Variante**	**Vorgangsstatus KCVGST**	**Transaktionsstatus KCTAST**	**Bedeutung**
END PROC	KP	O	O	unmittelbar nach CALL-Aufruf im AG
COMMIT WORK	RE	O	P	unmittelbar nach CALL-Aufruf im AG
COMMIT WORK	RE	O	C	unmittelbar nach COMMIT WORK im AG, das auf CALL folgt
STOP, COMMIT WORK WITH STOP	FI	C	P	unmittelbar nach CALL-Aufruf im AG
STOP, COMMIT WORK WITH STOP	FI	C	C	unmittelbar nach COMMIT WORK oder STOP im AG nach CALL

13.2.3 Gültigkeit von Definitionen

Ein DRIVE-Programm als Auftragnehmer bietet die gleiche Funktionalität an DRIVE-Variablen sowie temporären und permanenten SQL-Objekten wie ein lokal mit CALL gerufenes DRIVE-Programm. Dies gilt insbesondere für Gültigkeitsbereich und -dauer dieser Objekte und der globablen PARAMETER-Einstellungen, d.h. auch der mit PARAMETER DISTRIBUTION festgelegten Verteilungsinformationen (zu Gültigkeitsbereich und Lebensdauer von Definitionen siehe DRIVE-Programmiersprache [2], Abschnitt „Auswirkungen der Transaktionssteuerung auf Definitionen“ auf Seite 281).

Die DRIVE-Betriebsmittel gelten nur vorgangs-spezifisch. Ein erneuter Aufruf eines DRIVE-Programms in derselben Auftragnehmer-Anwendung kann also nur dann auf die zuvor permanent und global definierten und belegten Betriebsmittel wieder zugreifen, wenn dieser Aufruf im Auftraggeber-Vorgang über dieselbe logische Verbindung geht (siehe auch Abschnitt „Logische Verbindung zwischen Auftraggeber und Auftragnehmer“ auf Seite 345).

13.2.4 Datenübergabe

Wie beim lokalen CALL-Aufruf eines DRIVE-Programms werden Parameter in der USING-Klausel der PROCEDURE-Anweisung definiert. Sind im DRIVE-Programm der obersten Programmstufe in Auftragnehmer-Umgebung(also im DRIVE-Programm, das vom Auftraggeber direkt aufgerufen wurde) USING-Parameter mit RETURN definiert, übergibt DRIVE/WINDOWS die entsprechenden RETURN-Werte bei fehlerfreiem Rücksprung zum Auftraggeber. RETURN-Werte werden bei der Anweisung END PROC im Auftragnehmer-Programm übergeben sowie bei COMMIT WORK, COMMIT WORK WITH STOP und STOP.

Bei der Anweisung COMMIT WORK auf einer unteren Programmstufe in Auftragnehmer-Umgebung (also einem DRIVE-Programm, das nicht direkt vom Auftraggeber aufgerufen wurde) werden die RETURN-Parameterwerte aller Programmstufen zurückgegeben, die zwischen dem aktuellen und dem obersten Auftragnehmer-Programm liegen, das direkt vom Auftraggeber aufgerufen wurde. D.h. die RETURN-Parameter werden jeweils vom gerufenen zum rufenden DRIVE-Programm zurückgegeben.

Zwischen zwei DRIVE-Programmen kann das Auftragnehmer-DRIVE-Programm nur Variablen als USING-Parameter übergeben:

– einfache Variablen, Vektoren, Matrizen, beliebige Strukturen falls das Programm aus einem Auftraggeber-DRIVE-Programm aufgerufen wurde

– einfache Variablen, Vektoren, Matrizen sowie beliebige Strukturen, falls das Programm aus einem Auftraggeber-C/COBOL-Teilprogramm aufgerufen wurde (siehe DRIVE-Lexikon [3]).

Zwischen DRIVE-Programmen übernimmt das DRIVE-Run-Time-System (RTS) vollständig den Austausch von Parameterwerten, deren Ablage und Auswertung. Insgesamt darf der USING-Übergabebereich. d.h. USING-Daten und die dazugehörigen Typbeschreibungen, maximal 32000 Bytes groß sein.

Für den Nachrichten- und Datenaustausch **zwischen DRIVE/WINDOWS und einem anwendereigenen UTM-Teilprogramm in C/COBOL** wird ein Datenübergabebereich verwendet, der maximal 32000 Bytes groß sein darf (siehe Abschnitt „Datenübergabe bei UTM-Teilprogrammen in C/COBOL im Server (für C/COBOL-Programmierer)“ auf Seite 361).

13.2.5 Asynchrone Auftragnehmer-DRIVE-Programme

Für DRIVE-Programme, die asynchron als Auftragnehmer ablaufen, gelten die gleichen Regeln wie für den lokalen Asynchron-Betrieb (BATCH-Programme) (siehe DRIVE-Programmiersystem [1]). Allerdings kann ein asynchron ablaufendes DRIVE-Programm als Auftragnehmer keine CALL-Anweisungen remote verarbeiten, sondern nur ENTER-Anweisungen. Außerdem sind lokale CALL TAC-Anweisungen nicht erlaubt.

13.3 Datenübergabe bei UTM-Teilprogrammen in C/COBOL

Zum Nachrichtenaustausch zwischen einem anwendereigenen UTM-Teilprogramm in C/COBOL und einem DRIVE-Programm wird ein Übergabebereich verwendet.

Der Übergabebereich wird in DRIVE/WINDOWS als Auftraggeber verwendet bei

– Remote-CALL, Remote-ENTER

– lokalem CALL TAC, lokalem ENTER TAC.

Der Übergabebereich wird in DRIVE/WINDOWS vorausgesetzt bei

– synchronen und asynchronen Remote-Aufrufen des DRIVE-Auftragnehmers

– Rückkehr in ein DRIVE-Programm aus einem anwendereigenen UTM-Teilprogamm in C/COBOL bei lokalem CALL TAC-Aufruf (PEND PA-Logik).

Die Daten im Übergabebereich liegen in netzinvarianter Form (abdruckbar) vor. Beim Nachrichtenaustausch zwischen BS2000- und SINIX-Anwendungen setzt UTM(SINIX) zwischen EBCDI- und ASCII-Code um und umgekehrt. Voraussetzung: Bei der Generierung der SINIX-Anwendung ist in der SESCHA-Anweisung der Operand MAP auf SYSTEM gesetzt (siehe UTM : Anwendungen generieren und administrieren [28]).
In den anwendereigenen UTM-Teilprogrammen in C/COBOL müssen Sie bei MPUT-/FPUT-Anweisungen immer das Formatkennzeichen mit Leerzeichen belegen. Andernfalls setzt UTM (SINIX) EBCDI-Code nicht in ASCII-Code um und umgekehrt.

Der Übergabebereich besteht aus zwei Teilen:

– Header-Informationen, die immer vorhanden sind und eine feste Länge haben

– USING-Daten beim Übergeben von Parametern bzw. Diagnosemeldungen.

13.3.1 Header-Informationen im Datenübergabebereich

Der Header hat eine feste Länge von 144 Byte. Alle Informationen im Header werden in abdruckbarer Form hinterlegt. Dies gilt auch für alle numerischen Informationen (Längenfelder). Die folgende Tabelle zeigt das Layout des Header und die Länge der einzelnen Header-Felder für die Kommunikation von DRIVE/WINDOWS mit einem anwendereigenen UTM-Teilprogramm in C/COBOL.

Header-Feld	**Länge in Byte**	
	COBOL	**C**
1) Versionskennzeichen	8 (+ 1 Filler)	9
2) Name des Folgetac bei PEND PA	8 (+ 1 Filler)	9

Header-Feld	Länge in Byte	
	COBOL	**C**
3) Aufruf-Code	1	1
4) Direktive (DRIVE/Benutzer)	1	1
5) Gesamtlänge des Übergabereichs	6 (+ 1 Filler)	7
6) Länge des Bibliotheksnamens	6 (+ 1 Filler)	7
7) Bibliotheksname	54 (+ 1 Filler)	55
8) Länge des Elementnamens	6 (+ 1 Filler)	7
9) Elementname	31 (+ 1 Filler)	32
10) Reservefeld	16	16

Bei der Kommunikation von DRIVE/WINDOWS mit einem UTM-Teilprogramm in C ist der Wert in jedem Header-Feld mit Ausnahme von Aufruf-Code (3) und Direktive (4) mit einem Nullbyte abgeschlossen.

Bei den Längenfeldern 5), 6) und 8) wird der numerische Längenwert abgelegt wie beim DRIVE-Datentyp SMALLINT für USING-Daten im Übergabebereich (siehe Abschnitt „Datenkonvertierung bei DRIVE als Client und C/COBOL-Teilprogrammen als Server“ auf Seite 325 und Abschnitt „Datenkonvertierung bei C/COBOL-Teilprogramen als Auftraggeber und DRIVE als Auftragnehmer“ auf Seite 373).

Bedeutung der Header-Felder:

1) Versionskennzeichen muß versorgt werden mit DRIVTV0

2) Name des Folgetac bei PEND PA ist nur relevant, wenn im DRIVE-Programm als Auftraggeber eine Anweisung CALL TAC lokal verarbeitet werden soll (siehe auch Feld 3 Aufruf-Code).

3) Aufruf-Code

   Für die Verbindung vom Auftraggeber zum Auftragnehmer sind folgende Werte erlaubt:

   – S: Standard-Aufruf, um normale Programmverarbeitung im Auftragnehmer-Vorgang zu starten

   – R: Der Auftragnehmer-Vorgang hat den Transaktionsstatus „O“ und muß RSET und PEND FI auslösen.

   – K: Der Auftragnehmer-Vorgang hat den Transaktionsstatus „O“ und muß RSET und PEND KP auslösen.

– F: Der Auftragnehmer-Vorgang hat den Transaktionsstatus „C“ und muß PEND FI auslösen.

– A: Aufruf über PEND PA bei lokalem CALL TAC. Dieser Aufruf-Code ist nicht erlaubt, wenn DRIVE/WINDOWS der Auftragnehmer ist.

Die Aufruf-Codes R und F werden von DRIVE/WINDOWS verwendet, um bei fehlerhaftem Programmabbruch im Auftraggeber noch offene Auftragnehmer-Vorgänge zu beenden.
Der Aufruf-Code K wird von DRIVE/WINDOWS verwendet, um bei ROLLBACK WORK die Transaktionen der Auftragnehmer-Vorgänge zurückzusetzen.

Für die Verbindung vom Auftragnehmer zum Auftraggeber sind folgende Werte erlaubt:

– O: Aufruf wurde fehlerfrei beendet.

– E: Aufruf wurde mit Fehlern im Programmablauf beendet.

– P: Der Header wurde nicht korrekt versorgt (Parameterfehler), der Aufruf ist fehlerhaft.

4) Direktive

Das Feld gibt an, mit welchem Programmtyp DRIVE/WINDOWS kommuniziert und welches Layout der Übergabebereich hat. Folgende Werte sind möglich:

– S: Direktive COBOL (Standard)

– C: Direktive C

5) Gesamtlänge des Übergabebereichs

Die Gesamtlänge des Übergabebereichs schließt die Länge des Header mit ein und darf 32000 plus Headerlänge nicht überschreiten. Die tatsächliche maximale Gesamtlänge des Übergabebereichs hängt von der Generierung der UTM-Anwendung ab. Sie brerechnet sich aus der maximal generiereten SPAB-Länge minus 50 Byte für interne DRIVE-Verwaltung (siehe MAX-Anweisung bei KDCDEF-Steueranweisungen, DRIVE-Programmiersprache [1], Kapitel DRIVE/WINDOWS für den UTM-Betrieb generieren, Abschnitt „Beispiel für KDCDEF-Rahmen in der DRIVE-Anwendung“).

Ist die Gesamtlänge des Übergabebereichs größer als die Länge des Header, so schließen sich im Übergabebereich folgende Daten unmittelbar an den Header an in der Länge, die sich aus der Differenz von Gesamtlänge minus Headerlänge ergibt:

1. bei Aufruf-Code S oder A: USING-Daten des Auftraggebers an den Auftragnehmer

2. bei Aufruf-Code O: USING-Daten für RETURN-Parameter des Auftragnehmers zurück an den Auftraggeber
3. bei Aufruf-Code E oder P: Diagnosemeldungen des Auftragnehmers an den Auftraggeber

Die folgenden Felder sind nur bei Aufruf-Code S relevant, wenn ein DRIVE-Programm als Auftragnehmer aufgerufen werden soll.

6) Länge des Bibliotheksnamens

Gibt die Länge des Bibliotheksnamens an und darf Werte von 0 bis 54 enthalten.

7) Bibliotheksname

Name der DRIVE-Bibliothek. Die Länge des Namens muß im Feld 6 linksbündig hinterlegt sein.

8) Länge des Elementnamens

Gibt die Länge des Elementnamens an und darf Werte von 1 bis 31 enthalten.

9) Elementname

Name des Elements für das DRIVE-Programm. Die Länge des Namens muß im Feld 8 linksbündig hinterlegt sein.

10) Reservefeld wird zur Zeit nicht verwendet.

13.3.2 USING-Daten im Datenübergabebereich

Zwischen DRIVE/WINDOWS und einem anwendereigenen UTM-Teilprogramm in C/COBOL können nur Variablen, und zwar auch Vektoren, Matrizen und strukturierte Variablen als USING-Parameter übergeben werden.

Die USING-Daten des Übergabebereichs sind nur erforderlich, falls USING-Parameter angegeben werden. Datenwerte werden zwischen DRIVE/WINDOWS und anwendereigenen UTM-Teilprogrammen in C/COBOL grundsätzlich in abdruckbarer Form ausgetauscht. Dies gilt auch für numerische Werte.

DRIVE/WINDOWS übergibt Dezimalzeichen an ein anwendereigenes UTM-Teilprogramm in C/COBOL immer als „.“. In umgekehrter Richtung können anwendereigene UTM-Teilprogramme in C/COBOL Dezimalzeichen an DRIVE/WINDOWS entweder als „.“ oder als „,“ übergeben.

Verbindung DRIVE-Programme - anwendereigene UTM-Teilprogrammen in C und umgekehrt:

Datenwerte werden abhängig vom Wert in signifikanter Länge übergeben bzw. erwartet und grundsätzlich mit einem Nullbyte (X'00') abgeschlossen. Numerische Werte werden bei Zielsprache C mit Vorzeichen linksbündig, Datenwert linksbündig, unmittelbar gefolgt von einem Nullbyte übergeben. Das Vorzeichen wird nur bei negativen Werten übergeben. Indikatorwerte werden mit einem Nullbyte übergeben. Umgekehrt müssen Sie bei Zielsprache DRIVE/WINDOWS Indikatorwerte mit Nullbyte abschließen. Bei der Indikatoranzeige für den NULL-Wert wird kein Datenwert übergeben: Auf den Indikatorwert abgeschlossen mit Nullbyte folgt unmittelbar ein Nullbyte für den leeren Wert.

Da zwischen DRIVE/WINDOWS und anwendereigenen UTM-Teilprogrammen in C der Datenaustausch immer mit Nullbyte abgeschlossen wird, können Sie bei allen Datentypen die C-Bibliotheksfunktionen einsetzen.

Verbindung DRIVE-Programme - anwendereigene UTM-Teilprogramme in COBOL und umgekehrt:

Die Datenwerte werden abhängig vom Datentyp in fester Länge übergeben bzw. erwartet. Alle Daten folgen unmittelbar aufeinander, sie sind nicht durch Nullbyte getrennt. Die Datenwerte werden mit Vorzeichen linksbündig, Datenwert rechtsbündig (ggf. mit führenden Nullen aufgefüllt) übergeben und erwartet. Das Vorzeichen wird immer übergeben. Auch bei der Indikatoranzeige für den NULL-Wert wird, abhängig vom Datentyp, in fester Länge Platz reserviert. Der eigentliche Datenwert ist undefiniert.

Übergabeform und -länge

Die Übergabeform und -länge von Datenwerten ist abhängig vom Datentyp und der Programmiersprache des anwendereigenen UTM-Teilprogramms (C oder COBOL). Die folgende Tabelle zeigt Übergabeform und -länge am Beispiel, eine ausführliche Darstellung finden Sie in Abschnitt „Datenkonvertierung bei DRIVE als Client und C/COBOL-Teilprogrammen als Server“ auf Seite 325 und Abschnitt „Datenkonvertierung bei C/COBOL-Teilprogramen als Auftraggeber und DRIVE als Auftragnehmer“ auf Seite 373).

DRIVE-Datentyp Beispiel-Wert	**Übergabelänge in Byte (ohne Indikatorwertlänge)**	**Übergabe ohne/ mit Indikator**	**Übergabeformat des Beispielwerts**
SMALLINT 47	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	47# 00#47# -1##
	COBOL: 6	ohne Indikator mit Indikator: NOT NULL NULL	+00047 00+00047 -1...
INTEGER 47	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	47# 00#47# -1##
	COBOL: 11	ohne Indikator mit Indikator: NOT NULL NULL	+0000000047 00+0000000047 -1...
REAL 2345.67	C: wertabhängig (wegen Vorzeichen)	ohne Indikator mit Indikator: NOT NULL NULL	2.345670E+03# 00#2.345670E+03# -1##
	COBOL: 13	ohne Indikator mit Indikator: NOT NULL NULL	+2.345670E+03 00+2.345670E+03 -1...
FLOAT 2345.67123	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	+2.34567123000000E+03# 00#2.34567123000000E+03# -1##
	COBOL: 21	ohne Indikator mit Indikator: NOT NULL NULL	+2.34567123000000E+03 00+2.34567123000000E+03 -1...

DRIVE-Datentyp Beispiel-Wert	**Übergabelänge in Byte (ohne Indikatorwertlänge)**	**Übergabe ohne/ mit Indikator**	**Übergabeformat des Beispielwerts**
NUM/ DEC/ XDEC 234 in DEC(7,0) 234.56 in DEC(7,3)	C: wertabhängig	s = 0 ohne Indikator mit Indikator: NOT NULL NULL	 234# 00#234# -1##
		s > 0 ohne Indikator mit Indikator: NOT NULL NULL	 234.560# 00#234.560# -1##
		p = s ohne Indikator mit Indikator: NOT NULL NULL	 0.12345# 00#0.12345 -1##
	COBOL: p + 1 bei s = 0	s = 0 ohne Indikator mit Indikator: NOT NULL NULL	 +0000234 00+0000234 -1...
0,12345 in DEC(5,5) Bsp. DEC(5,5) Wert: 0,12345	p + 2 bei s > 0	s > 0 ohne Indikator mit Indikator: NOT NULL NULL	 +234.560 00+0234.560 -1...
	p + 3 bei p = s p = s Typen, die nur Nachkommastellen enthalten	p = s ohne Indikator mit Indikator: NOT NULL NULL	 +0.12345 00+0.12345 -1...
CHAR(n) 'abc' in CHAR(5)	C: n+1 *	ohne Indikator mit Indikator: NOT NULL NULL	abc␣␣ # 00abc␣␣ # -1##
	COBOL: n	ohne Indikator mit Indikator: NOT NULL NULL	abc␣␣ 00abc␣␣ -1...

DRIVE-Datentyp Beispiel-Wert	**Übergabelänge in Byte (ohne Indikatorwertlänge)**	**Übergabe ohne/ mit Indikator**	**Übergabeformat des Beispielwerts**
VARCHAR(n) 'abc'	C: wertabhängig	ohne Indikator mit Indikator: NOT NULL NULL	abc# 00#abc# -1##
	COBOL: akt. Länge + 6 bei COBOL-Verbindung ist immer ein Längenfeld vorhanden oder notwendig	ohne Indikator mit Indikator: NOT NULL NULL	+00003abc 00+00003abc -1+00000
DATE 19.01.96	C: 11 *	ohne Indikator mit Indikator: NOT NULL NULL	1996-01-19# 00#1996-01-19# -1##
	COBOL: 10	ohne Indikator mit Indikator: NOT NULL NULL	1996-01-19 001996-01-19 -1...
TIME 1 Sek. vor Mitter- nacht	C: 9 *	ohne Indikator mit Indikator: NOT NULL NULL	23:59:59# 00#23:59:59# -1##
	COBOL: 8	ohne Indikator mit Indikator: NOT NULL NULL	23:59:59 0023:59:59 -1...
TIME(3) 1 Hundertstel- Sekunde vor Mit- ternacht	C: 13 *	ohne Indikator mit Indikator: NOT NULL NULL	"23:59:59:999#" "00#23:59:59:999#" "-1##"
	COBOL: 12	ohne Indikator mit Indikator: NOT NULL NULL	"23:59:59:999" "0023:59:59:999" "-1....."

DRIVE-Datentyp Beispiel-Wert	Übergabelänge in Byte (ohne Indikatorwertlänge)	Übergabe ohne/ mit Indikator	Übergabeformat des Beispielwerts
TIMESTAMP(3) Datum 19.1.1996 Zeit 19.45.33 50 Hundertstel- Sekunden	C: 24 * COBOL: 23	ohne Indikator mit Indikator: NOT NULL NULL ohne Indikator mit Indikator: NOT NULL NULL	"1996-01-19 19:45:33.050#" "00#1996-01-19 19:45:33.050#" "1996-01-19 19:45:33.050" "001996-01-19 19:45:33.050" "-1....."
INTERVAL		wie bei INTEGER	

Legende
p = stellenanzahl
s = nachkommastellen
# = Nullbyte-Inhalt
... = undefinierter Inhalt
* Werte vom Datentyp CHAR, DATE und TIME, TIME(3) u. TIMESTAMP(3) werden bei Verbindung mit C immer in fester Länge übergeben bzw. erwartet. Eventuelle Nullbytes in den Datenwerten werden übertragen (bei CHAR) oder führen zu CONVERSION ERROR.

Indikatorwerte werden bei C und COBOL in fester Länge von 2 Byte abgelegt.

13.3.2.1 Aufbau der Übergabeparameter

Jeder Übergabebereich für die Verbindung zwischen DRIVE/WINDOWS und einem anwendereigenen UTM-Teilprogramm in C/COBOL und umgekehrt beginnt mit dem Header. Unmittelbar hinter diesem liegen die eigentlichen Datenwerte und ggf. Indikatoranzeigen für den NULL-Wert. Die Datenwerte sind in abdruckbarer Form abgelegt (zur Ablageform der einzelnen Datentypen siehe Abschnitt „Datenkonvertierung bei DRIVE als Client und C/COBOL-Teilprogrammen als Server“ auf Seite 325 und Abschnitt „Datenkonvertierung bei C/COBOL-Teilprogramen als Auftraggeber und DRIVE als Auftragnehmer“ auf Seite 373).

Ablageform der Indikatorwerte

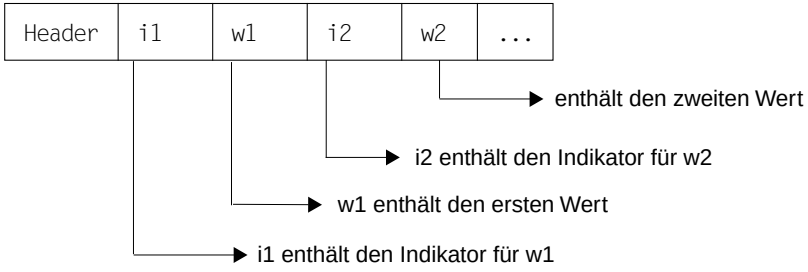
Indikatorwerte zeigen an, ob der folgende Datenwert ein NULL-Wert ist oder nicht. Sie werden in 2-Byte-Feldern in abdruckbarer Form unmittelbar vor dem eigentlichen Datenwert abgelegt. Dies gilt auch bei Vektoren, Matrizen und Strukturen: Der Indikatorwert für jede Komponente wird unmittelbar vor dem Komponentenwert abgelegt. Der Wert C’-1’ zeigt den NULL-Wert an, d.h. das unmittelbar folgende Feld enthält keinen signifikanten Datenwert. Der Indikatorwert C’00’ zeigt an, daß das folgende Feld keinen NULL-Wert , sondern

einen signifikaten Datenwert enthält. Ist ein Indikatorfeld vorhanden, muß es mit einem dieser beiden Werte versorgt sein, sonst meldet DRIVE/WINDOWS „CONVERSION ERROR“.

Ist der Auftraggeber ein anwendereigenes UTM-Teilprogramm in C/COBOL und der Auftragnehmer ein DRIVE-Programm, müssen Sie zu **allen** USING-Parameter Indikatorwerte übergeben. Beim Rücksprung zum Auftraggeber übergibt DRIVE/WINDOWS für alle RETURN-Parameter Indikatorwerte.

Ist der Auftraggeber ein DRIVE-Programm, so können Sie beim Aufruf eines anwendereigenen UTM-Teilprogramms in C/COBOL mit CALL/ENTER TAC die Indikator-Klausel angeben (siehe DRIVE-Lexikon [3] und Abschnitt „Daten an Unterprogramme in anderen Programmiersprachen übergeben“ auf Seite 112). DRIVE/WINDOWS übergibt Indikatorwerte nur für so spezifizierte USING-Parameter und erwartet beim Rücksprung Indikatorwerte auch nur bei den entsprechenden RETURN-Parametern.

Beispiel



Sind RETURN-Parameter angegeben, so müssen Sie beim Rücksprung vom Auftragnehmer zum Auftraggeber die USING-Daten für die RETURN-Parameter versorgen, wenn der Auftragnehmer ein UTM-Teilprogramm in C/COBOL ist. Ist der Auftragnehmer ein DRIVE-Programm, versorgt DRIVE/WINDOWS die USING-Daten für die RETURN-Parameter automatisch.

13.3.2.2 Datenkonvertierung bei DRIVE als Auftraggeber und C/COBOL-Teilprogrammen als Auftragnehmer

```
CALL/ENTER  ... TAC ... USING &r1,
                          RETURN &r2,
```

Ist das DRIVE-Programm Auftraggeber und ein UTM-Teilprogramm in C/COBOL Auftragnehmer, konvertiert DRIVE/WINDOWS die numerischen Werte in abdruckbare Form. Mit der INDIKATOR-Klausel wird für jeden USING-Parameter festgelegt, ob vor dem Datenwert

(bei Vektoren, Matrizen und Strukturen vor jedem Komponentenwert) ein Indikatorwert abgelegt wird. Das UTM-Teilprogramm kann ggf. die abdruckbaren Werte in interne Rechenformate übertragen.

Auf dem Rückweg konvertiert das UTM-Teilprogramm RETURN-Werte aus evtl. internen Rechenformaten in abdruckbare Form. Bei jedem RETURN-Parameter, der mit INDIKATOR-Klausel in der DRIVE-CALL-Anweisung angegeben ist, erwartet DRIVE/WINDOWS vor dem Datenwert (bei Vektoren, Matrizen und Strukturen vor dem Komponenetenwert) einen Indikatorwert. Das DRIVE-RTS (Runtime-System) kennt die Zieldatentypen der RETURN-Werte und konvertiert die abdruckbare Form in die interne DRIVE-Form.

Ablageform der einzelnen Datentypen

Die Übergabeform und -länge von Datenwerten ist abhängig vom Datentyp und der Programmiersprache des anwendereigenen UTM-Teilprogramms (C oder COBOL).

CHAR(n)

wird bei COBOL in Länge n übertragen, bei C in Länge (n+1).

VARCHAR(n)

wird in signifikanter Länge übertragen. Die Ablagefrom ist abhängig von der Programmiersprache:

C: String in signifikanter Länge, mit Nullbyte abgeschlossen

COBOL: Längenfeld (abdruckbar, Format wie bei SMALLINT), gefolgt von Wert. Sie können also Datenmengen in variabler Länge übertragen.

Verwenden Sie für COBOL-Anwendungen, die USING-Daten mit fester Länge erwarten, CHAR-Felder statt VARCHAR-Felder. Verwenden Sie für C-Anwendungen VARCHAR-Felder. Damit können Sie den Pointer jeweils um die C-String-Länge (strlen) +1 erhöhen.

INTEGER, INTERVAL

wird als Zahl übergeben, die aus Vorzeichen und einer ganzzahligen Ziffernfolge besteht.

SMALLINT

wird als Zahl übergeben, die aus Vorzeichen und einer ganzzahligen Ziffernfolge besteht.

REAL

wird als Zahl übergeben, die aus Vorzeichen und einer 7-stelligen Ziffernfolge besteht mit Dezimalpunkt, gefolgt von e oder E, gefolgt von einem Vorzeichen und abschließend einer ganzen Zahl.

FLOAT

wird als Zahl übergeben, die aus Vorzeichen und einer 15-stelligen Ziffernfolge besteht, gefolgt von e oder E, gefolgt von einem Vorzeichen und abschließend einer ganzen Zahl.

DEC(p,s), NUM(p,s), XDEC(p,s)

p = stellenanzahl; s = nachkommastellen

Der Daten-Wert kann ein Vorzeichen, einen Dezimalpunkt und Zahlen rechts davon haben.

DATE

Der Datumswert wird abdruckbar übergeben in der Form JJJJ-MM-TT (Jahr, Monat, Tag) .

TIME

Der Zeitwert wird abdruckbar übergeben in der Form HH:MM:SS (Stunden, Minuten, Sekunden).

Reaktionen bei fehlerhaften RETURN-Parametern

Werden RETURN-Parameter fehlerhaft übergeben oder verletzen sie die Datentypbeschreibung im Zielprogramm (ist z.B. der übergebene Wert zu groß für den Zieldatentyp), reagiert DRIVE/WINDOWS folgendermaßen:

– das Programm wurde außerhalb eines DISPATCH-Blocks aufgerufen:
  Meldung CONVERSION_ERROR; Aufrufen einer WHENEVER-Fehlerbehandlung, wenn sie vorhanden ist.

– das Programm wurde in einem DISPATCH-Block aufgerufen
  Meldung CONVERSION_ERROR, das Programm wird abgebrochen.

13.3.2.3 Datenkonvertierung bei C/COBOL-Teilprogramen als Auftraggeber und DRIVE als Auftragnehmer

```
PROC USING &r1 DEC(5,2)
          RETURN &r2 INTEGER
```

Das UTM-Teilprogramm in C/COBOL als Auftraggeber konvertiert numerische Daten aus evtl. internen Rechenformaten in abdruckbare Form. DRIVE/WINDOWS erwartet numerische Daten in abdruckbarer Form und konvertiert automatisch in die interne DRIVE-Form. Vor **jedem** Datenwert (bei Vektoren, Matrizen und Strukturen vor jedem Komponentenwert) erwartet DRIVE/WINDOWS einen Indikatorwert. Im DRIVE-Programm werden die Daten in der Form deklariert, in der sie verarbeitet werden sollen. Auf dem Rückweg konvertiert DRIVE/WINDOWS numerische RETURN-Werte aus dem internen DRIVE-Format in abdruckbare Form. Vor j**edem** Datenwert (bei Vektoren, Matrizen und Strukturen vor jedem Komponentenwert) legt DRIVE/WINDOWS einen Indikatorwert ab. Zur Weiterverarbeitung numerischer Daten durch ein UTM-Teilprogramm konvertiert dieses ggf. die abdruckbare Form in die interne Rechenform der entsprechenden Programmiersprache C oder COBOL.

Ablageform der einzelnen Datentypen

Das DRIVE-Programm erwartet die Datenwerte immer in fester Länge (abhängig vom Datentyp) in abdruckbarer Form.

CHAR(n)

DRIVE/WINDOWS übernimmt Zeichen in Länge n.

Insbesondere bei der Verbindung von UTM-Teilprogrammen in C mit DRIVE/WINDOWS und beim Übergeben von Strings vom Datentyp CHAR müssen Sie darauf achten, Literale ggf. mit Leerzeichen aufzufüllen, damit nicht eventuell vorhandene Endekennzeichen (X’00’) in die DRIVE-Variable übertragen werden.

VARCHAR(n)

DRIVE/WINDOWS übernimmt Daten in signifikanter Länge. Die Ablageform ist abhängig von der Programmiersprache:

C: String in signifikanter Länge, mit Nullbyte abgeschlossen

COBOL: Längenfeld (abdruckbar, Format wie bei SMALLINT), gefolgt von Wert. Sie können also Datenmengen in variabler Länge übertragen.

Verwenden Sie für COBOL-Anwendungen, die USING-Daten mit fester Länge übergeben, CHAR-Felder statt VARCHAR-Feldern.INTEGER, INTERVAL

DRIVE/WINDOWS übernimmt folgende Zeichen: eine Folge von Leerzeichen, Vorzeichen und eine ganzzahlige Ziffernfolge.

SMALLINT

DRIVE/WINDOWS übernimmt folgende Zeichen: eine Folge von Leerzeichen, Vorzeichen und eine ganzzahlige Ziffernfolge.

REAL, FLOAT

DRIVE/WINDOWS übernimmt folgende Zeichen: eine Folge von Leerzeichen, Vorzeichen, eine Ziffernfolge mit oder ohne Dezimalpunkt, gefolgt von e oder E, gefolgt von einem Vorzeichen und abschließend einer ganzen Zahl. Die Mantisse ist 7-stellig, der Exponent 2-stellig.

DEC(p,s), NUM(p,s), XDEC(p,s)

p = stellenanzahl; s = nachkommastellen

Führende Leerzeichen werden überlesen. Der Daten-Wert kann ein Vorzeichen, einen Dezimalpunkt und Zahlen rechts davon haben.

DATE

Der Datenwert wird abdruckbar erwartet in der Form JJJJ-MM-TT (Jahr, Monat, Tag) . .

TIME

Der Zeitwert wird abdruckbar erwartet in der Form HH:MM:SS (Stunden, Minuten, Sekunden).

Reaktionen bei fehlerhaften USING-Parametern

Wenn Datenwerte in ungültiger Form angeliefert werden oder z.B. zu groß sind für die Zielvariable, wird dies bereits beim Aufruf des DRIVE-Programms erkannt und der Übergabebereich mit der entsprechenden Fehlermeldung versorgt.

13.3.3 Diagnosebereich

Bei fehlerhaftem Ansprung des Auftragnehmers können im Diagnosebereich Diagnosemeldungen an den Auftraggeber zurückgegeben werden. Jede Meldung muß in einer festen Länge von 79 Byte abgelegt und ggf. mit Leerzeichen aufgefüllt werden.
Kommuniziert DRIVE/WINDOWS mit einem UTM-Teilprogramm in COBOL, wird unmittelbar hinter der Diagnosemeldung ein Filler der Länge 1 Byte vorgesehen oder erwartet.
Kommuniziert DRIVE/WINDOWS mit einem UTM-Teilprogrammm in C, ist jede Diagnosemeldung mit einem Nullbyte abgeschlossen oder abzuschließen. Für jede Diagnosemeldung im Diagnosebereich müssen Sie also 80 Byte reservieren.
Die Länge des gesamten Diagnosebereichs wird durch die Differenz bestimmt zwischen der Gesamtlänge des Übergabebereichs (siehe Header-Feld 5) und der festen Headerlänge von 144 Byte. Die Anzahl der Diagnosemeldungen ergibt sich also aus der ganzzahligen Division der Länge des gesamten Diagnosebereichs durch die feste Länge der Diagnosemeldungen von 80 Byte. Zur Zeit werden von DRIVE/WINDOWS maximal 32 Diagnosemeldungen verarbeitet. Diese Anzahl kann geringer werden je nach der maximalen Größe des Übergabebereichs.

13.4 VTV-Anwendung generieren

Zur Generierung einer VTV-Anwendung siehe DRIVE-Programmiersystem [1], Kapitel DRIVE/WINDOWS für UTM-Betrieb generieren.

13.5 Fehlerbehandlung

Bei KDCS-Aufrufen wertet DRIVE/WINDOWS die Felder KCRCCC (KDCS-Fehlercode) und KCRCDC (interner Fehlercode) aus, die von UTM im KB-Rückgabebereich versorgt werden.

KDCS-Fehlercodes bei APRO-Aufrufen

Die Fehlercodes 40Z,44Z und 46Z führen zu Programmabbruch, alle anderen Fehlercodes führen zum Vorgangsende.

KDCS-Fehlercodes bei MPUT-Aufrufen

Alle Fehlercodes führen zum Vorgangsende.

KDCS-Fehlercodes bei FPUT-Aufrufen

Die Fehlercodes 40Z und 44Z führen zu Programmabbruch, alle anderen Fehlercodes führen zum Vorgangsende.

KDCS-Fehlercodes bei MGET- und FGET-Aufrufen

Alle Fehlercodes führen zum Vorgangsende.

14 Dateien bearbeiten

Mit DRIVE/WINDOWS können Sie Dateien mit DRIVE-Programmen bearbeiten. Sie haben folgende Möglichkeiten der Dateibearbeitung:

– Dateien öffnen und schließen

– Dateipositionen ermitteln

– in einer Datei positionieren

– Daten aus einer Datei lesen und in eine Datei schreiben

– Daten in einer Datei löschen (nur ISAM-Dateien)

Folgende Dateiarten können Sie mit DRIVE/WINDOWS bearbeiten:

– SAM-Dateien
  Die SAM-Datei wird sequentiell bearbeitet. Bei jedem Dateizugriff wird ein Datensatz bearbeitet. Die Daten sind binär gespeichert.

– ISAM-Dateien
  Die ISAM-Datei wird entweder sequentiell oder über einen Schlüsselbereich (ISAM-Schlüssel) verarbeitet. Bei jedem Dateizugriff wird ein Datensatz bearbeitet. Die Daten sind binär gespeichert.

  DRIVE/WINDOWS bearbeitet nur ISAM-Dateien ohne Schlüsselverdoppelung (DUPKEY=NO).

– Textdateien
  Die Textdatei ist eine spezielle SAM-Datei: Alle Felder sind durch ein Trennzeichen getrennt.
  Diese Dateiart wird aus Kompatibilitätsgründen zu SINIX und MS-Windows unterstützt.

– Binärdateien
  Die Binärdatei ist eine SAM-Datei.
  Diese Dateiart wird aus Kompatibilitätsgründen zu SINIX und MS-Windows unterstützt.

Den Aufbau einer zu erstellenden Datei legen Sie mit der WITH-Klausel in der WRITE FILE-Anweisung fest. Sie können in der WRITE FILE-Anweisung eine oder mehrere Variable(n) angeben.Die DRIVE-Variablen müssen so aufgebaut sein wie die Datensätze, die in die Datei geschrieben werden sollen. Durch die Definition unterschiedlicher Variablen lassen sich verschiedene Satzarten in eine Datei schreiben.

Den Aufbau einer bereits existierenden Datei müssen Sie kennen. Die DRIVE-Variable, die Sie in der READ FILE-Anweisung mit der INTO-Klausel angeben, muß so aufgebaut sein wie die Datensätze, die aus der Datei gelesen werden sollen. Sie können in der READ FILE-Anweisung eine oder mehrere Variable(n) angeben.

Beachten Sie, daß Textdateien zeilenorientiert verarbeitet werden. Außerdem müssen in Textdateien die einzelnen Felder durch ein Trennzeichen getrennt sein. DRIVE/WINDOWS erwartet Zeilenendezeichen und Trennzeichen beim lesenden Zugriff auf Dateien. Beim schreibenden Zugriff trennt DRIVE/WINDOWS die einzelnen Felder durch Trennzeichen und fügt am Ende ein Zeilendenzeichen ein. Welches Zeichen das Trennzeichen sein soll, bestimmen Sie in der OPEN FILE-Anweisung. Zeilenendezeichen und Dateiendezeichen interpretiert DRIVE/WINDOWS als gültige Trennzeichen, so daß am Zeilenende und am Dateiende das Trennzeichen entfallen kann.

Für die Darstellung von NULL-Werten müssen Sie ebenfalls ein Zeichen festlegen. Soll DRIVE/WINDOWS einen NULL-Wert in eine Datei schreiben, so ist dies nur möglich, wenn ein Zeichen für die NULL-Wertdarstellung in der DECLARE FILE-Anweisung festgelegt wurde. DRIVE/WINDOWS schreibt dieses Zeichen so oft in die Datei, bis das Feld, das den NULL-Wert hat, vollständig gefüllt ist und erwartet in der Datei dieses Zeichen, wenn ein NULL-Wert vorliegt.

Bei Programmende und Programmabbruch auf oberster Programmstufe (END PROCEDURE und BREAK PROCEDURE im Hauptprogramm) schließt DRIVE/WINDOWS automatisch alle geöffneten Dateien.

Die Bearbeitung von Dateien unterliegt - anders als die Bearbeitung von Datenbanken - nicht der Transaktionssicherung. Wenn Sie Transaktionen zurücksetzen, werden Änderungen, die Dateien betreffen, nicht zurückgesetzt. Wird z.B. innerhalb einer Transaktion ein Datensatz geschrieben, steht dieser Datensatz weiterhin in einer Datei, auch wenn die Transaktion zurückgesetzt wurde (siehe auch Abschnitt „Programmierhinweise“ auf Seite 389). Um Fehler abzufangen, müssen Sie in Ihrem Programm geeignete Fehlerausgänge vorsehen.

Anweisungen zur Dateibearbeitung

In einem DRIVE-Programm verwenden Sie folgende Anweisungen, wenn Sie Dateien bearbeiten. (Syntax und Regeln zu den einzelnen Anweisungen finden Sie im „DRIVE-Lexikon“ [3].)

– DECLARE FILE (Datei definieren)

  Die DECLARE FILE-Anweisung muß im Deklarationsteil des DRIVE-Programms stehen.

– OPEN FILE (Datei öffnen)

  Voraussetzung für die Bearbeitung einer Datei ist, daß die Datei geöffnet ist. Die OPEN FILE-Anweisung ist die erste Anweisung zur Dateibearbeitung im Verarbeitungsteil des DRIVE-Programms.

– WRITE FILE (in Datei schreiben)

– READ FILE (Datei lesen)

– GET FILE POSITION (Dateiposition lesen)

– SET FILE POSITION (in einer SAM-Datei positionieren)

– LOCATE FILE (in einer ISAM-Datei positionieren)

– DELETE FILE RECORD (Satz in ISAM-Datei löschen)

  Diese Anweisungen zum Bearbeiten von Daten und zum Positionieren in der Datei stehen nach der OPEN FILE-Anweisung im Verarbeitungsteil des DRIVE-Programms.

– CLOSE FILE (Datei schließen)

  Nachdem die Datei vollständig bearbeitet ist, muß sie geschlossen werden. Die CLOSE FILE-Anweisung ist die letzte Anweisung zur Dateibearbeitung im Verarbeitungsteil des DRIVE-Programms.

14.1 Datei definieren

Mit der Anweisung DECLARE FILE definieren Sie eine Datei. D.h. Sie geben DRIVE/WINDOWS den logischen Dateinamen bekannt, unter dem die Datei im Programm angesprochen wird.

Gleichzeitig geben Sie mit dieser Anweisung das Zeichen für die NULL-Wertdarstellung bekannt. Soll DRIVE/WINDOWS einen NULL-Wert in eine Datei schreiben, so geht dies nur, wenn ein Zeichen für die NULL-Wertdarstellung festgelegt wurde. DRIVE/WINDOWS

schreibt dieses Zeichen so oft in die Datei, bis das Feld, das den NULL-Wert hat, gefüllt ist, und erwartet dieses Zeichen in der Datei, wenn ein NULL-Wert vorliegt. Die Anweisung DECLARE FILE muß im Deklarationsteil des DRIVE-Programms stehen.

14.2 Dateiaufbau beschreiben

DRIVE/WINDOWS verlangt bei der Bearbeitung von Dateien keinen festen Dateiaufbau. Um mit DRIVE/WINDOWS auf Datensätze oder einzelne Datenfelder in den Datensätzen zugreifen zu können, müssen Sie wissen, aus welchen Daten eine Datei aufgebaut ist. Wieviel Platz DRIVE-Variablen in Dateien belegen, finden Sie in der Tabelle am Ende dieses Abschnitts.

Sie beschreiben den Aufbau von Datensätzen, indem Sie mit der Anweisung DECLARE VARIABLE entweder eine strukturierte Variable definieren oder mehrere Variablen, d.h. eine Variablenliste. Wenn Sie mit einer strukturierten Variablen arbeiten, beschreibt die gesamte Variable einen Datensatz und die einfachen Komponenten die Datenfelder. Wenn Sie mit einer Liste von Variablen arbeiten, beschreiben alle Variablen zusammen einen Datensatz und jede einzelne Variable ein Datenfeld.

Die mit der Anweisung DECLARE VARIABLE definierte(n) Variable(n) geben Sie bei lesenden Dateizugriffen in der INTO-Klausel der READ FILE-Anweisung an und bei schreibenden Dateizugriffen in der WITH-Klausel der WRITE FILE-Anweisung. Dabei können Sie für jede READ FILE- oder WRITE FILE-Anweisung eine andere Variable(-nliste) angeben. Eine bestehende Datei bestimmt die Reihenfolge der mit READ FILE zu lesenden Variablen. Die Reihenfolge der mit WRITE FILE geschriebenen Variablen bestimmt den Aufbau einer zu erstellenden Datei.

Durch die Definition unterschiedlicher Variablen lassen sich verschiedene Satzarten in einer Datei bearbeiten.

Die Anweisung DECLARE VARIABLE muß im Deklarationsteil des DRIVE-Programms stehen.

Die folgende Tabelle zeigt, wieviel Byte DRIVE-Variablen mit unterschiedlichen Datentypen in einer Datei belegen:

DRIVE-Datentyp	**Länge (in Byte)**
CHARACTER (n)	n
VARCHAR (n)	n + 3
NUMERIC (n,m)	n
DECIMAL (n,m)	n/2, aufgerundet auf die nächste Ganzzahl

DRIVE-Datentyp	Länge (in Byte)
EXTENDED DECIMAL, XDEC	7 + n/2, aufgerundet auf die nächste Ganzzahl
INTEGER	4
SMALLINT	2
REAL	4
DOUBLE PRECISION, FLOAT	8
DATE	10
TIME	8
TIME(3)	12
TIMESTAMP(3)	23
INTERVAL	23

Beispiel

In die Datei „dat1“ werden Datensätze geschrieben, deren Aufbau durch die DRIVE-Variable &file_record festgelegt ist.

```
DECLARE VARIABLE 1 &file_record,
                   2 pers_nr     INT,
                   2 nachname    CHAR (30),
                   2 vorname     CHAR (30),
                   2 plz         NUM  (5),
                   2 ort         CHAR (20),
                   2 strasse     CHAR (20),
                   2 hausnr      NUM (3);
...
WRITE FILE dat1 WITH &file_record;
```

Sie erreichen das gleiche Ergebnis, wenn Sie den Aufbau der Datensätze mit einer Liste von Variablen festlegen.

```
DECLARE VARIABLE &pers_nr     INT,
                 &nachname    CHAR (30),
                 &vorname     CHAR (30),
                 &plz         NUM  (5),
                 &ort         CHAR (20),
                 &strasse     CHAR (20),
                 &hausnr      NUM (3);
```

```
...
WRITE FILE dat1 WITH &pers_nr, &nachname, &vorname, &plz, &ort,
                     &strasse, &hausnr;
```

14.3 Datei öffnen und schließen

Mit der Anweisung OPEN FILE öffnen Sie eine Datei und verbinden diese Datei mit einem logischen Namen, unter dem sie im DRIVE-Programm mit DECLARE FILE definiert und angesprochen wird.

DRIVE/WINDOWS erlaubt es, dieselbe BS2000-Datei unter verschiedenen logischen Namen zu öffnen. Beachten Sie, daß auch das Betriebssystem einen mehrfachen Zugriff auf eine Datei erlauben muß.

Eine Datei kann nur bearbeitet werden, wenn sie geöffnet ist.

Mit der Anweisung OPEN FILE geben Sie gleichzeitig die Dateiart an und in welcher Eröffnungsart (OPEN-Modus) die Datei geöffnet werden soll.

Es gibt folgende Eröffnungsarten:

INPUT	Öffnen zum Lesen. Die Datei muß bereits vorhanden sein.
OUTPUT	Öffnen zum Schreiben. Ist die Datei vorhanden, wird der alte Inhalt gelöscht. Ist die Datei nicht vorhanden, wird sie neu erstellt.
EXTEND	Öffnen zum Schreiben. Ist die Datei vorhanden, bleibt der alte Inhalt erhalten und die neuen Daten werden ans Ende der Datei angehängt. Ist die Datei nicht vorhanden, wird sie neu erstellt.
UPDATE	Öffnen zum Lesen und Schreiben. Ist die Datei vorhanden, bleibt der alte Inhalt erhalten und die neuen Daten werden ans Ende der Datei angehängt. Ist die Datei nicht vorhanden, wird sie neu erstellt.
INOUT	Öffnen zum Lesen und Schreiben. Die Datei muß bereits vorhanden sein. Der alte Inhalt bleibt erhalten und die neuen Daten werden ans Ende der Datei angehängt.
OUTIN	Öffnen zum Lesen und Schreiben. Ist die Datei vorhanden, wird der alte Inhalt gelöscht. Ist die Datei nicht vorhanden, wird sie neu erstellt.

Sie schützen Dateien vor ungewolltem Überschreiben, wenn Sie diese Dateien mit dem OPEN-Modus INPUT oder EXTEND öffnen.

Die Anweisung OPEN FILE ist die erste Anweisung zur Dateibearbeitung im Verarbeitungsteil des DRIVE-Programms.

Mit der Anweisung CLOSE FILE schließen Sie eine geöffnete Datei.

Diese Anweisung ist die letzte Anweisung zur Dateibearbeitung im Verarbeitungsteil des DRIVE-Programms. Eine Datei muß geschlossen werden, nachdem sie bearbeitet ist.

Falls Dateien bei Programmende oder Programmabbruch auf oberster Programmstufe (END PROCEDURE und BREAK PROCEDURE im Hauptprogramm) noch geöffnet sind, schließt DRIVE/WINDOWS sie automatisch.

14.4 Dateiposition ermitteln und festlegen

Mit der Anweisung SET FILE POSITION positionieren Sie in einer SAM-Datei an den Dateianfang, das Dateiende oder auf die Position, die Sie zuvor mit der Anweisung GET FILE POSITION ermittelt haben.

DRIVE/WINDOWS erhöht die Dateiposition nach Schreib- und Lesezugriffen automatisch. Mit der Anweisung GET FILE POSITION lesen Sie in einer SAM-Datei die aktuelle Dateiposition und übertragen diese Dateiposition in eine Variable. Die Variable muß groß genug sein, um die Dateiposition (260 Byte lang) aufnehmen zu können.

In ISAM-Dateien positionieren Sie mit der Anweisung LOCATE FILE mit Hilfe des ISAM-Schlüssels auf einen Datensatz.

14.5 Datei lesen und schreiben

Mit der Anweisung READ FILE lesen Sie an der aktuellen Dateiposition einen Datensatz und übertragen diesen Datensatz in eine Variable oder Variablenliste.

Voraussetzung dafür ist, daß die Datei zum Lesen geöffnet ist.

Nach dem Lesen weist die aktuelle Dateiposition auf den nächsten Satz.

Bei Erreichen des Dateiendes belegt DRIVE/WINDOWS die Systemvariable &ERROR mit dem Eintrag 'OK END'.

Nur direkt nach der Anweisung READ FILE können Sie das Dateiende über die Systemvariable &ERROR abfragen.

Wenn Sie auf Dateien zugreifen, die mit dem OPEN-Modus UPDATE, INOUT oder OUTIN geöffnet sind, darf die Anweisung READ FILE nicht unmittelbar auf die Anweisung WRITE FILE folgen. Zwischen der READ FILE- und der WRITE FILE-Anweisung muß eine Anweisung zur Positionierung (SET FILE POSITION) stehen.

Beim Lesen von Variablen mit dem Datentyp VARCHAR erwartet DRIVE/WINDOWS ein 2 Byte langes Längenfeld vor dem eigentlichen Satzinhalt. Der Inhalt nicht belegter Bytes in VARCHAR-Variablen ist undefiniert.

Mit der Anweisung WRITE FILE schreiben Sie einen Datensatz in eine Datei. Der Aufbau und der Inhalt des Datensatzes sind durch die WITH-Klausel definiert.

Voraussetzung dafür ist, daß die Datei zum Schreiben geöffnet ist.

In SAM-Dateien wird der Datensatz an die aktuelle Dateiposition geschrieben. Existiert schon ein Datensatz, der überschrieben werden soll, muß der neue Satz die gleiche Länge haben wie der zu überschreibende Satz.

In ISAM-Dateien wird der Datensatz an die Stelle geschrieben, die durch den Schlüssel festgelegt ist.

Nach dem Schreiben weist die aktuelle Dateiposition auf den nächsten Satz.

DRIVE/WINDOWS schreibt Variablen mit dem Datentyp VARCHAR folgendermaßen in eine Datei: vor die Maximallänge wird ein 2 Byte langes Längenfeld geschrieben, hinter die Maximallänge das Zeichen \0.

Überschreiben Sie die Inhalte von Textdateien nicht. Weil die tatsächlich abgespeicherte Länge von Variablen nicht bekannt ist, können überschreibende Daten länger oder kürzer sein als die zu überschreibenden. In beiden Fällen wird der Aufbau der Textdatei geändert und die Textdatei kann nicht mehr über die (sie beschreibende) DRIVE-Variable gelesen werden.

Sie schützen Dateien vor ungewolltem Überschreiben, wenn Sie diese Dateien mit dem OPEN-Modus INPUT (nur lesende Zugriffe sind erlaubt) oder EXTEND (die neuen Daten werden angehängt) öffnen.

Wenn Sie auf Dateien zugreifen, die mit dem OPEN-Modus UPDATE, INOUT oder OUTIN geöffnet sind, darf die Anweisung WRITE FILE nicht unmittelbar auf die Anweisung READ FILE folgen. Zwischen der WRITE FILE- und der READ FILE-Anweisung muß eine Anweisung zur Positionierung (oder SET FILE POSITION) stehen.

14.6 Besonderheiten bei ISAM-Dateien

Datensatz löschen

Mit der Anweisung DELETE FILE RECORD löschen Sie in einer ISAM-Datei einen Datensatz mit dem angegebenen ISAM-Schlüssel.

Gleichzeitige Bearbeitung einer Datei

ISAM-Dateien können sowohl im TIAM- als auch im UTM-Betrieb von mehreren Programmen gleichzeitig bearbeitet werden. Voraussetzungen dafür sind:

– In der OPEN FILE-Anweisung müssen Sie für den Dateinamen auf Betriebssystemebene einen Dateikettungsnamen angeben (siehe DRIVE-Lexikon [3], Anweisung OPEN FILE).

– Im BS2000 müssen Sie den Dateikettungsnamen bekanntgeben und für die Datei einen mehrfachen Zugriff erlauben. Sie müssen dazu folgendes BS2000-Kommando eingeben:

```
/SET-FILE-LINK ... LINK-NAME=...,SUPPORT=...(SHARED-UPDATE=YES)
```

  – im TIAM-Betrieb vor dem Start des DRIVE-Programms oder im DRIVE-Programm mit der SYSTEM-Anweisung. Die Anweisung SYSTEM muß vor der OPEN-FILE-Anweisung stehen.

  – im UTM-Betrieb vor dem Start der DRIVE-Anwendung.

ISAM-Schlüssel

Der ISAM-Schlüssel gehört zu den Satzdaten und wird vom DRIVE-Programm bearbeitet. Sie können ein beliebiges Datenfeld als ISAM-Schlüssel definieren. Dazu müssen Sie nur Position und Länge des Feldes, das den ISAM-Schlüssel enthält, im BS2000 mit dem Kommando SET-FILE-LINK ... KEY-LENGTH=..., KEY-POSITION=... bekanntgeben, und zwar

– im TIAM-Betrieb vor dem Start des DRIVE-Programms oder im DRIVE-Programm mit der SYSTEM-Anweisung. Die Anweisung SYSTEM muß vor der OPEN-FILE-Anweisung stehen.
– im UTM-Betrieb vor dem Start der DRIVE-Anwendung.

Dieses Vorgehen ist geeignet, wenn Sie eine Tabelle aus einer Datenbank in eine ISAM-Datei kopieren. Sie können ein Feld mit einem eindeutigen Wert als ISAM-Schlüssel definieren.

Beachten Sie, daß Sie nur Datenfelder als ISAM-Schlüssel definieren, deren Werte nicht doppelt vorhanden sind.

14.7 Fehlerbehandlung

Über das Ergebnis jeder DRIVE-Anweisung, die auf eine Datei zugreift, gibt DRIVE/WINDOWS eine nach Fehlerklassen differenzierte Rückmeldung in die Systemvariable &ERROR_STATE aus.

Der Fehlermeldungsschlüssel des Dateiverwaltungssystems (DVS) wird in der Systemvariablen &SUB_CODE ausgegeben.

Bei Erreichen des Dateiendes erhält die Systemvariable &ERROR nach der READ FILE-Anweisung den Inhalt 'OK END'. Dieser Eintrag kann nur direkt nach der Anweisung READ FILE abgefragt werden, weil für die nächste DRIVE-Anweisung ein neuer Eintrag in die Systemvariable &ERROR erfolgt.

Sichern Sie den Inhalt von &ERROR direkt nach der READ FILE-Anweisung in einer Variablen. Diese Variable können Sie dann beliebig ausgeben lassen.

Beispiel

Wird nach einer READ FILE-Anweisung das Dateiende erreicht, gibt DRIVE/WINDOWS eine Meldung mit dem Inhalt von &ERROR aus.

```
DECLARE VARIABLE &error_   &CHAR (16);
...
READ FILE dat1 INTO &satz;
SET &error_ = &ERROR;
IF &ERROR='OK END'
   THEN SEND MESSAGE &error_;
END IF;
```

Um Programmabbrüche zu verhindern, können Sie mit der Anweisung WHENEVER Fehlerausgänge definieren (siehe „DRIVE-Lexikon“ [3], Anweisung WHENEVER).

14.8 Beispiele

Die folgenden Beispiele dienen zur Einführung in die Funktionen der Dateiverarbeitung.

Beispiel 1

Eine Cursortabelle wird in eine Datei kopiert.

```
OPTION DBSYSTEM=SESAM;
PROCEDURE kopieren;
DECLARE c1 CURSOR FOR SELECT * FROM firma;        /* Cursor deklarieren */
DECLARE FILE dfirma NULL '%';                     /* Datei definieren */
```

```
DECLARE VARIABLE &vfirma LIKE CURSOR c1;          /* Variable definieren, die */
                                                  /* die Datenfelder aufnimmt */
OPEN FILE dfirma IN 'kopie.dat' SAM OUTPUT;       /* Datei „dfirma“ oeffnen */
CYCLE c1 INTO &vfirma.*;                          /* Verarbeitungsschleife */
   WRITE FILE dfirma WITH &vfirma;                /* Datensatz in Datei */
                                                  /* schreiben */
END CYCLE;
COMMIT WORK;
CLOSE FILE dfirma;                                /* Datei schliessen */
END PROCEDURE;
```

Beispiel 2

Ein Datenfeld in einer Datei wird geändert. Das Dateiende kann nur nach der Anweisung READ FILE über die Systemvariable &ERROR abgefragt werden.

```
PROCEDURE dat_bearb;

DECLARE VARIABLE &dat_struktur,                   /* Variable definieren, für */
                 2 pers_nr     INT,               /* die Datenfelder */
                 2 nachname    CHAR (30),
                 2 vorname     CHAR (30),
                 2 abt         CHAR (10),
                 2 gehalt      NUM  (7,2);
DECLARE VARIABLE &dat_position CHAR (260);        /* Variable für die */
                                                  /* Dateiposition */
DECLARE FILE dat1 ;                               /* Datei definieren */

OPEN FILE dat1 IN 'mitarb.dat' INOUT;             /* Datei „dat1“ oeffnen */
SET FILE POSITION dat1 TO BEGIN;
GET FILE POSITON dat1 TO &dat_position;           /* Dateiposition lesen */
READ FILE dat1 INTO &dat_struktur;                /* Datensatz lesen */
IF &ERROR='OK END'
   THEN SEND MESSAGE 'Datei ist leer.';
END IF;
CYCLE WHILE &ERROR <> 'OK END';                   /* Verarbeitungsschleife */
   SET &gehalt=&gehalt * 1.055;
   SET FILE POSITON dat1 TO &dat_position;        /* Datei positionieren */
   WRITE FILE dat1 WITH &dat_struktur;            /* Datensatz schreiben */
   GET FILE POSITON dat1 TO &dat_position;        /* Dateiposition lesen */
   READ FILE dat1 INTO &dat_struktur;             /* Datensatz lesen */
END CYCLE;
CLOSE FILE dat1;                                  /* Datei schliessen */
END PROCEDURE;
```

Beispiel 3

Zwei Satzarten aus einer Datei werden verarbeitet. Beachten Sie, daß das folgende Programm für den Einsatz auf einem BS2000-Rechner optimal angepaßt ist und nur hier fehlerlos abläuft. Der Grund dafür ist die unterschiedliche Bearbeitung von Dateien: im BS2000 erfolgt sie satzweise, in SINIX und MS-Windows zeichenweise.

```
PROCEDURE saetze_bearb;

DECLARE VARIABLE &satz,
                 2 &satzart  CHAR (1),            /* Variable für Satzart */
                 2 &dummy    CHAR (200);
DECLARE VARIABLE &projekt REDEFINES &satz,        /* Variable für Satz „p“ */
                 2 satzart   CHAR (1),
                 2 titel     CHAR (8),
                 2 kosten    NUM  (10,2);
DECLARE VARIABLE &mitarbeiter REDEFINES &satz,    /* Variable für Satz „m“ */
                 2 satzart   CHAR (1),
                 2 nachname  CHAR (30),
                 2 vorname   CHAR (30),
                 2 abt       CHAR (10),
                 2 gehalt    NUM  (7,2);
DECLARE FILE dat1 ;                               /* Datei definieren */

OPEN FILE dat1 IN 'dat.link' INPUT;               /* Datei oeffnen */
READ FILE dat1 INTO &satz;                        /* Datensatzart lesen */
IF &ERROR='OK END'
   THEN SEND MESSAGE 'Datei ist leer.';
END IF;
CYCLE WHILE &ERROR <> 'OK END';                   /* Verarbeitungsschleife */
   CASE &satz.satzart;                            /* Satzart und */
      OF 'p'                                      /* Verarbeitung bestimmen */
         CALL e_projekt USING &projekt;
      OF 'm'
         CALL e_mitarbeiter USING &mitarbeiter;
   END CASE;
   READ FILE dat1 INTO &satz;
END CYCLE;
CLOSE FILE dat1;                                  /* Datei schliessen */
END PROCEDURE;
```

Beispiel 4

Beispiel 4 zeigt Ihnen ein Programm mit gleicher Aufgabenstellung wie Beispiel 3 für den plattformabhängigen Einsatz. Zwei Satzarten aus einer Datei werden verarbeitet.

```
PROCEDURE saetze_bearb;

DECLARE VARIABLE &satzart;                          /* Variable für Satzart */
DECLARE VARIABLE &projekt,                          /* Variable für Satz „p“ */
                 2 titel     CHAR (8),
                 2 kosten    NUM  (10,2);
DECLARE VARIABLE &mitarbeiter,                      /* Variable für Satz „m“ */
                 2 nachname  CHAR (30),
                 2 vorname   CHAR (30),
                 2 abt       CHAR (10),
                 2 gehalt    NUM  (7,2);
DECLARE FILE dat1 ;                                 /* Datei definieren */

OPEN FILE dat1 IN 'dat.link' INPUT;                 /* Datei oeffnen */
READ FILE dat1 INTO &satzart;                       /* Datensatzart lesen */
IF &ERROR='OK END'
   THEN SEND MESSAGE 'Datei ist leer.';
END IF;
CYCLE WHILE &ERROR <> 'OK END';                     /* Verarbeitungsschleife */
   CASE &satzart;                                   /* Satzart und */
      OF 'p'                                        /* Verarbeitung bestimmen */
         READ FILE dat1 INTO &projekt;
         CALL e_projekt USING &projekt;
      OF 'm'
         READ FILE dat1 INTO &mitarbeiter;
         CALL e_mitarbeiter USING &mitarbeiter;
   END CASE;
   READ FILE dat1 INTO &satzart;
END CYCLE;
CLOSE FILE dat1;                                    /* Datei schliessen */
END PROCEDURE;
```

14.9 Programmierhinweise

Die Bearbeitung von Dateien unterliegt nicht der Transaktionssicherung. Wenn Sie Transaktionen zurücksetzen, werden Änderungen, die Dateien betreffen, nicht zurückgesetzt. Die folgenden Beispiele sollen dies verdeutlichen.

Beispiele

```
COMMIT WORK;
OPEN FILE dat1 ...
WRITE FILE ...
ROLLBACK WORK;
CLOSE FILE ...
```

Nach dem Rücksetzen der Transaktion ist die Datei „dat1“ weiterhin geöffnet und der geschriebene Datensatz steht in der Datei. Beim Wiederanlauf führt das erneute Öffnen der Datei zu einem Fehler.

```
OPEN FILE dat1 ...
COMMIT WORK;
WRITE FILE ...
ROLLBACK WORK;
CLOSE FILE ...
```

Nach dem Rücksetzen der Transaktion, steht der innerhalb der Transaktion geschriebene Datensatz in der Datei. Er wird nicht gelöscht.

```
OPEN FILE dat1 ...
COMMIT WORK;
WRITE FILE ...
CLOSE FILE ...
ROLLBACK WORK;
```

Nach dem Rücksetzen der Transaktion, ist die Datei „dat1“ geschlossen. Beim Wiederanlauf führt das erneute Schreiben eines Datensatzes zu einem Fehler.

14.10 Dateibearbeitung Oldstyle und Newstyle

Wesentliche Unterschiede zwischen der Dateibearbeitung mit DRIVE/WINDOWS in Old-Style (DRIVE bis V5.1) und New-Style (DRIVE V6.0 und DRIVE/WINDOWS) sind:

– Im New-Style können Sie Textdateien bearbeiten.

– Im New-Style sind Sie freizügig beim Öffnen und Schließen von Dateien. Dies geschieht durch das DRIVE-Programm. Dateien müssen nicht wie im Old-Style mit der Anweisung ACQUIRE angefordert werden.

– Im New-Style arbeiten Sie direkt mit Dateien und nicht mehr mit dem Hilfsmittel des DMS-Views. Deshalb gibt es im New-Style keine SQL-ähnlichen Sprachmittel. Außerdem ist die Dateibearbeitung nicht transaktionsgesichert.

– Im New-Style erstellte Programme zur Dateibearbeitung sind auf anderen Plattformen ablauffähig.

15 Integration von Old-Style-Prozeduren

Für den reibungslosen Einsatz der DRIVE/WINDOWS V2.1 für Anwendungen mit früheren DRIVE-Versionen wurde der DRIVE-Mischbetrieb bisher über die DO-Anweisung unterstützt.

Zur besseren Integration bestehender Old-Style-Anwendungen in New-Style-Umgebung können Old-Style-Prozeduren jetzt auch mit CALL aufgerufen werden.

Sie können bestehende Old-Style-Anwendungen in New-Style-Umgebung ablaufen lassen und so

– Old-Style-Anwendungen mit der neuen DRIVE-Funktionalität ausstatten, also z.B. neuen grafischen Oberflächen und Client-Server-Architektur.

– New-Style-Anwendungen Zugriff auf LEASY, DMS und SESAM-CALL-DML ermöglichen.

Old-Style-Prozeduren, die im UTM-Asynchronbetrieb ablauffähig sind, können in einer New-Style-Serverumgebung mit CALL aufgerufen werden. So können z.B. Verteilte Anwendungen Old-Style-Prozeduren im Server mit CALL aufrufen, SESAM-, LEASY- oder DMS-Daten als Parameter zurückgeben und diese im Client mit grafischer Benutzeroberfläche präsentieren und weiterverarbeiten.

15.1 Betriebsarten und Konfigurationen

Die Integration von Old-Style-Prozeduren in New-Style-Anwendungen wird für folgende Betriebsarten unterstützt:

– Dialogbetrieb in TIAM

– Dialogbetrieb in UTM

– Serverbetrieb in UTM

Im Mischbetrieb von DRIVE/WINDOWS V1.1 ist für Old-Style eine SESAM-Fassung obligatorisch. Diese enthält implizit Zugriffsmöglichkeiten auf DMS (siehe „DRIVE-Programmiersystem“ [1], Kapitel Mischbetrieb).

Der Mischbetrieb von DRIVE/WINDOWS V2.1 unterstützt die Konfigurationen SESAM, LEASY und reines DMS bei DO- und CALL-Aufruf (siehe DRIVE-Programmiersystem [1], Kapitel DRIVE/WINDOWS für den Mischbetrieb generieren und „DRIVE-Lexikon“ [3], Anweisung PARAMETER STATIC OLDSTYLE).

Old-Style-Asynchron-Anwendungen sind serverfähig, da sie

– keine Bildschirm-Ein-/Ausgabe haben

– mit END, BREAK PROCEDURE, STOP oder END TRANSACTION beendet werden.

Bibliothekseinstellung

DRIVE/WINDOWS schaltet bei CALL und DO auf Old-Style-Prozeduren automatisch von New-Style- nach Old-Style-Umgebung um. Für die Bibliothekseinstellung gibt es die folgenden zwei Möglichkeiten:

– Sie geben den Bibliotheksnamen im Programmaufruf an.

```
CALL bibliothek(elemname)

DO bibliothek(elemname)
```

oder

– Sie haben die PLAM-Bibliothek, in der sich die Old-Style-Prozedur befindet, mit der Anweisung PARAMETER DYNAMIC LIBRARY in New-Style-Umgebung vor dem Programmaufruf bereits zugewiesen.

```
PARAMETER DYNAMIC LIBRARY=bibliothek
```

Beim impliziten Umschalten wird diese DRIVE-Bibliothek aus der New-Style-Umgebung übernommen und zur aktuellen DRIVE-Bibliothek in der Old-Style-Umgebung. Diese DRIVE-Bibliothek kann dann in der aktuellen DRIVE-Sitzung im Old-Style nicht mehr geändert werden. Dies gilt auch, wenn zwischendurch in den New-Style-Betrieb gewechselt und dort mit anderen DRIVE-Bibliotheken gearbeitet wird. Zur Übergabe weiterer Einstellungen mit der Anweisung PARAMETER siehe „DRIVE-Programmiersystem“ [1], Kapitel Mischbetrieb.

Es gibt keine zusätzlichen Restriktionen bei der Einbindung fremdsprachiger Unterprogramme (z.B. COBOL, siehe Kapitel „Programmierlogik“ auf Seite 81).

Datenbankkonfigurationen

Sie legen die gewünschte Old-Style-Variante für die Old-Style-Prozedur mit der Anweisung PARAMETER STATIC OLDSTYLE fest. Als Vorbelegung gilt die im New-Style generierte SESAM-Variante (d.h. im TIAM-Betrieb die geladene SESAM-Variante, im UTM-Betrieb die SESAM-Variante, die Sie bei der UTM-Generierung angegeben haben). Für SESAM-Zu-

griff im New-Style-Programm und in der Old-Style-Prozedur müssen die gleichen SESAM-Varianten in beiden Umgebungen gültig sein, also entweder SESAM V1 oder SESAM V2 für New-Style-Programm und Old-Style-Prozedur.

Sie können in einer Old-Style-Prozedur auf das Datenbanksystem SESAM V2 zugreifen, soweit die Sprachelemente von der CALL-DML-Schnittstelle unterstützt werden. Voraussetzung ist, daß in der Server-Konfigurationsdatei über die DBH-Startanweisung ADD-OLD-TABLE-CATALOG-LIST alle benötigten CALL-DML-Tabellen bekannt gemacht werden. Besonderheiten und Einschränkungen für CALL-DML-Tabellen sind im „DRIVE-SQL-Lexikon“ [5] beschrieben, für detaillierte Informationen siehe SESAM-Handbuch „CALL-DML-Anwendungen“ [23].

Im TIAM-Betrieb werden im Mischbetrieb von SESAM für den Old- und New-Style zwei verschiedene Benutzer angelegt. D.h. die SQL- und die CALL-DML-Anweisungen laufen in unterschiedlichen Transaktionen. Ein Wechsel zwischen Old- und New-Style darf deshalb nur außerhalb von Transaktionen erfolgen. DRIVE/WINDOWS V2.1 prüft bei einem CALL-Aufruf einer Old-Style-Prozedur, ob eine SQL-Transaktion offen ist. Ist dies der Fall, wird der CALL-Aufruf nicht durchgeführt und das rufende Programm abgebrochen.

Im UTM-Betrieb wird zwar von SESAM für den Old- und New-Style nur ein Bentuzer angelegt, trotzdem müssen die New-Style-SQL- und die Old-Style-CALL-DML-Anweisungen in unterschiedlichen Transaktionen ablaufen. DRIVE/WINDOWS führt daher die oben genannte Prüfung auch im UTM-Betrieb durch.

15.2 Programmierregeln

– Die Old-Style-Prozedur kann mit CALL aus einer beliebigen New-Style-Programmstufe in einer Aufrufhierarchie aufgerufen werden. Der Rücksprung aus der Old-Style-Prozedur erfolgt bei END PROCEDURE. Es können Parameter aus der Old-Style-Prozedur übergeben werden.
  In der Old-Style-Prozedur sind mehrere Folge-DO-Aufrufe auf weitere Old-Style-Prozeduren möglich sowie Parameterübergaben zwischen den Aufrufen. Die letzte Old-Style-Prozedur übergibt die RETURN-Parameter an die New-Style-Anwendung. CALL-Aufrufe in Old-Style-Prozeduren werden nicht unterstützt.

– Ein Wechsel zwischen Old-Style und New-Style ist wie bisher nur bei geschlossener Transaktion möglich, da Old-Style- und New-Style-Transaktionen (CALL-DML- und SQL-Transaktionen) nicht mischbar sind.

– Von einer Old-Style-Prozedur ist kein Folge-DO auf ein New-Style-Programm erlaubt, denn für den New-Style ist der Old-Style eine CALL-Stufe, und im New-Style ist auf ein mit CALL gerufenes Programm kein Folge-DO möglich.

– Die Anzahl der USING-Parameter muß ≤128 sein, die Länge aller Parameter muß ≤ 3072 Bytes sein.

– Im Client-Server-Betrieb erfolgt bei Transaktionsende Rücksprung in das rufende New-Style-Programm und von dort Rücksprung zum Client

Beispiel 1

Im folgenden Beispiel ruft ein New-Style-Programm mit CALL eine Old-Style-Prozedur auf, das wiederum eine weitere Old-Style-Prozedur mit DO aufruft.

```
PROC newstyle;

...
CALL oldstyle_1;              (1)
   PROC /* oldstyle_1 */;     (2, 7)
   ...
   END PROC;                  (3, 8)
CALL oldstyle_2;              (4)


   PROC /* oldstyle_2 */;     (5)
   DO oldstyle_1;             (6)
   END PROC;


END PROC;
```

(1) In einem New-Style-Programm wird eine CALL-Anweisung durchlaufen. DRIVE/WINDOWS unterbricht das laufende Programm und erkennt intern, daß die gerufene Prozedur oldstyle_1 eine Old-Style-Prozedur ist.

(2) DRIVE/WINDOWS wechselt intern in die Old-Style-Umgebung und führt dort die Prozedur oldstyle_1 aus.

(3) Die Prozedur oldstyle_1 wird bei END PROCEDURE beendet. DRIVE/WINDOWS wechselt intern in die New-Style-Umgebung zurück und setzt das rufende Programm newstyle mit der Anweisung fort, die auf CALL (1) folgt.

(4) Das Programm newstyle führt erneut eine CALL-Anweisung auf eine Old-Style-Prozedur aus.

(5) DRIVE/WINDOWS wechselt intern in die Old-Style-Umgebung und führt dort die Prozedur oldstyle_2 aus.

(6) In der Prozedur oldstyle_2 wird eine Folge-DO-Anweisung auf die Old-Style-Prozedur oldstyle_1 durchlaufen. DRIVE/WINDOWS bricht oldstyle_2 ab.

(7) Die Folge-Prozedur oldstyle_1 wird von DRIVE/WINDOWS in Old-Style-Umgebung ausgeführt.

(8) Die Prozedur oldstyle_1 wird bei END PROCEDURE beendet. DRIVE/WINDOWS wechselt intern in New-Style-Umgebung zurück und setzt das rufende Programm newstyle mit der Anweisung fort, die auf CALL (4) folgt.

15.3 Parameterübergabe

Als Übergabeparameter können Sie alle Datentypen wählen, die Old-Style und New-Style gemeinsam bekannt sind:

INTEGER, SMALLINT, NUMERIC, DECIMAL, CHARACTER.

Es sind also z.B. keine strukturierten Variablen als Parameter möglich, d.h. ein Datensatz muß in seinen einzelnen Datenfeldern übergeben werden (siehe Beispiel Seite 403). Sie müssen mengen- und typverträglich sein. Beim Hinübertragen muß die Anzahl der Parameter beim CALL im New-Style-Programm größer oder gleich sein der Anzahl der Parameter in der Old-Style-Procedure-Leiste. Beim Rückübertragen muß die Anzahl der Parameter in der Old-Style-Procedure-Leiste größer oder gleich sein der Anzahl der RETURN-Parameter beim CALL im New-Style-Programm. Es werden nur die RETURN-Parameter beim CALL des New-Style-Programms versorgt.

In der Old-Style-Prozedur ist ein Folge-DO mit Parameterübergabe möglich , so daß Old-Style-Ketten bestehen bleiben können. Die letzte Old-Style-Prozedur übergibt die RETURN-Parameter an die New-Style-Anwendung.

Parameterwerte hinübertagen von New- nach Old-Style

Parameterwerte werden von New-Style-Programmen an Old-Style-Prozeduren übergeben gemäß der Folge-DO-Technik des DRIVE-Mischbetriebs in der DRIVE/WINDOWS V1.1.

In der USING-Klausel dürfen Ausdrücke angegeben werden. Das Ergebnis eines numerischen Ausdrucks wird vor der Übergabe in den Old-Style-Betrieb in ein numerisches Literal vom Typ NUMERIC, also numerisch ungepackt, konvertiert.

Beispiel 2

Die Parameter a, b und c werden vom New-Style- an die Old-Style-Prozedur übergeben. Die Old-Style-Prozedur gibt b und c an das New-Style-Programm zurück.

```
PROC newstyle;
DCL VAR
   &a INTEGER,
   &b CHAR(10),
   &c NUM(5,2);
...
CALL oldstyle USING &a, RETURN &b, RETURN &c;   (1, 4)
...
END PROC;


   PROC /* oldstyle */ USING &x, &y, &z;                    (2)
   CRE VAR
      &x INTEGER,
      &y CHAR(10),
      &z NUM(5,2);
   ...
   END PROC;                                                (3)
```

(1) Aufruf der Old-Style-Prozedur mit Übergabe der Parameterwerte von &a, &b und &c.

(2) Bevor die Old-Style-Prozedur ausgeführt wird, werden die Parameter-Variablen &x, &y und &z mit Werten versorgt.

(3) Rücksprung ins New-Style-Programm bei fehlerfreiem Ende der Old-Style-Prozedur.

(4) Die Returnvariablen &b und &c werden mit den aktuellen Werten von &y und &z versorgt. Anschließend wird das New-Style-Programm mit der Anweisung hinter CALL fortgesetzt.

Parameterwerte rückübertragen von Old- nach New-Style

Beim fehlerfreien Rücksprung der Old-Style-Prozedur in das New-Style-Programm werden die aktuellen Parameterwerte übergeben, die in der Anweisung CALL oldstyle USING... mit RETURN angegeben wurden.

Beispiel 3

Ein Folge-DO in Old-Style-Prozeduren ist auch bei Parameter-Rückgabe an ein New-Style-Programm möglich.

```
PROC newstyle;
DCL VAR
   &a INTEGER,
   &b CHAR(10),
   &c NUM(5,2);
...
CALL oldstyle_1 USING &a, RETURN &b, RETURN &c; (1, 6)
...
END PROC;


   PROC /* oldstyle_1 */ USING &x, &y, &z;                (2)
   CRE VAR
      &x INTEGER,
      &y CHAR(10),
      &z (NUM(5,2);
   CRE VAR
      &u SMALLINT,
      &v CHAR(10),
      &w DEC(5,2);
   ...
   DO oldstyle_2 USING &u, &v, &w;              (3)
   ...
   END PROC;
```

```
PROC /* oldstyle_2 */ USING &p, &q, &r;        (4)
CRE VAR
   &p SMALLINT,
   &q CHAR(10),
   &r DEC(5,2);
...
END PROC;                                       (5)
```

(1) Aufruf der Prozedur oldstyle_1. Es werden die Parameterwerte von &a, &b und &c übergeben.

(2) Bevor die Prozedur oldstyle_1 ausgeführt wird, werden die Parameter-Variablen &x, &y und &z mit Werten von &a, &b, &c von newstyle versorgt.

Sowohl im CALL-Aufruf als auch in der PROCEDURE-Anweisung der Old-Style-Prozedur ist der zu übergebende Parameter in der USING-Klausel anzugeben. In der Old-Style-Prozedur muß zusätzlich eine Variablendefinition der zu übergebenden Parameter erfolgen.

(3) Ansprung der Prozedur oldstyle_2.

(4) Bevor die Prozedur oldstyle_2 ausgeführt wird, werden die Parameter-Variablen &p, &q und &r mit den aktuellen Werten von &u, &v und &w der Prozedur oldstyle_1 versorgt.

(5) Bei fehlerfreiem Ende der Prozedur oldstyle_2 Rücksprung ins New-Style-Programm.

(6) Die Returnvariablen &b und &c werden mit den aktuellen Werten von &q und &r versorgt. Anschließend wird das New-Style-Programm mit der Programm-Anweisung hinter CALL fortgesetzt.

15.4 Programmierempfehlungen

Es gibt zwei grundsätzliche Möglichkeiten der Integration von Old-Style-Prozeduren in New-Style-Programme, die beide geringe Eingriffe erfordern und eine sichere Parameterübergabe bieten:

1. Sie schalen die Old-Style-Anwendung in einen Old-Style-Rahmen ein, der die Parameter korrekt an das rufende New-Style-Programm übergibt.

Die letzte Old-Style-Prozedur in der Folge-DO-Kette (im Beispiel 4 oldstyle_2) ruft am Verarbeitungsende mit Folge-DO eine Old-Style-Dummy-Prozedur auf, die in der PROCEDURE-Anweisung eine USING-Klausel enthält, die verträglich ist mit der USING-Klausel in der CALL-Anweisung des New-Style-Programmms (im Beispiel 3 newstyle). Diese Dummy-Prozedur muß nur aus der PROCEDURE-Anweisung mit USING-Klausel und der END PROCEDURE-Anweisung bestehen.

Beispiel 4

```
PROC /* oldstyle_2 */ USING &p, &q, &r;

...

DO dummy USING &q, &r;

END PROC;


PROC /* dummy */ USING &s, &t;

END PROC;
```

Aus Sicht der letzten Old-Style-Prozedur in der Folge-DO-Kette (im Beispiel 4 oldstyle_2) ist der Aufruf der Dummy-Prozedur wie ein Versorgen von eigenen Ausgabeparametern. Die Werte, die dem New-Style-Programm zum Rücksprung übergeben werden, werden in einem DO-Aufruf übersichtlich gebündelt. Die USING-Klausel der letzten Folge-DO-Prozedur (oldstyle_2) vor der Dummy-Prozedur ist beliebig, sie kann sogar entfallen. Die korrekte Rückübertragung von USING-Parametern wird auf diese Weise unabhängig von der Folge-DO-Kette von Old-Style-Prozeduren und deren USING-Klauseln gewährleistet.

2. Sie versorgen die Parameter-Klausel einer vorhandenen zentralen Old-Style-Steuer-Prozedur vor END PROCEDURE mit entsprechenden Werten über SET-Anweisungen.

   In Old-Style-Anwendungen, deren Ende immer über die Anweisung END PROCEDURE einer Steuerprozedur läuft, können Sie SET-Anweisungen unmittelbar vor END PROCEDURE einfügen. So versorgen Sie die USING-Klausel mit Werten, falls am Prozedurende die Eingabeparameter keine aktuellen Werte haben und sonst eine korrekte Versorgung der Parameter vor Rücksprung in das New-Style-Programm nicht gewährleistet ist. Die gesamte Old-Style-Anwendung bleibt kompatibel zu einer reinen Old-Style-Umgebung. D.h., sie kann auch ohne CALL-Aufruf aus einem New-Style-Programm ablaufen, da die SET-Anweisungen vor END PROCEDURE keine Auswirkungen auf den Ablauf der restlichen Anwendung haben.

Beispiel 5

```
PROC newstyle;
...
CALL steuer USING &a, RETURN &b, RETURN &c;
...
END PROC;


PROC /* steuer */ USING &x, &y, &z;
CREATE VAR &X INTEGER;
CREATE VAR &y NUM(5,2);
CREATE VAR &z CHAR(10);
...
CREATE FORM ...;           /*  Abfragemaske für Auswahl der Verarbeitungsprogramme */
FILL FORM ....;
DISPLAY FORM ....;
IF ....                                          /* Auswahl für ein Verarbeitungsprogramm */
     THEN DO oldstyle_1 USING &X, &y, &z;    /* Aufruf des Verarbeitungsprogramms */
END PROC;
...
SET &y=&var1;
SET &z=&var2;
END PROC;
```

15.5 Beispiel für Client-Server-Anwendung

In der Old-Style-Prozedur wird auf eine SESAM-Datenbank über die CALL-DML-Schnittstelle zugegriffen. Die Datensätze werden als Parameter an das New-Style-Programm im BS2000-Server übergeben und im Client-Programm unter MS-Windows in einem Fenster ausgegeben.

15.5.1 Windows-Client für Datensatzausgabe

```
/* Verteilung einstellen */
PAR DISTRIBUTION ELEMENT=newstyle APPLICATION=integration;
/* integration = Symbolic Destination Name in Side-Information Datei (upicfile) */
```

Hauptprogramm im Client

```
PROCEDURE fenster USING &SINGLE_ENTRY_FIELD_Name, SINGLE_ENTRY_FIELD_Kasse,
                RETURN &P_SCHLUESSEL INTEGER, &P_NAME CHAR(20), &P_KASSE CHAR(10),
                &Hilfsvar CHAR(1);
DCL WINDOW "ABFRAGE";
/*  Hauptfenster zum Abfragen des Suchkriteriums fuer Datenbankabfrage */
DCL WINDOW "Datensatzausgabe";          /* Dialogbox  zum Ausgaben des Datensatzes */
...



/* Subprozedur fuer den Fall, dass kein Datensatz gefunden wurde */
SUBPROCEDURE kein_Treffer;
SEND MESSAGE 'Kein Datensatz vorhanden';
END SUBPROCEDURE;

/* Fehlerroutine fuer Window-Error */
SUBPROCEDURE windowerror;
.
END SUBPROC;

/* Programmbody */
WHENEVER &ERROR IN ('WINDOW ERROR') CALL windowerror;
ADD WINDOW "ABFRAGE";
END PROCEDURE;
```

Skripts im Client

```
/* Skript fuer Hauptfenster ABFRAGE */

SCRIPT "ABFRAGE";

/**********  INITIALISIERUNGSBLOCK  **********/

CLEAR &SINGLE_ENTRY_FIELD_NAME;

CLEAR &SINGLE_ENTRY_FIELD_KASSE;

/**********  EREIGNISGESTEUERTER ABLAUF  **********/

ON ACTIVATE "auswahlknopf1";                    /* Wahl des Suchkriteriums Name */

   MOVE DATA "SINGLE_ENTRY_FIELD_Name";

   CALL newstyle USING &SINGLE_ENTRY_FIELD_Name,
                RETURN &P_SCHLUESSEL, RETURN &P_NAME, RETURN &P_KASSE,
                RETURN &Hilfsvar;

   IF &Hilfsvar = "T"                           /* T fuer Datensatz gefunden */

      THEN SET &OUTPUT_FIELD_S = &P_SCHLUESSEL;

           SET &OUTPUT_FIELD_N = &P_NAME;

           SET &OUTPUT_FIELD_K = &P_KASSE;

           NEXT WINDOW "Datensatzausgabe";

           ELSE call kein_Treffer;

   ...


ON ACTIVATE "auswahlknopf2";                    /* Wahl des Suchkriteriums Kasse */

   MOVE DATA "SINGLE_ENTRY_FIELD_Kasse";

   CALL newstyle USING SINGLE_ENTRY_FIELD_Kasse,
                RETURN &P_SCHLUESSEL, RETURN &P_NAME, RETURN &P_KASSE,
                RETURN &Hilfsvar;

   IF &Hilfsvar = "T"

      THEN SET &OUTPUT_FIELD_S = &P_SCHLUESSEL;

           SET &OUTPUT_FIELD_N = &P_NAME;

           SET &OUTPUT_FIELD_K = &P_KASSE;

           NEXT WINDOW "Datensatzausgabe";

           ELSE call kein_Treffer;
```

```
   ...
END SCRIPT;


/* Skript fuer Dialogbox DATENSATZAUSGABE */
SCRIPT "Datensatzausgabe";
/**********  INITIALISIERUNGSBLOCK  **********/
/* da zwischen Old-Style-Prozeduren  und New-Style-Programmen keine */
/*  Wiederholungsgruppen als Parameter uebergeben werden koennen, werden die */
/*  einzelnen Datenfelder den einzelnen Ausgabefeldern zugewiesen */
SET OUTPUT_FIELD_SCHLUESSEL = &SCHLUESSEL;
SET OUTPUT_FIELD_NAME = &NAME;
SET OUTPUT_FIELD_KASSE = &KASSE;
/**********  EREIGNISGESTEUERTER ABLAUF  **********/
ON ACTIVATE "PUSH_BUTTON_END";
     CLOSE WINDOW;
END SCRIPT;
```

15.5.2 NEW-Style-Programm im BS2000

Als DRIVE-Startparameter in der Startprozedur für die UTM-Anwendung auf Server-Seite wird im New-Style die PLAM-Bibliothek zugewiesen, in der sich die Old-Style-Prozedur befindet:

```
PARAMETER DYNAMIC LIBRARY=oldstyle_bibl;


/* Programm newstyle */
PROCEDURE newstyle USING SINGLE_ENTRY_FIELD_Kasse, RETURN &SCHLUESSEL INTEGER,
RETURN &NAME CHAR(20), RETURN &KASSE CHAR(10), RETURN &Hilfsvar CHAR(1);
...
CALL oldstyle USING RETURN &SCHLUESSEL, RETURN &NAME, RETURN &KASSE, RETURN
&Hilfsvar;
...
END PROCEDURE;                          /* Sprung zum Client-Programm */
```

15.5.3 OLD-Style-Prozedur im BS2000

DRIVE-Startparameter für die Old-Style-Umgebung:

```
PARAMETER PLAMLIB = oldstyle_bibl;


/* Prozedur oldstyle */
PROCEDURE &P_SCHLUESSEL, &P_NAME, &P_KASSE, &Hilfsvar;
/* Uebergabeparameter deklarieren */
DROP VIEWS;
CRE VAR &P_SCHLUESSEL INTEGER;
CRE VAR &P_NAME CHAR(20);
CRE VAR &P_KASSE CHAR(10);
CRE VAR &Hilfsvar CHAR(1);                    /* fuer Abfrage, ob Datensatz vorhanden */
CREATE TEMPORARY VIEW PERS
     AS SELECT SCHLUESSEL, NAME, KASSE FROM FIRMA;
...
SELECT * FROM PERS WHERE KASSE = 'PRIVAT' WITHOUT LOCK;
IF &SQL_STATE='NOT EXISTING'
     THEN SET &Hilfsvar = "L";                        /* L fuer leer*/
          ELSE
             SET &Hilfsvar = "T";                     /* T fuer Treffer */
             /* Versorgen der Uebergabeparamter */
             SET P_SCHLUESSEL = SCHLUESSEL;
             SET P_NAME = NAME;
             SET P_KASSE = KASSE;
...
END PROCEDURE;
```

Zur Generierung von Verteilten Anwendungen und des Mischbetriebs siehe „DRIVE-Programmiersystem“ [1], Kapitel DRIVE/WINDOWS für den UTM-Betrieb generieren.

15.6 Old-Style-Prozedur als Server-Anwendung

Kompilierte Old-Style-Prozeduren (=Programme) können auch als Server-Anwendung eingesetzt werden.

Old-Style-Prozeduren, die in New-Style-Umgebung eingesetzt werden sollen, müssen mit OPTION PROGRAMTYPE=ASYNCHRON neu übersetzt werden. D.h. die Compileroption PARAMETER COMPILEROPTION, die in DRIVE V5.1 nur für UTM-Asynchron-Teilprogramme gilt, wird in DRIVE/WINDOWS V2.1 erweitert auf alle Old-Style-Prozeduren (auch Dialog-Prozeduren), die in New-Style-Umgebung ablaufen sollen.

Fachwörter

In diesem Fachwortverzeichnis sind wichtige Begriffe für DRIVE/WINDOWS plattformübergreifend aufgelistet.

Alpha-Anwendung
Eine Alpha-Anwendung realisiert Bildschirmein-/-ausgaben entweder in dynamischen DRIVE-Formaten oder in statischen FHS-/FORMANT-Formaten.

Backend
Das Backend von INFORMIX regelt den Zugriff eines DRIVE-Anwenders oder -Programms auf das Datenbanksystem INFORMIX-SE und OnLine. Das Backend muß gestartet sein, bevor Sie von DRIVE/WINDOWS mit INFORMIX arbeiten können. Das Backend wird angesprochen durch SQL-Anweisungen und Referenzen auf Datenbanken oder SQL-Objekte, die von ihm verwaltet werden.

Basistabelle
In einer relationalen Datenbank sind Anwenderdaten in Basistabellen organisiert, deren Aufbau der Anwender in SESAM V2 und INFORMIX mit DDL-Anweisungen (DATA DESCRIPTION LANGUAGE) festlegt und in SESAM V1 und UDS mit Dienstprogrammen.

Batch-Prozedur
BS2000-Kommandos, die in einer ENTER-Datei stehen, werden als Stapelauftrag ausgeführt.

Betrieb mit TP-Monitor
Mehrere Benutzer arbeiten von verschiedenen Bildschirmen aus unter Kontrolle des TP-Monitors gleichzeitig und transaktionsgesichert mit demselben Programm.

Betrieb ohne TP-Monitor
Benutzer arbeiten mit DRIVE-Programmen ohne TP-Monitor (UTM).

Bibliothek

in BS2000: PLAM-Bibliiothek mit Elementen verschiedenen Typs für Sourcen, COPY-Elemente, Zwischencodes, Objektcodes, Übersetzungslisten und Benutzerkennsätze.

in SINIX: Dateiverzeichnisstruktur mit Unterverzeichnissen für Sourcen, COPY-Elemente, Zwischencodes, Objektcodes, Übersetzungslisten und Benutzerkennsätze.

in MS-WINDOWS: Dateiverzeichnisstruktur mit Unterverzeichnissen für Sourcen, COPY-Elemente, Zwischencodes, Übersetzungslisten, Fehlerlisten und Debuggerlisten.

Binärdatei

Datei, deren Daten binär gespeichert sind.

Compiler-Betrieb

Im Compiler-Betrieb arbeiten Sie, sobald Sie Programme aufrufen, die mit dem DRIVE-Compiler DRIVE/WINDOWS-Comp übersetzt wurden.

Datensatz

Die Anwenderdaten, die die horizontalen Felder einer Basistabelle (Satzelemente oder Spalten) füllen.

Im BS2000 ist ein Datensatz diejenige Einheit, mit der Zugriffe auf Dateien erfolgen. Dies geschieht entweder sequentiell (SAM) oder index-sequentiell (ISAM). Ein Datensatz besteht aus Datenfeldern.

DBH

Der DBH (Data Base Handler) regelt den Zugriff eines DRIVE-Anwenders oder -Programms auf die Datenbanksysteme SESAM und UDS. Der DBH muß gestartet sein, bevor Sie von DRIVE/WINDOWS mit einer SESAM- oder UDS-Datenbank arbeiten können. Der DBH wird angesprochen durch SQL-Anweisungen und Referenzen auf Datenbanken oder SQL-Objekte, die von ihm verwaltet werden.

Debug-Lauf

Ein Programm läuft unter Kontrolle des Debuggers ab. Ein Debug-Lauf ist beendet, sobald das Ende des zu testenden Programms erreicht wird.

Debug-Modus

Ein Programm läuft unter Kontrolle des Debuggers ab. Sie können DRIVE-Programme testen und Fehler suchen.

Der Debug-Modus beginnt mit der Anweisung DEBUG oder menügeführt über den Befehl **Debug** oder **Programm/Debuggen**. Der Debug-Modus endet mit der Anweisung BREAK DEBUG oder sobald das letzte Debug-Fenster geschlossen wird.

Dialog-Modus

Sie geben eine DRIVE-Anweisung interaktiv am Bildschirm ein. Den Dialog-Modus gibt es im BS2000, in SINIX im Alpha-Betrieb sowie im Experten-Modus der SPU. Unter MS-Windows machen Sie Ihre Eingaben ausschließlich menügeführt.

Dialog-Programm

Die Anweisung DO startet ein DRIVE-Programm als Dialog-Programm. Mit einem Dialog-Programm kann der Anwender kommunizieren (Eingaben über Bildschirm oder Manipulation von Fenstern). Ein Dialog-Programm belegt den Bildschirm/das Fenster, in dem das Dialog-Programm gestartet wird.

DRIVE-Anweisung

Anweisung des Sprachumfangs von DRIVE/WINDOWS, ausgenommen sind SQL-Anweisungen. DRIVE-Anweisungen sind in den DRIVE-Lexika [3], [7], [14] beschrieben.

DRIVE-Programm

Ein Programm, in dem mehrere DRIVE-Anweisungen zusammengefaßt sind (PROCEDURE / END PROCEDURE).
Ein DRIVE-Programm kann als Dialog-Programm (mit der Anweisung DO) oder als UTM-Asynchronvorgang (mit der Anweisung ENTER) gestartet werden.
Die Anweisung ENTER startet einen eigenständigen UTM-Asynchronvorgang, der im Hintergrund abläuft.
Das bedeutet: Für weitere DRIVE-Anweisungen ist der Bildschirm nicht gesperrt. Mit dem Programm selbst kann der Anwender allerdings nicht kommunizieren (keine Eingaben über Bildschirm an das Programm).

dynamisches Bildschirmformat

Ein mit DECLARE FORM im DRIVE-Programm erstelltes Bildschirmformat. Das Format steht nur innerhalb des DRIVE-Programms zur Verfügung und kann beliebig oft ausgegeben werden.

dynamisches New-Style-Programm

Ein New-Style-Programm heißt dynamisch, wenn es keine statischen SQL-Anweisungen außer COMMIT WORK und ROLLBACK WORK enthält.

dynamisches Objekt

Ein (DRIVE- oder SQL-) Objekt heißt dynamisch definiert, wenn es im Rahmen einer EXECUTE-Anweisung definiert wurde, andernfalls heißt es statisch definiert.

dynamische SQL-Anweisung

Eine SQL-Anweisung heißt dynamisch, wenn sie erst im Zuge der Ausführung einer EXECUTE-Anweisung generiert, übersetzt und ausgeführt wird.

Experten-Modus

Den Experten-Modus gibt es nur in SINIX. Sie führen Ihren Dialog mit DRIVE anweisungsgesteuert. Sie kommen in den Experten-Modus beim Starten von DRIVE/WINDOWS im Alpha-Betrieb. Im Grafik-Betrieb wechseln Sie von der SPU in den Experten-Modus über den Menüeintrag „Experte“. Sie arbeiten mit DRIVE/WINDOWS im Dialog-Modus und können alle DRIVE-Anweisungen des Dialog-Modus eingeben.

Fensterobjekt

Die folgende Tabelle stellt die Bezeichnungen für Fensterobjekte von DRIVE/WINDOWS und Dialog Builder gegenüber.

Name des Fensterobjekts (Metavariable window_objekt)	**Name des Fensterobjekts in der Bibliothek des Dialog Builders**
eingabefeld	SINGLE ENTRY FIELD MULTI ENTRY FIELD
ausgabefeld_variabel	OUTPUT FIELD
ausgabefeld_konstant	LABEL
aktionsknopf	PUSH BUTTON
auswahlliste	CHOICE LIST
auswahlknopf	TOGGLE BUTTON
einfachauswahlknöpfe	RADIO FIELD
mehrfachauswahlknöpfe	CHECK FIELD
menü	MENU
Menüeintrag	MENU ITEM
bulletin_board	BULLETIN BOARD
kombobox	COMBO BOX
kaskadenknopf	CASCADE BUTTON
gruppe	SET

Grafik-Anwendung

Eine Grafik-Anwendung realisiert Bildschirmein-/-ausgaben in Fenstertechnik über Window-4GL-Anweisungen.

Interpreter-Betrieb

Im Interpreter-Betrieb arbeiten Sie immer, solange Sie nicht den DRIVE-Compiler DRIVE/WINDOWS-Comp einsetzen.

ISAM-Datei
Datei für index-sequentiellen Zugriff (Indexed Sequential Access Method). Jeder Datensatz besitzt einen Schlüsselbereich (= ISAM-Schlüssel).

Kompakt-Bildschirmformat
Ein Bildschirmformat, das im Verarbeitungsteil eines DRIVE-Programms mit DISPLAY FORM definiert und sofort ausgegeben wird. Das Format steht nur innerhalb dieser DISPLAY FORM-Anweisung zur Verfügung.

Kompakt-Listenformat
Ein Listenformat, das im Verarbeitungsteil eines DRIVE-Programms mit DISPLAY LIST definiert und sofort in eine Druckdatei ausgegeben wird. Das Format steht nur innerhalb dieser DISPLAY LIST-Anweisung zur Verfügung.

Listenformat
Ein mit DECLARE LIST im DRIVE-Programm erstelltes Listenformat. Das Format steht nur innerhalb des DRIVE-Programms zur Verfügung und kann beliebig oft in eine Druckdatei ausgegeben werden.

Meldungsdatei
Datei mit den Systemmeldungen von DRIVE/WINDOWS (BS2000). Über die Meldungsnummer können Sie auf DRIVE-Meldungen zugreifen.

Meldungskatalog
Datei mit Meldungen. Der Meldungskatalog wird aus einer Meldungstext-Quelldatei generiert wird.
Meldungskataloge bieten die Möglichkeit, Meldungen in verschiedenen Sprachen zu speichern und - abhängig von der jeweiligen Sprachumgebung - auszugeben.
Der Standard-Meldungskatalog enthält die Systemmeldungen von DRIVE/WINDOWS (SINIX, MS-Windows).
Der benutzerdefinierte Meldungkatalog enthält selbst erstellte Meldungen, auf die in DRIVE-Programmen zugegriffen werden kann.

Meldungstext-Quelldatei
Datei, aus der ein Meldungskatalog generiert wird. Die Meldungstext-Quelldatei ist editierbar und besitzt ein definiertes Format

NULL-Wert
Der NULL-Wert zeigt an, daß der Inhalt eines Datenbankfeldes undefiniert ist.

NULL-Wertzeichen
Abdruckbares Zeichen zur Darstellung des NULL-Wertes bei Ein- und Ausgaben.

Old-Style-Programm
Eine Programm, in dem mehrere DRIVE V5.1-Anweisungen zusammengefaßt sind (PROCEDURE / END PROCEDURE). Ein Old-Style-Programm ist compiliert.

Old-Style-Prozedur
Eine Programm, in dem mehrere DRIVE V5.1-Anweisungen zusammengefaßt sind (PROCEDURE / END PROCEDURE). Eine Old-Style-Prozedur ist nicht compiliert.

Objektcode
Übersetzungsergebnis des Compilers. Objektcode erhalten Sie, wenn Sie eine Source übersetzen mit Einsatz des DRIVE-Compilers DRIVE/WINDOWS-Comp.

permanentes Objekt
Ein permanentes (DRIVE- oder SQL-) Objekt wird mit der DRIVE-Klausel PERMANENT definiert, d.h. es bleibt mit seinem Wert über das DRIVE- (Unter-) Programmende hinaus erhalten. Permanent definiert werden können das temporäre SQL-Objekt Cursor (siehe temporäres Objekt) sowie die DRIVE-Objekte Variable, DRIVE-Bildschirm- und Listenformat (BS2000/SINIX).

Permanente SQL-Objekte hingegen sind in der Datenbank festgeschrieben und exisitieren nicht nur in anwendungsspezifischen Speicherbereichen. Der Begriff permanent bedeutet also bei DRIVE und SQL etwas anderes, z.B. sind permanente Cursor temporäre SQL-Objekte. Permanente SQL-Objekte werden daher zur deutlichen Unterscheidung persistent genannt (siehe persistentes Objekt).

persistentes Objekt
Ein persistentes Objekt ist ein SQL-Objekt, das in der Datenbank gespeichert wird und zu den Metadaten der Datenbank gehört. Persistente Objekte sind z.B. Basistabellen der Anwendungen, Systemtabellen des DB-Servers, Spalten, Constraints, Indizes, Views (ausgenommen temporäre) und Synonyme.

Programm-Modus
Ein Programm, in dem mehrere DRIVE-Anweisungen zusammengefaßt sind, läuft ab. Der Programm-Modus beginnt mit der Anweisung DO oder menügeführt über den Befehl **Ausführen**.

Remote-Zugriff
Zugriff auf die Datenbanken SESAM und UDS auf einem entfernten BS2000-Rechner. Die Netzkomponente DRIVE/WINDOWS-NET stellt dem Anwender die SQL-Anweisungen so zur Verfügung, als würde der Datenbankanschluß lokal ablaufen.

Report
Ein Report ist ein geräteunabhängiges Listenformat, das innerhalb des DRIVE-Programms mit DECLARE REPORT ... END REPORT definiert wird. Die Report-Definition steht nur innerhalb des DRIVE-Programms zur Verfügung und kann zur Erzeugung beliebig vieler Reports verwendet werden.

SAM-Datei
Datei für sequentiellen Zugriff (Sequential Access Method).

Schema
Ein Schema bei SESAM und UDS enthält Verwaltungsdaten zu Objekten der Datenbank wie Views, Indizes und Privilegien. Bei UDS beschreibt es das CODASYL-Subschema in relationaler Form.

SPU
Mit der Software-Produktionsumgebung (SPU) führen Sie Ihren Dialog mit DRIVE/WINDOWS fenster- und menügesteuert. Sie können DRIVE-Programme erstellen, testen und verwalten. Über Menüeinträge (SINIX) / Befehle (MS-Windows) wechseln Sie in den Debug- oder Programm-Modus.

SQL-Anweisung
Anweisungen von DRIVE/WINDOWS, die auf Datenbanken zugreifen. SQL-Anweisungen sind in den „SQL-Lexika“ [8-11] beschrieben.

Startdatei
Datei mit DRIVE-Anweisungen, die beim Start von DRIVE/WINDOWS ausgeführt werden.

statisches Bildschirmformat
Ein mit FHS (BS2000) oder FORMANT (SINIX) erstelltes Bildschirmformat, das mit DECLARE SCREEN im DRIVE-Programm bekanntgegeben wird. Das Format kann von mehreren DRIVE-Programmen verwendet und innerhalb eines Programms beliebig oft ausgegeben werden.

statisches New-Style-Programm
Ein New-Style-Programm heißt statisch, wenn es mindestens eine statische SQL-Anweisung außer COMMIT WORK und ROLLBACK WORK und keine dynamischen SQL-Anweisungen enthält.

statische SQL-Anweisung
Eine SQL-Anweisung heißt statisch, wenn sie explizit in der Source codiert und damit übersetzbar ist

Tabelle
Unter Tabellen sind Basistabellen (s. o.), Ergebnistabellen wie Cursor oder temporäre Tabellen und Systemtabellen des jeweiligen Datenbanksystems zusammengefaßt, die es zur Verwaltung benötigt.

temporäres Objekt
Ein temporäres SQL-Objekt wird anwendungsspezifisch gespeichert (im Gegensatz zum persistenten) und existiert höchstens bis zum Ende der DRIVE-Session. Temporäre SQL-Objekte sind vor allem Cursor sowie temporäre Tabellen, temporäre Views und Synonyme. Ein temporäres DRIVE-Objekt ist die Variable sowie DRIVE-Bildschirm- und Listenformate (BS2000/SINIX), die mit der TEMPORARY-Klausel als Standard-Vorbelegung definiert sind.

Textdatei
Datei, deren Daten in CHARACTER-Darstellung gepeichert sind. Alle Felder sind durch ein Trennzeichen getrennt. Sie enthält Steuerzeichen für Zeilenvorschub (Zeilendezeichen).

TIAM-Betrieb
Teilnehmerbetrieb im BS2000. TIAM-Betrieb entspricht dem Betrieb ohne TP-Monitor unter SINIX.

TP-Monitor
Der Transaction Processing Monitor (TP-Monitor) koordiniert Arbeitsschritte und überwacht, daß zusammengehörende Arbeitsschritte in einer Transaktion vollständig durchlaufen oder - im Fehlerfall - zurückgesetzt werden.

Transaktion
Eine Transaktion ist eine logische Folge von Anweisungen, die zwischen zwei Sicherungspunkten liegen. Bei Transaktionssicherung werden entweder alle oder keine der in einer Transaktion liegenden Anweisungen ausgeführt.

User-Text
Vom Anwender frei definierter Textteil, der innerhalb einer Maske an beliebiger Stelle angegeben werden kann.

UTM-Betrieb
Teilhaberbetrieb im BS2000 mit dem Universellen Transaktionsmonitor (UTM). UTM-Betrieb entspricht dem Betrieb mit TP-Monitor unter SINIX.

UTM-Startprozedur
Datei mit Startparametern für UTM, die beim Start einer UTM-Anwendung ausgewertet werden. Außer den UTM-Startparametern darf die UTM-Startprozedur weitere Anweisungen enthalten.

variabler Cursor

Ein variabler Cursor ist ein statisches DRIVE-Objekt, das zwischen der dynamischen Spezifikation einer FOR-Klausel und der nächsten DROP-Anweisung zu einem dynamischen SQL-Objekt wird.

Verteilte Anwendung

Client-Programme, die z.B. eine grafische Oberfläche unter MS-Windows oder SINIX realisieren, können mit Server-Programmen gekoppelt werden, die z.B. den Datenbankzugriff im BS2000 realisieren (Trennung von Präsentation und Verarbeitung). Der Server-Aufruf mit CALL ist der gleiche wie für lokale Unterprogramm-Aufrufe und wird über die Verteilinformationen (parametrisierbare Verteiltabellen) gesteuert.

Verteilte UTM-Anwendung (VTV)

Eine Anwendung kann Verarbeitungsaufträge an eine entfernte Anwendung geben. Beide Anwendungen müssen im UTM-Betrieb bzw. im Betrieb mit TP-Monitor laufen. Sie führen anwendungsübergreifende Transaktionen aus (verteilte Transaktionsverarbeitung - VTV), so daß sich die Daten aller Anwendungen in einem konsistenten Zustand befinden.

VTV

verteilte Transaktionsverarbeitung, siehe „Verteilte UTM-Anwendung“

Wiederanlauf

Ein interner Wiederanlauf tritt ein nach Rücksetzen der Transaktion durch das Datenbanksystem oder den Datenbank-Administrator oder durch Ausführen der DRIVE-Anweisungen ROLLBACK WORK und ROLLBACK WORK WITH RESET. Ein externer Wiederanlauf ist im Betrieb mit TP-Monitor möglich und tritt beim Hochfahren der UTM-Anwendung nach einem Systemzusammenbruch ein.

Window-Anweisung

DRIVE-Anweisungen, die sich auf Fenster oder Fensterobjekte beziehen.

Zwischencode

Übersetzungsergebnis des Interpreters. Zwischencode erhalten Sie, wenn Sie eine Source mit derr Option CODE=ON übersetzen ohne Einsatz des DRIVE-Compilers DRIVE/WINDOWS-Comp.

Literatur

[1] **DRIVE/WINDOWS** (BS2000)
Programmiersystem
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Einführung in das Programmiersystem DRIVE/WINDOWS und Erläuterung der Funktionen des Dialog-Modus' sowie Beschreibung der Installation, der Generierung und der Administration von DRIVE/WINDOWS

[2] **DRIVE/WINDOWS** (BS2000)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Bildschirm- und Listenformaten und Reports. Beschreibung des Transaktionskonzepts und der Verteilten Transaktionsverarbeitung. Beispiele.

[3] **DRIVE/WINDOWS** (BS2000)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen, Meldungen und Schlüsselwörter von DRIVE/WINDOWS

[4] **DRIVE/WINDOWS** (BS2000/SINIX)
Lexikon der DRIVE-SQL-Anweisungen für SESAM V1.x
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für SESAM V1.x in Kurzform.

[5] **DRIVE/WINDOWS** (BS2000/SINIX)
Lexikon der DRIVE-SQL-Anweisungen für SESAM V2.x
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für SESAM V2.x in Kurzform.

[6] **DRIVE/WINDOWS** (BS2000/SINIX)
Lexikon der DRIVE-SQL-Anweisungen für UDS
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für UDS in Kurzform.

[7] **DRIVE/WINDOWS** (MS-Windows)
Software-Produktionsumgebung (SPU)
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Erläuterung der Funktionen der Software-Produktionsumgebung (Arbeitsplatz). Einsatzvorbereitung für DRIVE/WINDOWS, den Remote-Zugriff auf BS2000- und SINIX-Datenbanken und für Client-Server-Anwendungen.

[8] **DRIVE/WINDOWS** (MS-Windows)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Grafik-Bildschirmformaten und Client-Server-Anwendungen.

[9] **DRIVE/WINDOWS** (MS-Windows)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen, Meldungen und Schlüsselwörter von DRIVE/WINDOWS.

[10] **DRIVE/WINDOWS** (SINIX)
Software-Produktionsumgebung (SPU)
Benutzerhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Erläuterung der Funktionen der Software-Produktionsumgebung (Arbeitsplatz) und des Expertenmodus. Einsatzvorbereitung für Remote-Zugriff auf BS2000-Datenbanken, für das Erstellen von Anwendungen für das BS2000 und für DRIVE/WINDOWS allgemein.

[11] **DRIVE/WINDOWS** (SINIX)
Programmiersprache
Sprachbeschreibung

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Grafik- und Alpha-Bildschirmformaten sowie Listenformaten mit DRIVE und Report Generator.

[12] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-Anweisungen
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-Anweisungen. Meldungen und Schlüsselwörter von DRIVE/WINDOWS.

[13] **DRIVE/WINDOWS** (SINIX)
Lexikon der DRIVE-SQL-Anweisungen für INFORMIX
Referenzhandbuch

Zielgruppe
Anwendungsprogrammierer
Inhalt
Syntax und Funktionsumfang aller DRIVE-SQL-Anweisungen für INFORMIX in Kurzform.

[14] **DRIVE V5.1** (BS2000)
Teil 1: Benutzerhandbuch

Zielgruppe
– DV-Laien in der Fachabteilung
– Anwendungsprogrammierer
Inhalt
– Allgemeiner Überblick über das System DRIVE in OLD-Style
– Erläuterung der DRIVE-Komponenten
– Beschreibung möglicher Anwendungsfälle anhand einführender Beispiele
– Generierung und Administration von DRIVE im UTM-Betrieb

[15] **DRIVE V5.1** (BS2000)
Teil 2: LEXIKON

Zielgruppe
– DV-Laien in der Fachabteilung
– Anwendungsprogrammierer
Inhalt
– Syntax und Funktionsumfang aller DRIVE-Anweisungen in OLD-Style
– Meldungen und Schlüsselwörter von DRIVE

[16] **DRIVE/WINDOWS-COMP** (BS2000)
Benutzerhandbuch

Inhalt
Beschreibung der Sprachabweichungen zu DRIVE/WINDOWS V1.1 und Darstellung des Compilierungsvorganges. Beschreibung des Generierens und Startens von Anwendungen von compilierten DRIVE-Objekten (TIAM- und UTM-Betrieb) unter besonderer Berücksichtigung des Versionsmischbetriebes.

[17] **SQL für SESAM/SQL**
Sprachbeschreibung

Zielgruppe
Programmierer, die mit SQL-Anweisungen auf SESAM-Datenbanken zugreifen wollen.
Inhalt
SQL-Anweisungen für den Zugriff auf SESAM-Datenbanken.

[18] **SESAM-SERVER** (BS2000/OSD)
SQL-Sprachbeschreibung Teil 1: SQL-Anweisungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle Personen, die eine SQL-Datenbank mit SQL-Anweisungen bearbeiten.
Inhalt
Das Handbuch beschreibt die Programmeinbettung von SQL-Anweisungen und die SQL-Sprachelemente. In einem alphabetischen Nachschlagteil sind alle SQL-Anweisungen ausführlich dargestellt.

[19] **SESAM/SQL-Server** (BS2000/OSD)
SQL-Sprachbeschreibung Teil 2: Utilities
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle Personen, die mit der Verwaltung einer SESAM/SQL-Datenbank befaßt sind.
Inhalt
Das Handbuch enthält eine alphabetische Beschreibung der Utility-Anweisungen; Utility-Anweisungen sind Anweisungen in SQL-Syntax und realisieren die Dienstprogrammfunktionen von SESAM/SQL.

[20] **SESAM/SQL-Server** (BS2000/OSD)
Basishandbuch
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an alle Anwender und alle, die sich über SESAM/SQL informieren wollen.
Inhalt
Das Handbuch gibt einen Überblick über das Datenbanksystem und beschreibt Grundlagen, Konzepte und Zusammenhänge. Es ist die Basis für das Verständnis der weiteren SESAM/SQL-Handbücher.

[21] **SESAM/SQL-Server** (BS2000/OSD)
SESAM/SQL-Server Utility-Monitor
Benutzerhandbuch

Zielgruppe
Das Handbuch ist für den Datenbankverwalterund den Systemverwalter von SESAM/SQL-Server bestimmt.
Inhalt
Das Handbuch beschreibt die Bedienung des Utility-Monitors. Mit dem Utility-Monitor können DB-Verwaltungs- und Administrationsaufgaben u.a. im maskengeführten Dialog ausgeführt werden.

[22] **SESAM/SQL-Server** (BS2000/OSD)
Umstellen von SESAM-Datenbanken u.-Anwendungen auf SESAM/SQL-Server
Benutzerhandbuch

Zielgruppe
Das Handbuch richtet sich an alle, die sich für SESAM/SQL-Server V2.0 interessieren.
Inhalt
Dieses Handbuch beschreibt die neuen Konzepte und Funktionen im Überblick. Im Vordergrund steht der Bezug zu Vorgängerversionen, um bisherigen SESAM/SQL-Anwendern den Umstieg in die neue Welt von SESAM/SQL-Server V2.0 zu erleichtern.

[23] **SESAM/SQL-Server** (BS2000/OSD)
CALL-DML-Anwendungen
Benutzerhandbuch

Zielgruppe
Das Handbuch richtet sich an alle CALL-DML-Anwendungen-Progammierer.
Inhalt
Es enthält die Beschreibung der CALL-DML-Schnittstelle mit den DML-Anweisungen und dazugehörigen Beispielen. Außerdem sind unter anderem Binden, Laden, Anwenden im Teilhaberbetrieb und die CALL-DML-Dienstprogramme beschrieben.

[24] **SESAM/SQL-Server** (BS2000/OSD)
Meldungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle SESAM/SQL-Anwender.
Inhalt
Das Handbuch enthält sämtliche Meldungen zu SESAM/SQl nach Meldungsnummern sortiert.

[25] **SQL für UDS/SQL**
Sprachbeschreibung

Zielgruppe
Programmierer, die mit SQL-Anweisungen auf UDS-Datenbanken zugreifen wollen.
Inhalt
SQL-Anweisungen für den Zugriff auf UDS-Datenbanken.

[26] **UDS/SQL** (BS2000)
Verwalten und Bedienen
Benutzerhandbuch

Zielgruppe
Datenbankadministrator
Inhalt
– Verwaltungs- und Bedienungsarbeiten wie Sichern der Datenbank,
– Datenbankbetrieb
– Ausgeben von Datenbankinformationen
– Reorganisieren der Datenbank Datenbankinformationen,
– Konsistenzprüfprogramm
Einsatz
Datenbankadministration im laufenden Betrieb

[27] **UDS/SQL** (BS2000)
Aufbauen und Umstrukturieren
Benutzerhandbuch

Zielgruppe
Datenbankadministrator
Inhalt
– Übersicht über die von UDS benötigten Dateien
– UDS-Dienstprogramme, die zum Aufbauen der UDS-Datenbank nötig sind
– Dienstprogramme zum Umstrukturieren
Einsatz
Datenbankadministrator beim Aufbauen einer Datenbank

[28] **IFG für FHS** (TRANSDATA)
Benutzerhandbuch

Zielgruppe
Datenstationsbenutzer, Anwendungsdesigner und Programmierer
Inhalt
Der Interaktive Formatgenerator (IFG) ist ein System zur komfortablen und einfachen Erstellung und Verwaltung von Formaten an Datensichtstationen. Diese Formate können zusammen mit FHS im Verarbeitungsrechner eingesetzt werden. Das Benutzerhandbuch beschreibt, wie die Formate erstellt, geändert und verwaltet werden sowie die neuen Funktionen von IFG V8.1.

[29] **FHS** (TRANSDATA)
Benutzerhandbuch

Zielgruppe
Programmierer
Inhalt
Programmschnittstellen von FHS für TIAM-, DCAM- und UTM-Anwendungen. Erstellen, Einsatz und Verwalten von Formaten.

[30] **UTM** (BS2000/OSD)
Anwendungen generieren und administrieren
Benutzerhandbuch

Zielgruppe
Organisierer, Einsatzplaner und Administratoren von UTM-Anwendungen.
Inhalt
– Installation von UTM
– Einrichten, Bedienen und Verwalten von UTM-Anwendungen
– UTM-Benutzerkommandos

[31] **UTM** (TRANSDATA)
Anwendungen programmieren
Benutzerhandbuch

Zielgruppe
Programmierer von UTM-Anwendungen
Inhalt
– Sprachunabhängige Beschreibung der Programmschnittstelle KDCS,
– Aufbau von UTM-Programmen
– KDCS-Aufrufe
– Testen von UTM-Anwendungen
– Alle Informationen, die der Programmierer von UTM-Anwendungen benötigt
Einsatz
BS2000-Transaktionsbetrieb

[32] **UTM** (SINIX)
Formatierungssystem

Zielgruppe
UTM(SINIX)-Anwender, die mit Formaten arbeiten wollen, C-Programmierer und COBOL-Programmierer
Inhalt
Einsetzen der Formatsteuerung FORMANT in UTM(SINIX)-Teilprogrammen, Erstellen von Formaten, konvertieren von Formaten zwischen BS2000 und SINIX.

[33] **EDT** (BS2000/OSD)
Anweisungen
Benutzerhandbuch

Zielgruppe
EDT-Einsteiger und EDT-Anwender
Inhalt
Bearbeiten von SAM- und ISAM-Dateien und Elementen aus Programm-Bibliotheken und POSIX-Dateien.

[34] **LMS** (BS2000)
ISP-Format
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Beschreibung der Anweisungen zum Erstellen und Verwalten von PLAM-Bibliotheken und darin enthaltenen Elementen.
Häufige Anwendungsfälle werden an Hand von Beispielen erklärt.

[35] **BS2000/OSD-BC**
Kommandos Band 1 - 3
Benutzerhandbuch

Zielgruppe
Die Handbücher wenden sich sowohl an den nichtprivilegierten Anwender als auch an die Systembetreuung.
Inhalt
Sie enthalten die BS2000/OSD-Kommandos (BS2000/OSD-Grundausbau und ausgewählte Produkte) mit der Funktionalität für alle Privilegien. Die Einleitung gibt Hinweise zur Kommandoeingabe.

[36] **BS2000/OSD-BC**
Systeminstallation
Benutzerhandbuch

Zielgruppe
BS2000/OSD-Systemverwaltung
Inhalt
Das Handbuch beschreibt
– die Generierung der Hardware- und Software-Konfiguration mit UGEN
– die Installationsdienste
  – Plattenorganisation mit MPVS
  – Programmsystem SIR
  – Datenträgerinstallation mit SIR
  – Configuration Update (CONFUPD)
  – Dienstprogramm IOCFCOPY.

[37] **BS2000/OSD-BC**
Einführung in das DVS
Benutzerhandbuch

Zielgruppe
Das Handbuch wendet sich an den nichtprivilegierten Anwender und an die Systembetreuung.
Inhalt
Es beschreibt die Dateiverarbeitung im BS2000.
Themenschwerpunkte:
- Datei- und Katalogverwaltung
- Dateien und Datenträger
- Datei- und Datenschutz
- OPEN-, CLOSE-, EOV-Verarbeitung
- DVS-Zugriffsmethoden (SAM, ISAM,...)

[38] BS2000
Einführung in die Systemanwendung
Benutzerhandbuch

Zielgruppe
BS2000-Anwender
Inhalt
– Einführung ins BS2000
– Beschreibung der meistgebrauchten Benutzerkommandos bis BS2000 V8.5A
– Einführung in die Benutzung der Dienstprogramme und Softwareprodukte EDT, SORT, ARCHIVE, TSOSLNK, LMS, PERCON
– Hinweise für den programmierenden Benutzer
Einsatz
BS2000-Dialogbetrieb und -Stapelbetrieb

[39] **FORMANT** (SINIX)
Beschreibung

Zielgruppe
– C-Programmierer
– COBOL-Programmierer
– Anwendungsplaner
Inhalt
FORMANT ist eine Maskensteuerung für alle SINIX-Systeme. Das Manual enthält:
– Einführung in FORMANT
– Beschreibung von FORMANTGEN
– Beschreibung der Bedienerschnittstelle
– Programmschnittstellen in C und COBOL
– Beispiele zur Programmierung

[40] **OMNIS** (TRANSDATA, BS2000)
Administration und Programmierung
Benutzerhandbuch

Zielgruppe
– OMNIS-Administrator
– Programmierer
Inhalt
Beschreibung der Grundlagen der Administration von OMNIS, der OMNIS-Dienstprogramme sowie der Anwendungsschnittstelle zur Erweiterung des Funktionsumfanges von OMNIS.

[41] **DRIVE/WINDOWS-COMP** (SINIX)
Compiler
Benutzerhandbuch
Zielgruppe
Anwendungsprogrammierer und Systemverwalter
Inhalt
Beschreibung des Compilierungsvorgangs durch den DRIVE-Compiler.

[42] **INFORMIX-NET** (SINIX)
INFORMIX-STAR (SINIX)
Benutzerhandbuch

Zielgruppe
INFORMIX-Benutzer und Systemverwalter
Inhalt
Das Handbuch beschreibt die Arbeit mit den INFORMIX-Netzprodukten INFORMIX-NET und INFORMIX-STAR.
Mit den INFORMIX-Netzprodukten können INFORMIX-Anwendungen von einem lokalen Rechner aus Datenbanken auf fernen Rechnern erstellen und bearbeiten.

[43] **SESAM/SQL-Server** (BS2000/OSD)
Meldungen
Benutzerhandbuch

Zielgruppe
Zur Zielgruppe gehören alle SESAM/SQL-Anwender.
Inhalt
Das Handbuch enthält sämtliche Meldungen zu SESAM/SQl nach Meldungsnummern sortiert.

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis* der Siemens Nixdorf Informationssysteme AG. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an Ihre zuständige Geschäftsstelle. Dort können Sie auch die Handbücher bestellen.

Stichwörter

C

G

M

N

Q

R

S

U

W

X

Z

Inhalt

DRIVE/WINDOWS V2.1 (BS2000)

Programmiersprache

Zielgruppe
Anwendungsprogrammierer
Inhalt
Beschreibung der Programmerstellung einschließlich Alpha-Bildschirmformaten sowie Listenformaten mit DRIVE und Report-Generator.

Ausgabe: Februar 1996

Datei: DRV_PROG.PDF

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.