1 Preface

This chapter briefly describes the product DRIVE, the target group for this manual and the organization of the suite of DRIVE manuals. It also contains a list of changes incorporated since the last version of the manual and explains the notational conventions used in the DRIVE manuals.

1.1 Brief product description

DRIVE is a fourth-generation programming language (4GL) for the development of commercial client/server applications. It is the 4GL used to access the BS2000 database system SESAM/SQL V2 and to access files.

The uniform language with its powerful and easily learned statements allows programmers to create complex applications for database access, reports, user interfaces, communications and processing. DRIVE automatically provides system-specific interfaces to components, thus relieving the programmer of this task.

DRIVE provides programmers with an integrated debugger to help them test their DRIVE applications.

DRIVE applications can be created and tested with or without a transaction monitor and can run unmodified irrespective of whether or not a transaction monitor is connected.

Performance can be improved by compiling the DRIVE applications using the DRIVE-COMP compiler.

1.2 Target group

This manual is aimed at programmers who develop DRIVE applications or components of Distributed Transaction Processing (DTP) applications using DRIVE on BS2000 computers. This means that programmers must be familiar with the BS2000 operating system.

Depending on the application in question, programmers may need an understanding of:

– the SESAM database system

– the *open*UTM transaction monitor

– the FHS Format Handling System for creating screen forms

1.3 Summary of contents

This manual supplements the manuals for DRIVE/WINDOWS V2.1. It describes the new and modified functions of DRIVE V2.2. Each chapter refers specifically to a manual from the range of DRIVE/WINDOWS V2.1 (BS2000) documentation; an additional chapter provides manual corrections for DRIVE/WINDOWS V2.1.

The individual chapters relate to the following manuals:

- Chapter 2 contains corrections for “DRIVE Programming Language” [2], “DRIVE Directory” [3] and “DRIVE SQL Directory” [4].
- Chapter 3 refers to “DRIVE Programming System” [1].
- Chapter 4 refers to “DRIVE SQL Directory” [4].
- Chapter 5 refers to “DRIVE Directory” [3].
- Chapter 6 and 7 refer to “DRIVE Programming Language” [2].

1.4 README file

Please refer to the product-specific README file for any functional modifications or additions to the current product version. You can find the README file on your BS2000 computer under the file name SYSRME.*product*.*version*.*language*. Please ask your systems support staff for the login name under which the README file is stored. You can view the file with the /SHOW-FILE command or by opening it in an editor, or you can print it at the default printer by entering the following command:

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

or, in the case of SPOOL versions earlier than V3.0A:

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

1.5 Changes compared to DRIVE V2.1

1.5.1 Components

- DRIVE V2.2 only operates with SESAM V2.x or later. The descriptions for implementing DRIVE (see chapter 3, “Implementing DRIVE”, on page 13) and for connecting databases (see chapter 7, “Databases”, on page 183) have thus been modified.
- Temporary views are no longer supported as of SESAM/SQL V3.0.
- FHS-DE is not supported. Therefore you cannot use the DRIVE statements ADD BOX, REMOVE BOX and REPLACE BOX as well as the FHS-DE specific parameters of the statement DISPLAY screenform.

1.5.2 New DRIVE SQL statements

All of the statements listed below can also be executed dynamically.

- New SQL statements for managing user entries

Statement	Page
CREATE SYSTEM_USER	113
CREATE USER	119
DROP SYSTEM_USER	131
DROP USER	134

- New SQL statements for managing the memory structure

Statement	Page
ALTER SPACE	85
ALTER STOGROUP	87
CREATE INDEX	103
CREATE SPACE	108
CREATE STOGROUP	111
DROP INDEX	127
DROP SPACE	129

Statement	Page
DROP STOGROUP	130
REORG STATISTICS	151

- New UTILITY statements for database management, see page 157:

  ALTER MEDIA DESCRIPTION
  CHECK CONSTRAINTS
  CHECK FORMAL
  COPY
  CREATE CATALOG
  CREATE MEDIA DESCRIPTION
  CREATE REPLICATION
  DROP MEDIA DESCRIPTION
  LOAD
  MIGRATE
  MODIFY
  RECOVER
  REFRESH REPLICATION
  REORG
  UNLOAD

- New pragma clauses, see page 141f:

  JOIN
  LOCK MODE
  UTILITY MODE

1.5.3 Extended DRIVE SQL statements

- SQL statements for database management

Statement	Extension	Page
ALTER TABLE	column-definitions	89
CREATE SCHEMA	CREATE INDEX	106
CREATE TABLE	CALL DML	115
DROP SCHEMA	CASCADE	128
DROP TABLE	CASCADE	133

Statement	Extension	Page
DROP VIEW	CASCADE	135
GRANT	special privileges	136
REVOKE	special privileges	152

- SQL statements for querying and changing data

Statement	Extension	Page
DECLARE	FOR READ ONLY	120

1.5.4 Handling SESAM warnings and messages

- Exception handling of SESAM warnings:

  The system variable &WARNING has been added to the DRIVE statement WHENEVER, see page 160f.

- Determining SQL message texts:
  The new SQLMSGSTRING function can be used to determine SQL message texts, see page 165.
- Addition of &WARNING to the system variable &ERROR_STATE and &DIS_WARNING to the system variable &DISTRIBUTION_STATE.

1.5.5 Other changes

- The POSITION keyword must be specified for position specifications with the report statement PAGE PRINT, see page 164.
- The possible uses of the abbreviation ".*" have changed, see page 178.

1.6 Notational conventions

The symbols and fonts used in the DRIVE/WINDOWS manuals have the following meanings:

`type-written text`

is used for fixed names (e.g. operating system commands, file names) and error messages in the running text. It is also used in examples.

Italics

are used in secondary headings to denote examples and, in continuous text, for freely selectable names and metavariables.

i This character identifies very important information that it is essential that you read.

The metalanguage used is described in the “Directory of DRIVE Statements” [3[.

References to other publications, e.g. the manuals mentioned above, consist of an abbreviated title together with a number in square brackets. The appendix contains a section (“Related publications”) that lists these publications in ascending order by the number in the brackets.

Syntax of the DRIVE and the DRIVE SQL statements

The following notation has been used for the formal representation of statements and metavariables.

Formal representation	Meaning	Example
UPPERCASE LETTERS	Uppercase letters denote a keyword which must be entered in the form shown.	COLUMNS
Boldface	Letters in boldface denote the abbreviation for a keyword.	**PERM**ANENT
Lowercase letters	Lowercase letters denote a variable for which you must enter the current value.	LIBRARY=*lib-name*

Formal representation	Meaning	Example
( )	Parentheses are an integral part of the statement. Parentheses must be entered if a value is shown in parentheses.	*lib*(*member-name*) or CONCAT (*char-expression1*, *char-expression2*) or ATTRIBUTE (*attribute*, ...)
{ }	Braces are used to enclose units. Braces are read from the inside towards the outside. Braces must not be entered.	STATUS={ OFF | ADD | REMOVE } or USING { [RETURN] [*level*] *var-name data-type* }, ...
[ ]	Square brackets enclose optional specifications. Brackets are read from the inside towards the outside. Square brackets must not be entered.	[*set transaction*] or [COBOL | C] TAC *tacname*
< >	Angle brackets are an integral part of the statement. Angle brackets must be entered if a value is shown in angle brackets.	*aggregate*=< {*value* | NULL}, ... >
|	A vertical line separates alternative operand values. One of the alternatives shown in braces must be entered.	LETTERS={ CAPITAL | BOTH | UNCHANGED }
...	An ellipsis indicates that the variable which immediately precedes the ellipsis can be repeated several times. If the ellipsis is preceded by a unit enclosed in brackets, the entire unit must be entered. If a comma or semicolon precedes the ellipsis, it must be specified in each of the repetitions in order to separate the specifications from each other.	AT *line* ... USING { [RETURN] [level] *var-name data-type* }, ... (*attribute*, ...)

2 Error handling

The set of DRIVE/WINDOWS V2.1 manuals unfortunately contained a number of errors, which are corrected below.

2.1 Corrections for “DRIVE Programming Language”

The manual “DRIVE/WINDOWS - Programming Language” [2] has to be corrected as follows:

Data type conversion compatibility

(see section 3.6, page 80, explanation of topic 19 of the table on page 76)

19 The conversion is assignment-compatible and comparable. A comparison between different data types always returns the value “not equal to”.

DATE → TIMESTAMP(3)
During the conversion, the current time is inserted for *hour*:*minute*:*second*.*fraction*.

2.2 Corrections for “DRIVE Directory”

The manual “DRIVE/WINDOWS V2.1- Directory of DRIVE Statements” [3] has to be corrected as follows:

DECLARE VARIABLE - Define variable

(see chapter 3, page 72)

The LIKE clause for copying a cursor or table to a variable is only permitted on the top level (Level = 1).

OPTION - Control compilation of a program

(see chapter 3, page 141)

The following parameter statements are forbidden in the program when compiling a DRIVE program with the option OBJECT=ON for creating an object code:

– all static parameters

– dynamic parameters LOG, LOGFILE, LOGPASSWORD, NORMSQL, SCHEMA, CATALOG, AUTHORIZATION and TEST

PARAMETER LOCK - Lock statement

(see chapter 3, page 169)

The DELETE and UPDATE statements can be locked as follows:

– DELETE { SEARCHED | POSTIONED }

– UPDATE { SEARCHED | POSTIONED }

READ FILE - Read a file

(see chapter 3, page 177)

The DRIVE system variables &PHYS_REC_LENGTH and &DRIVE_REC_LENGTH do not exist.

null-value - Define a representation of the null value

(see chapter 5, page 334)

The default setting for null value representation for screen input/output is the character ‘@’.

2.3 Corrections for “DRIVE SQL Directory”

The manual “DRIVE/WINDOWS V2.1- Directory of DRIVE SQL Statements for SESAM/SQL 2” [4] has to be corrected as follows:

DECLARE - Declare cursor

(see chapter 3, page 88)

```
DECLARE CURSOR [ { PERMANENT | TEMPORARY } ]
          [ SCROLL ]  CURSOR  [ PREFETCH n ]
          [ FOR cursor_description ]
```

SET TRANSACTION – Define transaction attributes

(see chapter 3, page 142)

When a DRIVE UTM application is in dialog mode, the SET TRANSACTION statement has no effect on the following SQL transaction.

This is because the SET TRANSACTION statement does not open the SQL transaction and DRIVE acknowledges the execution of the statement with message DRI0009, whereby the UTM transaction is terminated. Due to transaction synchronization with SESAM, the SQL transaction is thus also “terminated”.

The default setting of the transaction level is hence reactivated.

3 Implementing DRIVE

This chapter contains the changes and supplements to “DRIVE Programming System” [1]. The specified section and chapter numbers refer to this manual.

3.1 Starting and terminating the DRIVE dialog

This chapter describes

– the dialog structure in TIAM applications (from page 14)

– the dialog structure in UTM applications (from page 20)

– how to invoke DRIVE from a BS2000 procedure (from page 25)

3.1.1 Dialog structure in TIAM applications

Structure of the dialog with DRIVE in TIAM applications:

Names in lowercase letters must be replaced by the names valid in the environment involved.

`/LOGON userid,accountno,'password'`	Initiate a BS2000 task

`/SET-FILE-LINK LINK-NAME=DRIVEOML,-`
`/  FILE-NAME=SYSLNK.DRIVE.022`
`/SET-FILE-LINK LINK-NAME=LIBOML,-`
`/  FILE-NAME=SYSPRG.DRIVE.022`
	Assign DRIVE module libraries

`/SET-FILE-LINK LINK-NAME=BLSLIB01,-` `/  FILE-NAME=crte-lib`	Assign CRTE runtime library
`/SET-FILE-LINK LINK-NAME=BLSLIB02,-` `/  FILE-NAME=lms-lib`	Assign LMS runtime library
`/SET-FILE-LINK LINK-NAME=BLSLIB03,-` `/  FILE-NAME=fhsmacro-lib`	Assign FHS runtime library
`/SET-FILE-LINK LINK-NAME=BLSLIB04,-` `/  FILE-NAME=systemmacro-lib`	Assign system macro library (library containing the module DCCOBRTS)

`/SET-FILE-LINK LINK-NAME=SESAMOML,-`
`/  FILE-NAME=sesam-lib`
`/SET-FILE-LINK LINK-NAME=SESCONF,-`
`/   FILE-NAME=sesam-conf`
	Assign SESAM module library and configuration
	(only if you are working with a SESAM database)

`/SET-FILE-LINK LINK-NAME=FORMOML,-` `/  FILE-NAME=format-lib`	Assign forms library (only if you are working with FHS forms)
`/SET-FILE-LINK LINK-NAME=MROUTLIB,-` `/  FILE-NAME=fhsrts-lib`	Assign FHS runtime library (only if your are working with FHS forms)
`/SET-FILE-LINK LINK-NAME=RSOML,-` `/  FILE-NAME=SYSLIB.DRIVE.022`	Assign library for the report generator (only if you are working with reports)
`/SET-FILE-LINK LINK-NAME=USEROML,-` `/  FILE-NAME=usr-lib`	Assign DRIVE library (not necessary if you assign the DRIVE library with the DRIVE statement PARAMETER DYNAMIC LIBRARY, see section 3.1.1.2, “Parametrizing the dialog”, on page 18)

`/START-PROGRAM FROM-FILE=*MOD(-`
`/  LIB=obj-lib,ELEM=mod-name,PROG-MO=ANY,-`
`/  RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,-`
`/              UN-EXTRNS=DELAY,LO-IN=REF))`
	Call the generated DRIVE variant (LLM) (see page 36)

`PARAMETER`	Enter DRIVE parameters
Work in interactive mode:	

`DECLARE ... CURSOR`
`FETCH ...`
`INSERT ...`
`UPDATE ...`
	DRIVE statements in interactive mode

or in program mode:	
DO program-name	Data processing with DRIVE programs
or branch to EDT:	
EDT . . . HALT	Create DRIVE programs
or work in debugging mode:	
DEBUG program-name . . . BREAK DEBUG	Debug DRIVE programs
STOP	End dialog with DRIVE
/LOGOFF	Terminate BS2000 task

3.1.1.1 Starting the dialog

➤ First initiate a BS2000 task with the BS2000 command /LOGON ...

```
/LOGON userid,accountno,'password'
```

The BS2000 operating system issues a message indicating that the BS2000 task has been initiated. It awaits further BS2000 commands.

➤ The module library containing the DRIVE object modules should now be assigned as the DRIVE object module library. Use the BS2000 command:

```
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
```

In order to run, DRIVE requires internal DRIVE programs whose intermediate code is located in the library supplied under the name SYSPRG.DRIVE.022.

➤ Assign this library using the BS2000 command:

```
/SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
```

For external references to be resolved by the dynamic linking loader, the runtime libraries of CRTE, LMS and FHS, as well as the system macro library, must be assigned.

➤ Assign these libraries using the following BS2000 commands:

```
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crte-lib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lms-lib
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhsmacro-lib
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=systemmacro-lib
```

For the SESAM variant, DRIVE requires SESAM connection modules at the time of execution. The module library containing the SESAM connection modules must be assigned as the SESAM object module library.

➤ Use the BS2000 command:

```
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesam-lib
```

➤ Now assign the configuration of the SESAM database with which you wish to work (only in the SESAM variant):

```
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesam-conf
```

If you are working with FHS forms, DRIVE needs the runtime library of FHS and the format library with the FHS forms.

➤ Assign the format library with the following BS2000 command:

```
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=format-lib
```

and the runtime library with the BS2000 command:

```
/SET-FILE-LINK LINK-NAME=MROUTLIB,FILE-NAME=fhsrts-lib
```

This assignment of the runtime library is not necessary if the modules are loaded from the user file TASKLIB or the system file $TSOS.TASKLIB. See “FHS” [14] for the search sequence.

If you are working with reports, DRIVE needs the library with the modules for the report generator.

➤ Assign this library with the following BS2000 command:

```
/SET-FILE-LINK LINK-NAME=RSOML,FILE-NAME=SYSLIB.DRIVE.022
```

If you want to store your program sources, copy members, interpreter listings, intermediate code, and user labels in a particular library, you must assign a user library.

➤ Assign your own user library using the command:

```
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usr-lib
```

When assigning this user library with the DRIVE statement PARAMETER DYNAMIC LIBRARY, the above assignment is only necessary if user-specific programs are called with the DRIVE program statement CALL MODULE.

Call the generated DRIVE variant using the /START-PROGRAM command. The DRIPRC.INSTALL.DRIVE procedure (supplied in the library SYSPRC.DRIVE.022) can be used to assign any name to the generated link and load module (see section 3.3, “Generating DRIVE for TIAM applications”, on page 36). This is necessary if two or more link and load modules are provided under one ID.
If necessary, ask your system administrator for the name of the desired link and load module.

➤ Start the generated DRIVE variant with the following command:

```
/START-PROGRAM FROM-FILE=*MOD(LIB=obj-lib,ELEM=module-name,PROG-MO=ANY,-
/  RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,UN-EXTRNS=DELAY,LO-IN=REF))
```

The desired DRIVE variant is now called, loaded and started. A message is output indicating that DRIVE is loaded, followed by a blank screen with an asterisk (*) as the prompt symbol. DRIVE awaits input.

3.1.1.2 Parametrizing the dialog

The statement PARAMETER on its own results in the following screen display:

```
PAR01                                   PARAMETER
_____________________________________________________________________________

  SELECT PARAMETER STATEMENT

        ( ) PARAMETER STATIC

        ( ) PARAMETER DIAGNOSIS

        ( ) PARAMETER DYNAMIC

        ( ) PARAMETER KFKEY

        ( ) PARAMETER LOCK DIAOLG

        ( ) PARAMETER LOCK PROCEDURE


_____________________________________________________________________________
                                ( B = BREAK )


LTG       EM:1                                                      TAST
```

You can now branch to one of the six follow-up screens by entering an “X”. You can also reach that point directly by entering PARAMETER with the appropriate keyword, e.g. PAR STATIC.

Another method of parameter assignment is direct input of the individual PARAMETER statements, e.g. PARAMETER DYNAMIC LIBRARY=drive-lib.

All PARAMETER screens have already been supplied with default values with which you can work. If, however, you wish to work with different values, you need to overwrite the defaults and then press DUE, i.e. the transmission key.

You will find a detailed description of the PARAMETER statements in the “DRIVE Directory” [3].

Example:

You wish to change the default parameter assignments by specifying your name under USER in PARAMETER STATIC, and DRIVE library PLAM.LIB.DRIVE under LIBRARY in PARAMETER DYNAMIC.

To do this, you can either use PARAMETER STATIC or PARAMETER DYNAMIC to output the appropriate screen and overwrite the default values, or you can enter the following statements directly.

```
PARAMETER STATIC USER = 'MAYOR'
PARAMETER DYNAMIC LIBRARY = "PLAM.LIB.DRIVE"
```

To confirm the parameter assignments, DRIVE issues the following message on the message line (bottom line of the screen):

```
% DRI0009  STATEMENT EXECUTED
```

3.1.1.3 Terminating the dialog

The dialog with DRIVE is terminated by entering the STOP or EXIT statement.

All resources used by DRIVE are released. Open transactions are handled in the following manner:

STOP The dialog can only be terminated with STOP if all open transactions are closed. You can terminate transactions with the COMMIT or ROLLBACK WORK statement (see “DRIVE SQL Directory [4]).

EXIT The EXIT statement terminates the DRIVE dialog even if there are open transactions. These transactions are rolled back. The contents of EDT work file 0 are not saved. EXIT is only allowed in interactive mode and program mode and only in screen inputs in programs.

The following message is displayed on the screen after you terminate the DRIVE dialog with STOP or EXIT:

```
% DRI0088 'xxxxxx' TERMINATED NORMALLY
```

Terminate BS2000 with the command /LOGOFF. BS2000 then issues a message indicating that the task has ended.

3.1.2 Dialog structure in UTM applications

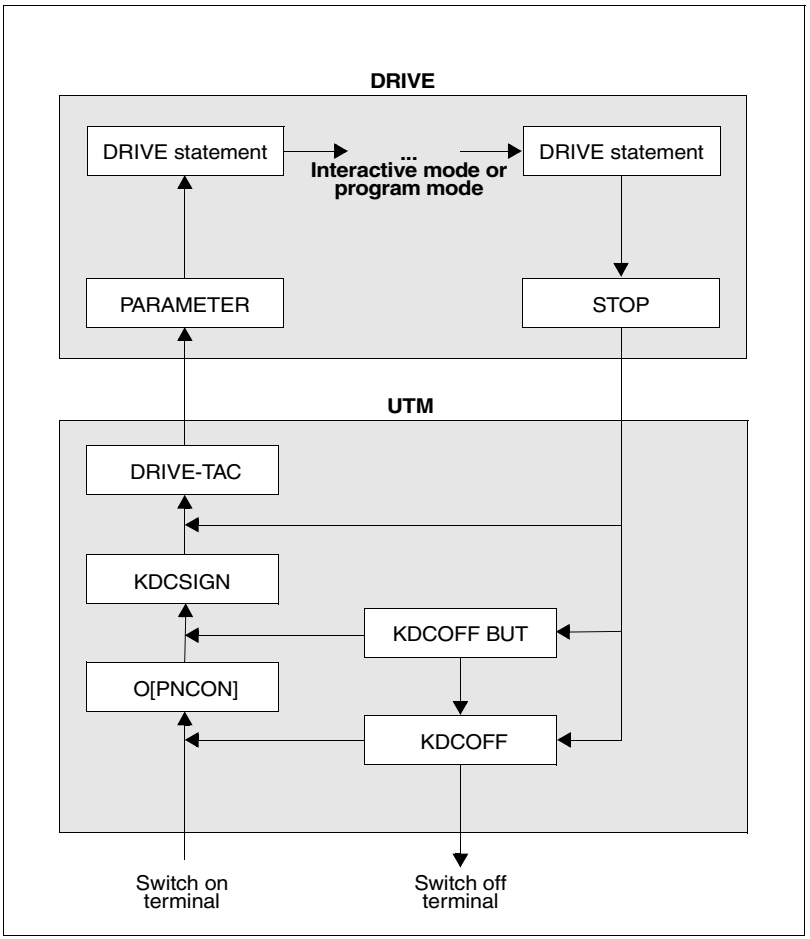
In DRIVE UTM applications, the user communicates with DRIVE via the UTM universal transaction monitor. This means that

– all data entered at a terminal is transferred to DRIVE via UTM

– all data output by DRIVE is transferred to your terminal (or a printer) via UTM.

A connection to UTM must therefore be created before you initiate the DRIVE dialog. Similarly, this connection must be cleared after the DRIVE dialog has been terminated.

The following diagram shows the steps that have to be executed in order to use DRIVE under UTM, from switching the terminal on to switching it off again:



During the DRIVE dialog, you can enter the following UTM transaction codes (TACs) as well as DRIVE statements:

TAC	Meaning
KDCLAST	Repeats the last DRIVE output or an asynchronous message
KDCOUT	Retrieves an asynchronous message
KDCDISP	Rebuilds the last screen after an asynchronous message has been retrieved.
KDCOFF	Terminates interactive mode. There may be no transactions open. This can also be automatically issued by UTM if the dialog is interrupted because of a timeout. This may happen if there are long pauses between entries in an open transaction, for example.

3.1.2.1 Starting the dialog

The following steps are involved in starting a dialog with DRIVE:

– connecting to the UTM application

– signing on to the UTM application

– calling DRIVE

Connecting to the UTM application

You set up the connection to the UTM application with the statement:

```
O[PNCON] application-name[,pp/rr][PW=C'connection-password']
```

where:

application-name Name of the UTM application.

pp/rr Processor and region number of the host on which the UTM application is being run

connection-password Only required if the system administrator assigned a connection password when the UTM application was initiated

Once the connection to the UTM application has been set up, the following message is displayed on the screen:

```
K002 CONNECTED WITH APPLICATION application-name - PLEASE SIGN ON
```

Signing on to the UTM application

You then sign on to the UTM application by entering the KDCSIGN command:

```
KDCSIGN userid[,password]
```

where:

userid	UTM user identification
password	The password assigned to the UTM user identification by the system administrator

If KDCSIGN has been entered correctly, the following message is displayed on the screen:

```
K008 SIGNON IS ACCEPTED – INPUT PLEASE
```

Calling DRIVE

You start DRIVE by entering the UTM transaction code (TAC) for DRIVE. The predefined UTM transaction code is DRISQL. However, the system administrator can alter this when generating the UTM application. If necessary, therefore, ask your system administrator for the valid UTM transaction code for DRIVE.

Once you have entered the transaction code for DRIVE, DRIVE issues the following message: `% DRI0008 PLEASE ENTER STATEMENT`

DRIVE now waits for DRIVE statements to be entered.

The above message (DRI0008) is only output if PERMIT was set to OFF in the UTM start parameters. If this is not the case, the PERMIT screen is output in a mixed mode application.

3.1.2.2 Parametrizing the dialog

DRIVE parameters are assigned in UTM applications in the same way as in TIAM applications (see section 3.1.1, “Dialog structure in TIAM applications”, on page 14). In UTM applications, however, some DRIVE parameters are already defined by the system administrator when initiating the UTM application (see section 3.5.2.1, “Start procedure”, on page 73). Furthermore, user-specific DRIVE parameters can be transferred to DRIVE in DRIVE UTM applications by leader programs (see section “Data protection in TIAM applications” in the manual “DRIVE Programming System” [1]).

PARAMETER STATIC operands can only be assigned once.
It is immaterial whether the operand was assigned in the UTM start procedure, in a leader program, or interactively.

PARAMETER STATIC operands FIRSTPAGE and LASTPAGE can only be assigned as DRIVE start parameters in the UTM start procedure. You cannot change these operands in the DRIVE dialog.

There are PARAMETER operands, such as the PARAMETER DYNAMIC operands, which can be changed as required within the DRIVE session.

If you wish to query set PARAMETER values or reassign PARAMETER operands, enter the PARAMETER statement with the appropriate keyword; for example:

```
PARAMETER STATIC
```

The screen form now displayed differs from the PARAMETER STATIC screen form in TIAM applications, because different operands are significant here.

The operands are displayed as follows in the PARAMETER screen form:

– operand values which can be changed → high intensity

– operand values which cannot be changed → low intensity

Since only the operand values of PARAMETER STATIC must not be changed during a DRIVE run, only these are displayed at low intensity. The values appear at full intensity in the remaining PARAMETER screens.

The ON entry in a PARAMETER LOCK menu also appears at low intensity as a locked statement cannot be unlocked during a DRIVE run. All values in PARAMETER KFKEY menus appear at low intensity as K/F keys can only be assigned in the UTM start procedure.

3.1.2.3 Terminating the dialog

The dialog is terminated in two steps:

– terminating the dialog with DRIVE

– terminating the dialog with UTM

Terminating the dialog with DRIVE

You terminate the dialog with DRIVE with the STOP or EXIT statement. All resources used by DRIVE are released. Open transactions are handled in the following manner:

STOP The dialog can only be terminated with STOP if all open transactions are closed. You can terminate transactions with the COMMIT or ROLLBACK WORK statement.

Entering STOP without the keyword WITH in UTM applications results in a PEND FI. The conversation and the transaction are terminated.

EXIT The EXIT statement terminates the dialog with DRIVE even if a transaction is open. This transaction is rolled back.

Notes on terminating a dialog with STOP

In program mode, you may specify a follow-up TAC in the STOP statement:

```
STOP WITH followup-tac
```

followup-tac should be entered as a literal or variable in the program. It results in a PEND FC. The DRIVE conversation and the transaction are terminated; the dialog step is to continue in the chained conversation.

If you wish to terminate DRIVE with STOP and output a specific screen form at the same time, you can do so by modifying the STOP statement as follows:

```
STOP WITH DISPLAY screen-form
```

screen-form is the name of an FHS form (see “DRIVE Programming Language” [2]).

Use the following statement to output a form generated using DRIVE tools:

```
STOP WITH DISPLAY FORM form-name
```

How to generate screen forms (DRIVE and FHS) is described in “DRIVE Programming Language” [2].

After STOP without DISPLAY, UTM responds with the following message:

```
PLEASE ENTER TRANSACTION CODE:
```

Terminating the dialog with UTM

You sign off from the UTM application by entering the transaction code KDCOFF. The connection to the host is cleared down. A message is displayed on the screen indicating that the connection to the host has been cleared.

3.1.3 Calling DRIVE with BS2000 procedures

In TIAM applications, you can also initiate DRIVE within interactive or batch BS2000 procedures.

3.1.3.1 Interactive procedure

Processing sequences which recur at DRIVE initiation can be executed automatically. For this purpose, DRIVE can be called in an interactive BS2000 procedure.

An interactive BS2000 procedure can be used to

– assign module libraries

– call DRIVE

– define PARAMETER operands

– start DRIVE programs, etc.

If necessary, it is possible to control the procedure run using BS2000 procedure variables and check it on the screen.

The DEBUG statement in an interactive BS2000 procedure switches you to debugging mode (see the “DRIVE Directory” [3], DEBUG statement). You can now conduct a debugging session as normal. The DEBUG statement should be the last statement before the END-PROCEDURE statement, because DRIVE interactive mode is resumed after the debugging session is terminated and no further BS2000 statements are read from the procedure.

Debugging statements are not permitted in interactive BS2000 procedures.

Example

You wish to initiate the SESAM variant of DRIVE with an interactive BS2000 procedure.

Names in lowercase letters must be replaced by the names valid in the environment involved. If, for example, you wish to use subprograms in other programming languages, you must extend the procedure accordingly.

```
/BEGIN-PROC A
/SET-FILE-LINK LINK-NAME=DRIVEOML,-
/  FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,-
/  FILE-NAME=SYSPRG.DRIVE.022
```

```
/SET-FILE-LINK LINK-NAME=BLSLIB01,-
/  FILE-NAME=crte-lib
/SET-FILE-LINK LINK-NAME=BLSLIB02,-
/  FILE-NAME=lms-lib
/SET-FILE-LINK LINK-NAME=BLSLIB03,-
/  FILE-NAME=fhsmacro-lib
/SET-FILE-LINK LINK-NAME=BLSLIB04,-
/  FILE-NAME=systemmacro-lib
/SET-FILE-LINK LINK-NAME=SESAMOML,-
/  FILE-NAME=sesam-lib
/SET-FILE-LINK LINK-NAME=SESCONF,-
/  FILE-NAME=sesam-conf
/SET-FILE-LINK LINK-NAME=FORMOML,-
/  FILE-NAME=format-lib
/SET-FILE-LINK LINK-NAME=MROUTLIB,-
/  FILE-NAME=fhsrts-lib
/SET-FILE-LINK LINK-NAME=RSOML,-
/  FILE-NAME=SYSLIB.DRIVE.022
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROGRAM FROM-FILE=-
/  *MOD(LIB=objlib,ELEM=module-name,-
/  PROG-MO=ANY,RUN-MO=ADV(ALT-LIB=YES,-
/  NAME-COL=ABORT,UN-EXTRNS=DELAY,-
/  LO-IN=REF))
/END-PROC
```

Call generated DRIVE variant (LLM).
The LLM name and the library name were defined in the generation (ask your system administrator).

DRIVE now awaits entries at the terminal.

If DRIVE is to execute certain statements automatically, e.g. parameter assignment or invoking DRIVE programs, you can also enter these statements in the interactive BS2000 procedure. This is done after the START-PROG command without a slash, because these are not BS2000 commands. Any statement allowed in the DRIVE interactive mode can be entered here. If an interactive BS2000 procedure contains incorrect DRIVE statements, DRIVE outputs an appropriate error message.
DRIVE then awaits all further entries at the terminal.

If no errors are found, DRIVE reads all the entries in the interactive BS2000 procedure. When the last DRIVE statement from the procedure has been processed:

– DRIVE is terminated if the last statement in the procedure was STOP

– you can continue processing with DRIVE (no STOP).

3.1.3.2 Batch procedure

It is possible to automate long-running tasks and execute them as BS2000 batch tasks by calling DRIVE in a BS2000 batch procedure.

Example

The long-running DRIVE program MONTHSTAT is to run in the background. The name of the SESAM database, of the schema and of the authorization must be made known.

Names in lowercase must be replaced by the names valid in the environment involved.

```
/LOGON
/SET-FILE-LINK LINK-NAME=DRIVEOML,-
/  FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,-
/  FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,-
/  FILE-NAME=crte-lib
/SET-FILE-LINK LINK-NAME=BLSLIB02,-
/  FILE-NAME=lms-lib
/SET-FILE-LINK LINK-NAME=BLSLIB03,-
/  FILE-NAME=fhsmacro-lib
/SET-FILE-LINK LINK-NAME=BLSLIB04,-
/  FILE-NAME=systemmacro-lib
/SET-FILE-LINK LINK-NAME=SESAMOML,-
/  FILE-NAME=sesam-lib
/SET-FILE-LINK LINK-NAME=SESCONF,-
/  FILE-NAME=sesam-conf
/SET-FILE-LINK LINK-NAME=RSOML,-
/  FILE-NAME=SYSLIB.DRIVE.022
/ASSIGN-SYSDTA TO=*SYSCMD
/START-PROGRAM FROM-FILE=-
/  *MOD(LIB=obj-lib,ELEM=module-name,-
/  PROG-MO=ANY,RUN-MO=ADV(ALT-LIB=YES,-
/  NAME-COL=ABORT,UN-EXTRNS=DELAY,-
/  LO-IN=REF))
PARAMETER DYNAMIC LIBRARY=...
PARAMETER DYNAMIC SCHEMA=...
PARAMETER DYNAMIC CATALOG=...
PARAMETER DYNAMIC AUTHORIZATION=...
DO MONTHSTAT
STOP
/LOGOFF
```

Call generated DRIVE variant (LLM).
The LLM name and the library name were defined in the generation (ask your system administrator).

FHS forms must not be used here by DRIVE. They result in an abnormal program termination. All other outputs are written to SYSOUT.

Outputs to a printer (SYSLST) or file are possible. The output file must be assigned with the BS2000 batch procedure using the command /ASSIGN-SYSLST=*file*.

When END OF FILE (EOF) on SYSDTA is reached, the batch task is aborted.

When DRIVE is invoked in a BS2000 batch procedure which is to process a database, the corresponding DBH must be loaded. Otherwise the BS2000 batch procedure is placed in a wait state. In this case the BS2000 batch task must be aborted and restarted after the DBH has been loaded.

Input data can be read via SYSDTA. This is used in automated testing, for example.

3.2 Setting up DRIVE

(Changes to chapter 13 of “DRIVE Programming System” [1])

This chapter describes preparatory tasks for implementing DRIVE.

Module libraries must be assigned. In addition, a DRIVE library must be created and assigned. All other tasks are optional and need only be carried out if they are necessary for your particular environment.

This chapter is primarily intended for the DRIVE administrator. It is assumed that you have a basic knowledge of the BS2000 operating system as well as BS2000 address space management.

This chapter describes the activities involved in

– minimizing the amount of memory required by DRIVE (from page 29)

– assigning module libraries (from page 31)

– creating libraries for DRIVE programs, copy members, user labels, intermediate code and interpreter listings (from page 32)

– allocating components for logging DRIVE (from page 33)

– readying a list file for UTM applications (from page 34)

– readying a diagnostics file (from page 35)

– selecting the desired language (English or German) for the DRIVE dialog (from page 35).

3.2.1 Loading DRIVE modules as shared code

If DRIVE is loaded in the normal way and two or more DRIVE TIAM users are working in parallel or two or more DRIVE UTM tasks have been generated, there is a number of identical DRIVE modules in class 6 memory. This would result in an unnecessarily high load on class 6 memory.
To prevent this, you can load the DRIVE main module DRILLM22 as shared code in class 3/4 memory (subsystem name = DRIVE22).

Shared code is loaded – as a subsystem – with BS2000 Dynamic Subsystem Management (DSSM) (see “BS2000 System Installation” manual [17]).

In order to use the DRIVE22 subsystem, you must:

– enter the subsystem in the BS2000 subsystem catalog

– load (and unload) the subsystem

To use the shared code in **old-style operation**, the subsystem for old-style operation must be loaded (subsystem name = DRIVE).

To use the shared code in **mixed operation**, the subsystems for new-style operation and old-style operation must be loaded.

3.2.1.1 Entering subsystems in the subsystem catalog

The DRIVE22 subsystem is defined in an object file which was created with the SSCM (Static Subsystem Catalog Manager). This object file must be entered in the subsystem catalog, its name is SYSSSC.DRIVE.022.

Example

The object file SYSSSC.DRIVE.022 is added to the subsystem catalog “sskat” using the SSCM.

```
...
/START-SSCM
//START-CATALOG-MOIFICATION CATALOG-NAME=sskat
//ADD-CATALOG-ENTRY FROM-FILE=SYSSSC.DRIVE.022
//CHECK-CATALOG CATALOG-NAME=*CURRENT
//SAVE-CATALOG CATALOG-NAME=*CURRENT
//END
...
```

3.2.1.2 Loading and unloading subsystems

In order to load the DRIVE22 subsystem as shared code during the current BS2000 session, use the following BS2000 system administrator command /CREATE-SS:

```
/CREATE-SS SS-NAME=DRIVE22
```

Subsystems loaded as shared code can also be unloaded during the current BS2000 session. A prerequisite for this, however, is that no DRIVE-TIAM user is working with it and no DRIVE-UTM task is linked to the subsystem. To unload the subsystem, use the BS2000 system administrator command /DELETE-SS:

```
/DELETE-SS SS-NAME=DRIVE22
```

3.2.2 Assigning module libraries

The following table lists the module libraries and files required at runtime for a DRIVE session. Names in lowercase letters must be replaced by the names valid for the respective environment.

Library/file contains	**Name**	**Link name**
DRIVE modules*	SYSLNK.DRIVE.022	DRIVEOML
DRIVE system programs	SYSPRG.DRIVE.022	LIBOML
User-specific program library *	usr-lib	USEROML
SESAM modules	sesam-lib	SESAMOML
SESAM configuration file	sesam-conf	SESCONF
Library for report generator	SYSLIB.DRIVE.022	RSOML
User-specific FHS format library	format-lib	FORMOML
FHS modules	fhslrts-lib	MROUTLIB
Runtime libraries to resolve open external references by the dynamic load linkage editor	crte-lib	BLSLIB01
	lms-lib	BLSLIB02
	fhsmacro-lib	BLSLIB03
	systemmacro-lib e.g. $TSOS.SYSLIB.TIAM.xxx	BLSLIB04

DRIVE dynamically loads the modules required. For this purpose, DRIVE searches the following module libraries in the order indicated. Modules that are loaded dynamically are marked with an “*”.

Search sequence:

1. Module library with file link names
2. Module library assigned as object module library using /SET-TASKLIB... (default assignment: $TSOS.TASKLIB)
3. Library SYSLNK.DRIVE.022 under the user ID under which DRIVE was started
4. Library SYSLNK.DRIVE.022 of the BS2000 default user ID
5. TASKLIB of the BS2000 default user ID

Search sequence when accessing DRIVE system programs:

1. Module library with the link name LIBOML
2. SYSPRG.DRIVE.022 under the user ID under which DRIVE was started

The dynamic loading of non-DRIVE modules (SESAM and FHS modules) is subject to the link behavior of the particular product.

3.2.3 Creating a DRIVE library

DRIVE programs, copy members, user labels, intermediate code and compiler listings are stored and managed as members of DRIVE libraries. For this purpose, a PLAM library is used which is assigned as the DRIVE library (see “DRIVE Programming System” [1], chapter “Managing DRIVE members in PLAM libraries“).

Creating a PLAM library

A PLAM library can be created by using the software product LIBRARY MAINTENANCE SYSTEM (LMS), see the “LMS” manual [15]), e.g.:

```
/START-PROGRAM $LMS
$LIB DRI.PLAM, NEW
$END
```

LMS creates a PLAM library under the name DRI.PLAM.

Assigning a DRIVE library

There are two ways of assigning a PLAM library as the DRIVE library:

– using the DRIVE statement PARAMETER DYNAMIC LIBRARY=...
– using the file assignment /SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=...

If you assign a non-existent PLAM library as the DRIVE library with PARAMETER DYNAMIC LIBRARY=..., DRIVE outputs an error message.

3.2.4 Readying components for logging dialogs

If desired, DRIVE logs inputs and outputs during a DRIVE session in a log file. DRIVE logging is performed by the utility routine SYSPRG.DRIVE.022.DRILOG. The program writes the data to be logged to the relevant log file.

SYSPRG.DRIVE.022.DRILOG is started as a BS2000 batch task. It is loaded only once, irrespective of the number of TIAM users working with DRIVE in parallel or the number of DRIVE UTM tasks generated.

Required files

The following files are supplied for DRIVE logging:

SYSENT.DRIVE.022.DRILOG
: This file contains the following BS2000 batch procedure:

```
/LOGON
/START-PROGRAM SYSPRG.DRIVE.022.DRILOG
/LOGOFF
```

The SYSPRG.DRIVE.022.DRILOG utility routine is started as a batch tasks by means of this BS2000 batch procedure.

SYSPRG.DRIVE.022.DRILOG
: Logging routine.

SYSPRG.DRIVE.022.DRILOGP
: Routine for editing and printing the log file.

SYSPRG.DRIVE.022.DRIENDE
: Routine for terminating SYSPRG.DRIVE.022.DRILOG.

In order to log the DRIVE dialog in UTM applications, you must make these files available under the user ID under which the DRIVE UTM application is started.

Starting the SYSPRG.DRIVE.022.DRILOG utility routine

There are two ways of starting SYSPRG.DRIVE.022.DRILOG as a BS2000 batch task:

– Using the BS2000 command /ENTER-JOB SYSENT.DRIVE.022.DRILOG (not possible for DRIVE UTM)

– Starting the batch task by activating DRIVE logging via the DRIVE statement PARAMETER DYNAMIC (see “DRIVE Programming System” [1], chapter “Logging the DRIVE dialog“).

Prerequisite:

The file SYSENT.DRIVE.022.DRILOG must be available under the user ID under which DRIVE was called (TIAM) or under which the DRIVE UTM application was started (UTM).

If the message `DRILOG NICHT GELADEN` (= DRILOG not loaded) is output after logging has been activated, the batch task could not be started, in which case no logging is performed.

Terminating the SYSPRG.DRIVE.022.DRILOG utility routine

The batch task SYSENT.DRIVE.022.DRILOG can be terminated

– using the SYSPRG.DRIVE.022.DRIENDE routine:
`/START-PROGRAM SYSPRG.DRIVE.022.DRIENDE`

– by the operator following the message
`DRILOG KEIN ANWENDER MEHR? DRILOGP NUR BEENDEN MIT /INTR tsn,STOP`
`(No more DRILOG users? Terminate DRILOGP only with /INTR tsn,STOP)`

3.2.5 Readying a list file for UTM applications

All print outputs generated in UTM applications are buffered in the list file. You can create this list file with the following characteristics:

```
LINK-NAME       = DRILIST
ACCESS-METHOD   = ISAM
RECORD-FORMAT   = V
BUFFER-LENGTH   = STD(SIZE=b)    (b must not exceed 16)
SPACE           = REL(nn,nn)
KEY-POSITION    = 5
KEY-LENGTH      = 24
```

If it finds no file with the file link name DRILIST, DRIVE creates a list file, if needed, under the name DRI.LIST.FILE with the specified file characteristics and BUFFER-LENGTH = STD(SIZE=16), SPACE = REL(33,16).

3.2.6 Readying a diagnostics (INTTRACE) file

The print outputs generated by parameter DIAGNOSIS (see “DRIVE Directory” [3], PARAMETER statement) are written to the INTTRACE file.

You can create an INTTRACE file with the following file characteristics:

```
LINK-NAME       = INTTRACE
ACCESS-METHOD   = ISAM
RECORD-FORMAT   = V
BUFFER-LENGTH   = STD(SIZE=16)
SPACE           = REL(33,16)
KEY-POSITION    = 5
KEY-LENGTH      = 32
OPEN-MODE       = INOUT
SHARED-UPDATE   = YES
```

If it finds no file with file link name INTTRACE, DRIVE creates a file, if necessary, under the name DRI.INTTRACE.FILE with the specified file characteristics and BUFFER-LENGTH = STD(SIZE=16), SPACE = REL(33,16).

3.2.7 Language option for the DRIVE dialog

You determine the language used for DRIVE/WINDOW output.

By default, the DRIVE messages are output in English.

If you want to convert the DRIVE messages to German, enter the following command in BS2000 system mode:

```
/MODIFY-MSG-ATTRIBUTES,TASK-LANGUAGE=D
```

3.3 Generating DRIVE for TIAM applications

(Changes to chapter 14 of “DRIVE Programming System” [1])

The procedure DRIPRC.INSTALL.DRIVE in the PLAM library SYSPRC.DRIVE.022 is used for generating DRIVE. Depending on the parameter values to be entered, this BS2000 procedure links the DRIVE modules required for the respective DRIVE variant to form a linking loader module (LLM).

Call the procedure using the following BS2000 command:

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)
```

Enter the following parameter values in the procedure run to generate a DRIVE variant for TIAM applications:

`&BETRIEB = TIAM`

`&STYLE = NEW`

`&OBJLIB = obj-lib` For *obj-lib* specify the module library in which the linkage editor is to store the LLM *module-name*. This module library must not be the library SYSLNK.DRIVE.022.

`&EDTLIB = edt-lib` For *edt-lib* specify the module library containing the EDT object module IEDTGLE.

`&DRIVELIB = drive-lib` For *drive-lib* specify the module library with the DRIVE modules.

`&BINDER = linkage-ed` For *linkage-ed* specify the file name of the linkage editor.

`&STARTLLM = module-name`
For *module-name* specify the name of the LLM to which the generated DRIVE variant is linked.
This name must be specified at startup in the /START-PROG command (see section 3.1.3.1, “Interactive procedure”, on page 25 and section 3.1.3.2, “Batch procedure”, on page 27).

module-name can be up to 32 characters long.

`&SESAMLIB = sesam-lib`
For *sesam-lib* specify the module library with the SESAM modules.

Example

Generating a DRIVE new-style variant for accessing SESAM.

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)-
/  PROC-PARAM=(STYLE=NEW,BETRIEB=TIAM,OBJLIB=MYOBJLIB.DRIVE22,-
/               DRIVELIB=SYSLNK.DRIVE.022,-
/               BINDER=$TSOS.BINDER,STARTLLM=DRISES,-
/               SESAMLIB=$TSOS.SYSLNK.SESAMSQL.022)
```

Mixed operation

To generate a DRIVE variant for mixed operation, you also call the DRIPRC.INSTALL.DRIVE procedure (see page 36).

In this case, specify the same parameters as for the new style variant described above.

3.3.1 Special characteristics of old-style operation

To generate a DRIVE variant for old-style operation, you also call the DRIPRC.INSTALL.DRIVE procedure (see page 36).

The only difference is that you must specify the following parameter values for the &STYLE and &FASSUNG parameters:

`&STYLE = OLD`

`&FASSUNG = [ SESAM | LEASY | DMS ]`
Enter the database system you access with DRIVE.

In addition, the parameters &PHASE, &TSOSLNK and possibly &LEASYLIB must be specified:

`&PHASE = module-name` For *module-name* specify the name of the module to which the generated DRIVE variant is linked.
This name must be specified at startup in the /START-PROG command (see section 3.1.3.1, “Interactive procedure”, on page 25 and section 3.1.3.2, “Batch procedure”, on page 27).

`&TSOSLNK = tsoslnk-name`
For *tsoslnk-name* specify the file name of the static linkage editor TSOSLNK.
The default name is $TSOS.TSOSLNK.

`&LEASYLIB = leasy-lib` For *leasy-lib* specify the module library with the LEASY modules.
Only necessary if &FASSUNG=LEASY.
The default name is $TSOS.LEA.OML.

The remaining parameters are described on page 36. The parameters &BINDER, &STARTLLM and &OBJLIB are not applicable.

Example

Generating a DRIVE variant for accessing LEASY.

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)-
/  PROC-PARAM=(STYLE=OLD,BETRIEB=TIAM,FASSUNG=LEASY,-
/               DRIVELIB=SYSLNK.DRIVE.022,PHASE=DRILEA,-
/               TSOSLNK=$TSOS.TSOSLNK,-
/               LEASYLIB=$TSOS.LEA.OML)
```

3.4 Generating DRIVE for UTM applications

(Changes to chapter 15 of “DRIVE Programming System” [1])

This chapter describes

– the general background to generating DRIVE for UTM applications (as of page 39)

– how to prepare the generation for UTM applications (as of page 41)

– how to generate DRIVE for UTM applications (as of page 51)

– the generation for mixed operation (as of page 52)

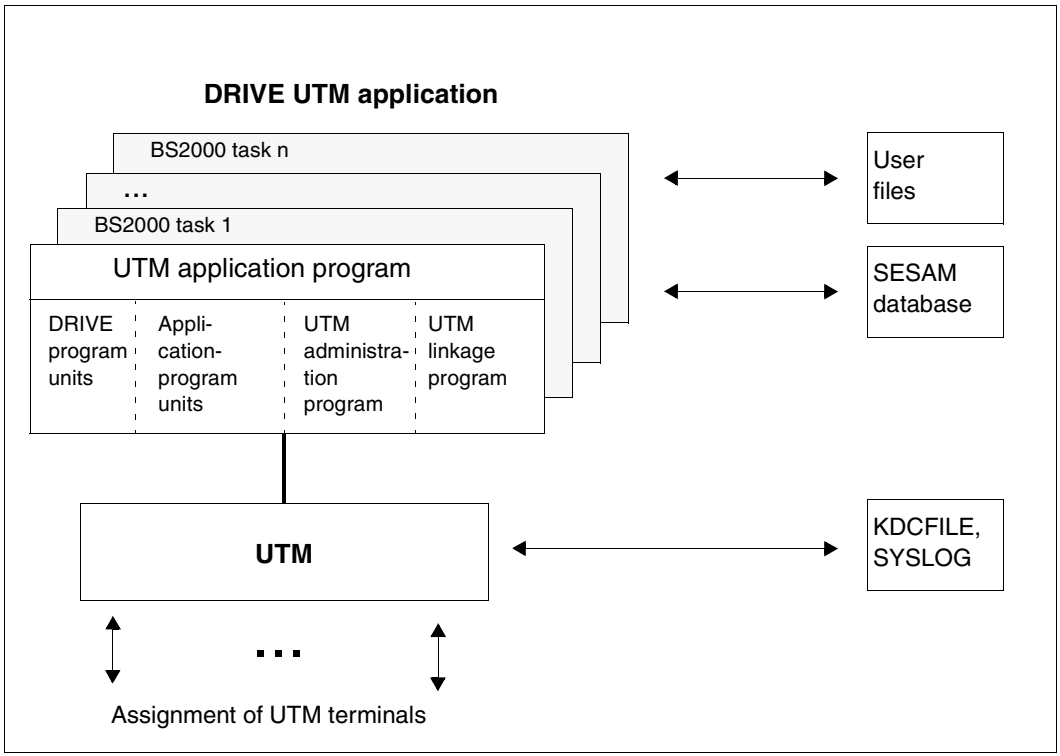
– the generation for old-style operation (as of page 57)

– the generation for distributed transaction processing (as of page 62)

To generate DRIVE for UTM applications, you must create a UTM application. The components of a UTM application, with the exception of the UTM linkage program (KDCROOT) and the application configuration (KDCFILE) are thus supplied. KDCROOT and KDCFILE must be generated.

You also have the option of integrating DRIVE into an (existing) UTM application with user program units (see section 3.4.2.2, “Integrating DRIVE in an existing UTM application”, on page 49).

The figure below shows a schematic of a UTM application with DRIVE:



A UTM application comprises:

– the definition of the application configuration (KDCFILE) and
– the UTM application program.

The UTM application program consists of:

– The UTM linkage program (KDCROOT). This is the main program to which the program units are linked as subprograms. The linkage procedure produces the load module of the UTM application program.
– The DRIVE program unit DRIVROOT.
– The exits DRIVVORG, EXSTRT and EXSHUT.
– The UTM administration program KDCADM for synchronous and asynchronous administration.

Steps in the generation of a DRIVE UTM application

Normally, the generation of a UTM application with DRIVE requires the following steps:

1. Prepare the use of user exits by means of the ALLEX macro (see section 3.4.1 on page 41).
2. Create the application configuration (KDCFILE) and the UTM linkage program (KDCROOT) (see section 3.4.2 on page 43).
3. Assemble the UTM linkage program (see section 3.4.3 on page 50).

   Steps (2) and (3) can be performed using a generation procedure supplied with UTM.
4. Generate a UTM application (see section 3.4.4 on page 51).

   SESAM databases can be edited in UTM applications. You must therefore integrate the appropriate DRIVE variant in the UTM application program.

   This step is completed using a procedure supplied by DRIVE.

3.4.1 Integrating user exits

When DRIVE is executed under UTM, DRIVE provides two exits:

– the start exit

– the shut exit

Apart from these two standard DRIVE exits, additional user-specific start or shut exits may exist. They are called by the corresponding DRIVE exits when these have been processed. The standard module for the DRIVE exits is in the module library SYSLNK.DRIVE.022.

User exits can be integrated using the ALLEX macro.

The source of the ALLEX macro is in the library SIPLIB.DRIVE.022.

Structure of the ALLEX macro

```
[name]   ALLEX START=(DRIVSTRT,module,...),
              SHUT= (DRIVSHUT,module,...)
         END
```

Explanation:

name	Any module name; may be omitted
module	Name of the relevant user exit module; maximum number: 9 The DRIVE exits DRIVSTRT and DRIVSHUT are mandatory. The exits must be specified in the order indicated.

Example

The following procedure assembles the ALLEX macro.

```
/BEGIN-PROC
/DELETE-SYSTEM-FILE OMF
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=ALTLIB,FILE-NAME=SIPLIB.DRIVE.022 ————————————————— (1)
/START-PROG $ASSEMBHC
*COMOPT ALTLIB
                    ————————————————————————————————————————————————————— (2)
*COMOPT SOURCE=ALLEX
*END HALT
/START-PROG $LMS
$LIB FILE=SYSLNK.DRIVE.022
                    ————————————————————————————————————————————————————— (3)
$ADDR *OMF
$END
/END-PROC
```

Explanation:

(1) Prior to assembly, the macro library containing the ALLEX macro must be assigned: SIPLIB.DRIVE.022.

(2) Assemble the ALLEX macro.

The modules MOD1 and MOD2 contain the user exit routines for the start and shut exits.
Macro calls begin at column 10, and their parameters at column 16.

(3) Enter the object module in module library SYSLNK.DRIVE.022.

The ALLEX macro generates an ENTRY EXTAB which must not appear in other application program units.

The order in which the exit routines are entered for START must match that of the start parameters in the start procedure for the UTM application.

3.4.2 Generating the application configuration and the UTM linkage program

The application configuration (KDCFILE) contains information on:

– application characteristics
– user identifications and access protection
– characteristics of data terminals
– characteristics of transaction codes

KDCFILE is stored in a file with the name *base-name.KDCA*. If required, it is maintained in duplicate, i.e. additionally in file *base-name.KDCB*.
If required, KDCFILE can also be split and distributed over two or more disks (see "*open*UTM Generating and Handling Applications" [13]).

The UTM linkage program (KDCROOT) is the part of the UTM application program which creates the link between UTM and the program units.

The UTM utility routine KDCDEF defines the application configuration (KDCFILE) and generates the KDCROOT source program, which must be assembled and linked.

The generation procedure SYSPRC.UTM.xxx(GEN) supplied with UTM starts the KDCDEF utility routine. Here, xxx means the version of UTM, e.g. 040 or 050. KDCDEF generates the KDCFILE and KDCROOT source program, and assembles the KDCROOT source program (see "*open*UTM Generating and Handling Applications" [13]).

In order to control KDCDEF, the generation procedure SYSPRC.UTM.xxx(GEN) requires a file containing KDCDEF control statements (KDCDEF input file). The DRIKDCDEF.* elements, which are part of the DRIVE product, contain DRIVE-specific skeletons with KDCDEF control statements and are located in the library SYSPRC.DRIVE.022. One of these elements can be used as the KDCDEF input file once user-defined KDCDEF control statements have been added. You will find a sample statement sequence in the "Example" on page 47.

3.4.2.1 KDCDEF control statements

The control statements for the KDCDEF utility routine are described in "*open*UTM Generating and Handling Applications" [13].

The following must be specified for a DRIVE UTM application:

MAX

`MAX TASKS ≥ 2`	If a task is not available, another task can be used for administration
`MAX ASYNTASKS ≥ 1`	Asynchronous processing is permitted
`MAX KB ≥ 512`	Length of the communication area
`MAX SPAB = 32767`	Maximum length of the SPAB

`MAX NB = 32700` Maximum length of the message area

`MAX TRMSGLTH = 32700` Maximum message length

`MAX LSSBS ≥ 200` Maximum number of LSSBs (depending on application)

`MAX RECBUF ≥ (30,4096)`
Size of the restart area in PAM pages and the length of the storage areas in bytes per task

`MAX PGPOOL ≥ (1000,80,95)`
Size of the page pool in PAM pages; first warning at 80% level, second warning at 95% level

`MAX VGMSIZE ≥ 128` Size of the buffer area for the activity memory of SESAM V2 in Kbytes

DATABASE

The following must be specified for the database system SESAM V2.x:

```
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=sesam-lib
```

PROGRAM

The following DRIVE-specific program units must be specified:

```
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT  ,COMP=ILCS
PROGRAM EXSHUT  ,COMP=ILCS
```

As must the UTM-specific program unit:

```
PROGRAM KDCADM,COMP=ILCS
```

MODULE

```
MODULE EXTAB   ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

EXIT

```
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
```

TAC

The following DRIVE-specific transaction codes must be specified:

```
TAC DRISQL  ,TYPE=D,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG
TAC DRISQLF ,TYPE=D,STATUS=ON,CALL=NEXT ,PROGRAM=DRIVROOT,
TAC SQLNEXT ,TYPE=D,STATUS=ON,CALL=NEXT ,PROGRAM=DRIVROOT,
TAC SQLENTER,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG
TAC SQLLIST ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG
TAC SQLRET,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIVROOT,TIME=300000
```

TAC name	Function
DRISQL	Start of processing with DRIVE. User-specific TACs may be used instead of DRISQL.
DRISQLF	Start of processing with DRIVE via PEND PA. User-specific TACs may be used instead of DRISQLF.
SQLNEXT	Call of DRIVE interactive mode
SQLENTER	Asynchronous operation of DRIVE
SQLLIST	Asynchronous conversation to output user-specific list records
SQLRET	Return to the DRIVE submitting partner after calling a user-specific UTM program unit in C/COBOL in the same application (CALL TAC). This TAC need only be specified if a CALL TAC statement is processed locally in the DRIVE program; see also KDCDEF frame DRIKDCDEF.CALL.LOCAL.TAC in the SYSPRC.DRIVE.022 library.

The following transaction codes must be specified for the synchronous administration of UTM:

```
TAC KDCTAC  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCLTERM,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCPTERM,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCSWTCH,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCUSER ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCSEND ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCAPPL ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCDIAG ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCLOG  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCINF  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCHELP ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCSHUT ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
TAC KDCTCL  ,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM,
```

The following transaction codes must be specified for the asynchronous administration of UTM:

```
TAC KDCTACA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCLTRMA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCPTRMA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCSWCHA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCUSERA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCSENDA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCAPPLA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCDIAGA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCLOGA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCINFA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCHELPA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCSHUTA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
TAC KDCTCLA ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=KDCADM,ADMIN=YES,DBKEY=UTM
```

SFUNC

`SFUNC key,RET=return-code`

The return code key for PARAMETER KFKEY must agree with the corresponding SFUNC specification,
e.g. `SFUNC K1,RET=2OZ`

USER

`USER user-name[,STATUS=ADMIN]`

At least one user with administration authorization must exist.

PTERM

`PTERM pterm-name,PRONAM=processor-name,PTYPE=partner-type,LTERM=lterm-name`

The names of the PTERM parameters must be taken from the PDN generation.
All terminal types supported by VTSU and FHS may be used.

LTERM

`LTERM lterm-name` The LTERM name must be identical with LTERM name of the PTERM statement.

END

The name of the file with the KDCDEF statements (KDCDEF input file) must be transferred for processing to the KDCDEF program with OPTION DATA=file (see "*open*UTM Generating and Handling Applications" [13]).

The KDCDEF control statements for incorporating DRIVE into an **existing** UTM application are described in the section “Integrating DRIVE in an existing UTM application” on page 49.

The KDCDEF control statements that you must specify when the DRIVE-COMP compiler is implemented, are described in the “DRIVE Compiler” manual [5].

The DRIVE product is supplied with DRIKDCDEF.* skeletons which contain KDCDEF control statements and are located in the library SYSPRC.DRIVE.022. These must be adapted by the user to suit the actual conditions.

Example

This section contains an example of the KDCDEF control statements required when DRIVE is to access a SESAM V2.x database (new style) in UTM applications.

```
REM
REM ******** SET MAXIMUM VALUES ************************************
REM
MAX TASKS=6,ASYNTASKS=1
MAX KB=512,SPAB=32767,NB=32700
MAX VGMSIZE=128
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700
MAX PGPOOL=(1000,80,95),RECBUF=(30,4096),REQNR=8
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60,CONRTIME=10
MAX LOGACKWAIT=600,BRETRYNR=10
REM
REM ************ DEFINE DATABASE ***************************************
REM
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=sesam-lib
REM
REM ************ DEFINE PROGRAM UNITS **********************************
REM
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM KDCADM,COMP=ILCS
REM
REM ********* LOAD DRIVE MODULES ***************************************
REM
MODULE EXTAB   ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
REM
REM ************** DEFINE USER EXITS **********************************
```

```
REM
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM *********** TRANSACTION CODES FOR DRIVE ****************************
REM
DEFAULT TAC PROGRAM=DRIVROOT,STATUS=ON,
TAC DRISQL,TYPE=D  ,CALL=FIRST,EXIT=DRIVVORG
TAC DRISQLF,TYPE=D ,CALL=NEXT
TAC SQLNEXT,TYPE=D ,CALL=NEXT
TAC SQLENTER,TYPE=A,CALL=FIRST,EXIT=DRIVVORG
TAC SQLLIST ,TYPE=A,CALL=FIRST,EXIT=DRIVVORG
REM
REM *********** SYNCHRONOUS ADMINISTRATION ****************************
REM
DEFAULT TAC ADMIN=Y,PROGRAM=KDCADM,TYPE=D,STATUS=ON,CALL=BOTH,DBKEY=UTM
TAC KDCTAC
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCUSER
TAC KDCSEND
TAC KDCAPPL
TAC KDCDIAG
TAC KDCLOG
TAC KDCINF
TAC KDCHELP
TAC KDCSHUT
TAC KDCTCL
REM
REM *********** ASYNCHRONOUS ADMINISTRATION ****************************
REM
DEFAULT TAC ADMIN=Y,PROGRAM=KDCADM,TYPE=A,STATUS=ON,CALL=FIRST,DBKEY=UTM
TAC KDCTACA
TAC KDCLTRMA
TAC KDCPTRMA
TAC KDCSWCHA
TAC KDCUSERA
TAC KDCSENDA
TAC KDCAPPLA
TAC KDCDIAGA
TAC KDCLOGA
TAC KDCINFA
TAC KDCHELPA
TAC KDCSHUTA
TAC KDCTCLA
```

```
REM
REM *********** ASSIGNMENT OF FUNCTION KEYS ***************************
REM
SFUNC K1,RET=2OZ
SFUNC F1,CMD=KDCOFF
SFUNC K2,RET=KDCOUT
REM
REM *********** PERMITTED USERS ****************************************
REM
USER user1
USER user2
USER user3
USER adm     ,STATUS=ON,PERMIT=ADMIN
USER adm1    ,STATUS=ON,PERMIT=ADMIN
USER adm2    ,STATUS=ON,PERMIT=ADMIN
USER admin   ,STATUS=ON,PERMIT=ADMIN
REM
REM *********** DEFINE PTERMS AND LTERMS *******************************
REM
PTERM xxxxxxxx,PRONAM=xxxxxx,PTYPE=T9750,LTERM=driterm1
LTERM driterm1
PTERM xxxxxxxx,PRONAM=xxxxxx,PTYPE=T9750,LTERM=driterm2
LTERM driterm2
PTERM xxxxxxxx,PRONAM=xxxxxx,PTYPE=T9750,LTERM=driterm3
LTERM driterm3
END
```

3.4.2.2 Integrating DRIVE in an existing UTM application

Modify or extend the MAX, PROGRAM, MODULE, EXIT, TAC and SFUNC statements in the input file for controlling the UTM utility routine KDCDEF. The required specifications are described in section 3.4.2.1, “KDCDEF control statements”, on page 43.

3.4.3 Assembling the UTM linkage program

There are two ways of assembling the UTM linkage program: with or without the support of SYSPRC.UTM.xxx(GEN) (xxx=version of UTM, e.g. 040).

– Assembly controlled by SYSPRC.UTM.xxx(GEN)

  The example below is an excerpt from the SYSPRC.UTM.040(GEN) installation procedure (see "*open*UTM Generating and Handling Applications" [13]). The message: ASSEMBLING KDCROOT ? is displayed. If you respond with Y, the KDCROOT file is assembled. A second ALTLIB assignment (ALTLIB2) to the library containing the KDCDB macro for the corresponding database system must also be included.

```
...
/REMARK TEXT=&ASSEMB= Y/N  ASSEMBLING KDCROOT?
/SKIP-COMMANDS TO-LABEL=RT&ASSEMB
/.RTY REMARK
/DELETE-FILE FILE-NAME=SYSLST.KDCROOT
/SET-JOB-STEP
/DELETE-SYSTEM-FILE FILE-NAME=OMF
/SET-JOB-STEP
/ASSIGN-SYSLST TO-FILE=SYSLST.KDCROOT
/SET-FILE-LINK LINK-NAME=ALTLIB,FILE-NAME=SYSLIB.UTM.040.ASS-
/                               ,ACCESS-METHOD=BY-CATALOG
/SET-FILE-LINK LINK-NAME=ALTLIB2,FILE-NAME=kdcdb-lib
/START-PROGRAM FROM-FILE=$ASSEMBHC
*COMOPT FLGLST,ATXREF,ALTLIB,ALTLIB2
*COMOPT SOURCE=ROOT.SRC.ASSEMB.&ROOTELEM
*END HALT
...
```

– Assembly without the support of SYSPRC.UTM.xxx(GEN)

```
/BEGIN-PROC C,YES,(&SRC),c'&' ------------------------------------------------ (1)
/DELETE-SYSTEM-FILE FILE-NAME=OMF
/SET-FILE-LINK LINK-NAME=ALTLIB,-
/    FILE-NAME=SYSLIB.UTM.040.ASS,-
/    ACCESS-METHOD=BY-CATALOG                  ------------------------------- (2)

/SET-FILE-LINK ALTLIB,SYSLIB.UTM.040.ASS
/SET-FILE-LINK LINK-NAME=ALTLIB2,FILE-NAME=kdcdb-lib ------------------------- (3)
/PARAMETER ALTLIB=YES
/ASSIGN-SYSDTA *SYSCMD
/START-PROGRAM $ASSEMBHC
*COMOPT SOURCE=&SRC
*COMOPT MODULE=root-lib-name  ------------------------------------------------ (4)
*COMOPT ALTLIB2
*END HALT
```

```
/SET-JOB-STEP
/ASSIGN-SYSDTA *PRIMARY
/END-PROC
```

Explanation:

(1) The variable &SRC must be supplied with the name of the KDCROOT file.

(2) The UTM macros are made available in the SYSLIB.UTM.040.ASS library.

(3) Specification of the library containing the KDCDB macro.

(4) The source program from file &SRC is assembled.

The object module created by the assembly is stored in module library root-lib-name.

3.4.4 Generating UTM applications

You generate a UTM application by linking a DRIVE variant to the UTM linkage program.

The DRIPRC.INSTALL.DRIVE procedure in the PLAM library SYSPRC.DRIVE.022 is used to generate the desired DRIVE variant. Depending on the parameter values to be entered, this BS2000 procedure links the DRIVE modules required for the respective DRIVE variant with the UTM linkage program to form a linking loader module (LLM).

Call the procedure using the following BS2000 command:

```
/CALL-PROCEDURE SYSPRC.DRIVE.022(DRIPRC.INSTALL.DRIVE)
```

To generate a DRIVE variant for UTM applications, enter the following parameter values in the procedure run:

```
&STYLE = NEW
```

```
&BETRIEB = UTM
```

```
&ROOTELEM = root-name
```

root-name is the compiled KDCROOT table module. Specify the CSECT name of the KDCROOT table module which was defined with the &ROOTELEM parameter in the UTM generation procedure SYSPRC.UTM.xxx(GEN) or with the KDCDEF control statement ROOT root-name (see “*open*UTM Generating and Handling Applications” [13]).

`&ROOTLIB = root-lib-name`
Specify the library which contains the ROOT member and in which the linkage editor is to store the LLM *module-name*.

`&DRIVELIB = drive-lib` For *drive-lib* specify the module library with the DRIVE modules. The default library name is SYSLNK.DRIVE.022.

`&BINDER = linkage-ed` For *linkage-ed* specify the file name of the linkage editor. The default name for the linkage editor is $TSOS.BINDER.

`&EDTLIB = edt-lib` For *edt-lib* specify the module library containing the EDT object module IEDTGLE. The default library name is $TSOS.EDTLIB.

`&STARTLLM = module-name`
For *module-name* specify the name of the LLM to which the UTM application is linked.
This is the name you use to start the UTM application (see the examples in section 3.5.2.1, “Start procedure”, on page 73).

`&UTMLIB = utm-lib` For *utm-lib* specify the UTM module library. The default library name is SYSLNK.UTM.xxx.

`&UTMSPLLIB = splrts-lib`
For *splrts-lib* specify the SPLRTS library of UTM. The default library name is SYSLNK.UTM.xxx.SPLRTS.

3.4.5 Linking DRIVE to an existing UTM application

The DRIVE-specific KDCDEF control statements must be added to the KDCDEF input file, see page 54.

The DRIVE-specific link statements must be incorporated into the user-specific link procedure. The SYSPRC.DRIVE.022 library contains appropriate framework files for this purpose:

– DRIPRC.INSTALL.RAHMEN.SESAM for new-style DRIVE

– DRIPRC.INSTALL.RAHMEN.MIX for mixed-operation DRIVE

3.4.6 Special characteristics of mixed operation

The procedure for generating a UTM application with DRIVE for mixed operation is essentially the same as the generation described for new-style operation. Therefore, only the differences are listed here. A detailed description can be found on pages 41 through 51.

3.4.6.1 Integrating user exits

DRIVE offers two start exits, two shut exits, and one form exit for implementing DRIVE in mixed operation under UTM.

On top of these DRIVE exits, there are additional user-specific start or shut exits. These are called by the respective DRIVE exits as soon as the DRIVE exits have been processed.

A standard module for the DRIVE exits can be found in the module library SYSLNK.DRIVE.022.

The SIPLIB.DRIVE.022 library contains the standard module EXTAB.MIXSQLLEA for mixed operation with SESAM V2/LEASY, and the standard module EXTAB.MIXSQLDMS for mixed operation with SESAM V2/DMS. The appropriate module must be copied to the SYSLNK.DRIVE.022 library under the name EXTAB.

User exits can be linked with the ALLEX macro.

The source of the ALLEX macro is in the SIPLIB.DRIVE.022 library.

Structure of the ALLEX macro

```
[name]  ALLEX START=(DRIVSTRT,exit,module,...),
              SHUT =(DRIVSHUT,DRISHUT,module,...),
              FORM =(DRIFORM,module,...)
        END
```

Explanation:

name	Any module name; may be omitted.
exit	Specify the following for exit: DRISTARD for DMS variant DRISTARL for LEASY variant DRISTARS for SESAM V2 variant
module	Name of the respective user exit module; maximum number: 8 The DRIVE exits DRIVSTRT, DRIVSHUT, DRISHUT and DRIFORM are mandatory. DRISTARD, DRISTARL or DRISTARS must also be entered. The exits must be specified in the order indicated.

3.4.6.2 KDCDEF control statements

The KDCDEF control statements for DRIVE in mixed operation are essentially the same as the KDCDEF control statements for new-style operation. Therefore, only the differences are listed here. A description of the KDCDEF control statements for new-style operation can be found in section 3.4.2.1, “KDCDEF control statements”, on page 43.

Modify or extend the following statements in the input file to control the UTM utility routine KDCDEF:

MAX

The following values are required for mixed operation:

```
KB ≥ 512
SPAB = 32767
NB = 32700
TRMSGLTH = 32700
LSSBS ≥ 200
MAX VGMSIZE ≥ 128
```

DATABASE

In addition to SESAM V2, it is also possible to access LEASY and DMS in mixed operation.

The following must be specified for the database system SESAM V2:

```
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=sesam-lib
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=sesam-lib
```

The following must be specified for LEASY:

```
DATABASE TYPE=LEASY,ENTRY=LEASY
```

The DATABASE statement is not applicable for DMS.

PROGRAM

The following PROGRAM statements must also be specified:

```
PROGRAM DRIKROOT,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM DRIVORG,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM EXFORM,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

MODULE

The following MODULE statements must also be specified:

```
MODULE DRIFORM,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRISHUT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIEXT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2INT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2LINK,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIC51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2REF,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

DRIDUM51 is omitted.

The following must be added for the database system SESAM V2.x:

```
MODULE DRISTARS,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SESORT,LIB=sesam-lib,LOAD=STATIC
```

The following must be added for DMS:

```
MODULE DRISTARD,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

The following must be added for LEASY:

```
MODULE DRISTARL,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

DRIC51 is omitted for LEASY.

EXIT

The following EXIT statement must be added:

```
EXIT PROGRAM=EXFORM,USAGE=FORMAT
```

TAC

The following TAC statements must be added:

For new style:

```
TAC DRISQLM,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIVROOT
```

For old style:

```
TAC DRIVE    ,TYPE=D,STATUS=ON,CALL=FIRST,PROGRAM=DRIKROOT,EXIT=DRIVORG
TAC DRISES   ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRISEQ   ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRINEXT  ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT,TIME=300000
TAC DRIRTP ,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRISQL51,TYPE=D,STATUS=ON,CALL=NEXT,PROGRAM=DRIKROOT
TAC DRIENTER,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIKROOT,TIME=300000,EXIT=DRIVORG
TAC DRILIST  ,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIKROOT,TIME=300000,EXIT=DRIVORG
```

SFUNC

```
SFUNC K1,RET=2OZ
```

3.4.6.3 Generating a UTM application

To generate a UTM application for mixed operation, call the BS2000 procedure DRIPRC.INSTALL.DRIVE (see page 51).

The only difference is that you must specify the following parameter value for the &FASSUNG parameter:

```
&FASSUNG = MIX
```

The remaining parameters are described on page 51. For a description of how to link DRIVE to an existing UTM application, see page 52.

3.4.7 Special characteristics of old-style operation

The procedure for generating a UTM application with DRIVE for old-style operation is essentially the same as the generation described for new-style operation. Therefore, only the differences are listed here. A detailed description can be found on pages 41 through 51.

3.4.7.1 Integrating user exits

The following three exits are available for DRIVE UTM processing under old-style operation: START, SHUT and FORMAT exits.

User-own start, form and shut exits are permitted in addition to the standard exits. They are called by the DRIVE exits as soon as the corresponding DRIVE exits have been processed. The module library SYSLNK.DRIVE.022 contains the standard module for the DRIVE exits.

User exits can be linked with the ALLEX macro.

The DRIVE system library SIPLIB.DRIVE.022 contains the source of the ALLEX macro.

Structure of the ALLEX macro

```
[name]   ALLEX START=(exit,module,...),
               SHUT =(DRISHUT,module,...),
               FORM =(DRIFORM,module,...)
         END
```

Explanation:

name — Any module name; may be omitted.

exit — Specify the following for exit:

DRISTARD for DMS variant
DRISTARL for LEASY variant
DRISTARS for SESAM V2 variant

module — Name of the respective user exit module; maximum number: 9

The DRIVE exits DRISHUT and DRIFORM are mandatory. DRISTARD, DRISTARL or DRISTARS must also be entered. The modules must be specified in the order indicated.

3.4.7.2 KDCDEF control statements

The KDCDEF control statements for the KDCDEF utility routine are described in "*open*UTM Generating and Handling Applications" [13].

The following KDCDEF control statements are required for a UTM application in old-style operation:

MAX

```
MAX KB ≥ 512
MAX SPAB ≥ 10000
MAX NB ≥ 4632
MAX TRMSGLTH ≥ 4632
MAX LSSBS ≥ 50
```

DATABASE

The following must be specified for the database system SESAM V2.x:

```
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=sesam-lib
```

The following must be added for LEASY:

```
DATABASE TYPE=LEASY,ENTRY=LEASY
```

The DATABASE statement is not applicable for DMS.

PROGRAM

```
PROGRAM DRIKROOT,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM DRIVORG,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM EXFORM,COMP=ASSEMB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
PROGRAM KDCADM,COMP=ILCS
```

MODULE

```
MODULE DRIFORM,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRISHUT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIEXT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2INT,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2LINK,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIC51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXTAB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIVMC,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SF2REFX,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

The KDCDEF control statement MODULE EXTAB must appear before the MODULE DRIVMC statement.

The following must be specified for the database system SESAM V2.x:

```
MODULE DRISTARS,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE SESUTMC,LIB=sesam-lib,LOAD=STATIC
MODULE SESORT,LIB=sesam-lib,LOAD=STATIC
```

The following must be added for DMS:

```
MODULE DRISTARD,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

The following must be added for LEASY:

```
MODULE DRISTARL,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
```

DRIC51 is omitted for LEASY.

EXIT

```
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
EXIT PROGRAM=EXFORM,USAGE=FORMAT
```

TAC

```
DEFAULT TAC PROGRAM=DRIKROOT
TAC DRIVE   ,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRIVORG
TAC DRISES  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC DRISEQ  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC DRINEXT ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRIRTP  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC DRIENTER,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVORG
TAC DRILIST ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVORG
```

SFUNC

```
SFUNC K1,RET=2OZ
```

Integrating DRIVE in an existing UTM application

Modify or extend the MAX, PROGRAM, MODULE, EXIT, TAC and SFUNC statements in the input file for controlling the UTM utility routine KDCDEF. The required specifications are described in section 3.4.2, “Generating the application configuration and the UTM linkage program”, on page 43.

3.4.7.3 Assembling the UTM linkage program

Assembling the UTM linkage program for DIRVE/WINDOWS under old-style operation is essentially the same as the assembling process under new-style operation. Therefore, only the differences are mentioned here. A detailed description can be found in section 3.4.3, “Assembling the UTM linkage program”, on page 50.

– Assembly controlled by SYSPRC.UTM.xxx(GEN)

  The ALTLIB2 assignment is omitted for the DMS variant.

– Assembly independent of SYSPRC.UTM.xxx(GEN)

  ALTLIB2 is omitted for the DMS variant.

3.4.7.4 Generating a UTM application

To generate a UTM application for old-style operation, call the BS2000 procedure DRIPRC.INSTALL.DRIVE (see page 51).

The only difference is that you must specify the following parameter values for the &STYLE and &FASSUNG parameters:

&STYLE = OLD

`&FASSUNG = [ SESAM | LEASY | DMS ]`
Specify the database system which DRIVE is to access.

In addition you must specify the parameters &PHASE, &EDTLIB, &TSOSLNK and possibly &LEASYLIB:

`&PHASE = module-name` For *module-name* specify the name of the module to which the generated DRIVE variant is linked.
This name must be specified at startup in the /START-PROG command (see section 3.5.2.1, “Start procedure”, on page 73).

`&TSOSLNK = tsoslnk-name`
For *tsoslnk-name* specify the file name of the static linkage editor TSOSLNK.
The default name is $TSOS.TSOSLNK.

`&EDTLIB = edt-lib` For *edt-lib* specify the module library containing the EDT object module IEDTGLE.
The default library name is $TSOS.EDTLIB.

`&LEASYLIB = leasy-lib` For *leasy-lib* specify the module library with the LEASY modules.
Only necessary if &FASSUNG=LEASY.

The other parameters are described on page 51.

3.4.8 Generating a DTP application

A UTM application for DTP is generated just like a UTM application without DTP (see pages 41 through 51). However, the UTM linkage program KDCROOT must be assembled with the SYSPRG.UTM-D.xxx.KDCDEF procedure (because of UTM-D). In addition, the KDCDEF input files must be extended to include control statements for addressing the respective submitting and receiving partners.

3.4.8.1 Addressing receiving partners

The submitting partner addresses the receiving partner via the KDCS call APRO:

```
APRO KCRN=<tac-name-in-partner-application>,KCPI=<conversation-id>
     KCPA=<partner-application-name>
```

The specification <tac-name-in-partner-application> must match the LTAC specification in the submitting partner's KDCDEF.

For single-level addressing, the <partner-application-name> must be specified in the submitting partner's KDCDEF (LPAP specification in the LTAC statement). For two-level addressing, you do not need to specify the name in the submitting partner's KDCDEF: in this case, you must specify the name of the partner application as a receiving partner in the PARAMETER DISTRIBUTION statement (see "DRIVE Directory" [3]).

3.4.8.2 KDCDEF control statements

The general KDCDEF control statements for a DRIVE-UTM generation are described in section 3.4.2.1, "KDCDEF control statements", on page 43.

In addition to the DRIVE-specific transaction codes described there, you must specify the following TACs for DTP for the receiving partner:

```
TAC SQLRMT,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=DRIVROOT,TIME=30000
TAC SQLRMTA,TYPE=A,STATUS=ON,CALL=FIRST,PROGRAM=DRIVROOT,EXIT=DRIVVORG,TIME=300000
```

TAC name	Function
SQLRMT	Synchronous call of a DRIVE receiving application (CALL)
SQLRMTA	Asynchronous call of a DRIVE receiving application (ENTER)

You address the submitting/receiving applications for DTP by means of the following KDCDEF statements:

```
UTMD RSET=LOCAL
```

for the submitting and receiving partners so that, in the case of ROLLBACK WORK and program abortion with an error, the RSET call from DRIVE acts only locally on the current conversation (i.e. the local transaction is reset, not all participating transactions, as is the case with ROLLBACK WORK WITH RESET via PEND RS)

Otherwise, UTM aborts the current conversation with the error code 87Z (K320).

```
LTAC partner-tac-name [,LPAP=partner-application-name]
```

defines local names for transaction codes in the remote partner application (LPAP need only be specified for single-level addressing; with two-level addressing, the partner application is identified via the PARAMETER DISTRIBUTION statement).

```
SESCHA sescha-name,CONNECT=Y,PLU=Y/N
```

defines session characteristics.

PLU=Y/N:
PLU=N means that this application is responsible for setting up the session; PLU=Y means that the partner application is responsible for setting up the session. PLU=Y must be specified on the one computer and PLU=N on the other. These specifications are independent of which application is the submitting partner and which the receiving partner.

Recommendation: Specify all connections with PLU=Y on one computer, and the connections in the partner computer with PLU=N.

```
LSES local-session-name,LPAP=partner-application-name,
     RSES=session-name-in-partner application
```

defines the session name for the connection between two applications.

```
LPAP partner-application-name,SESCHA=sescha-name
```

defines local names for the remote partner application.

```
BCAMAPPL bcam-application-name,T-PROT=ISO
```

defines an additional local application name for the transport connection (in addition to the name assigned via MAX APPLINAME). This means that there are two or more transport connections between two applications, rather than just one, and that requests can be executed in parallel. For each concurrent connection, you must specify a BCAMAPPL statement, a CON statement and an LSES statement. The LPAP and SESCHA statements are required only once each (see the note below for an exception to this).

```
CON bcam-partner-application-name,LPAP=partner-application-name,
    BCAMAPPL=bcam-application-name,PRONAM=processor-name
```

defines the logical transport connection between the local application and the remote application.

If, in one application system, two partner applications are to be both the submitting partner and the receiving partner for each other, you should enter two SESCHA statements and two LPAP statements, together with the corresponding BCAMAPPL, CON and LSES statements. Otherwise, bottlenecks could easily occur on the connections.

Sample KDCDEF skeleton for DTP

This section contains examples of the KDCDEF control statements required when DRIVE is to access a SESAM V2.x database for DTP. The KDCDEF control statements differ depending on whether the DRIVE application is the submitting or receiving partner.

Submitting partner

```
REM
REM *** Set maximum values *************************
REM
MAX KB=512,SPAB=32767,NB=32700
MAX TASKS=6,ASYNTASKS=1
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700,
MAX PGPOOL=(1000,80,95),RECBUF=(30,4096),REQNR=8
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60, CONRTIME=10
MAX VGMSIZE=128
REM
REM *** RSET is to act locally only ***
REM
UTMD RSET=LOCAL
REM
REM *** Assign SESAM V2.2 database ********************
REM
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=$TSOS.SYSLNK.SESAMSQL.022
```

```
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=$TSOS.SYSLNK.SESAMSQL.022
REM
REM *** Define program units ************************
REM
PROGRAM KDCADM  ,COMP=ILCS
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT  ,COMP=ILCS
PROGRAM EXSHUT  ,COMP=ILCS
REM
REM *** Define USER-EXITS ***************************
REM
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM *** Load DRIVE modules **************************
REM
MODULE EXTAB  ,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
REM
REM *** Transaction codes for DRIVE *****************
REM
DEFAULT TAC PROGRAM=DRIVROOT
TAC SQLRET,TYPE=D,STATUS=ON,CALL=NEXT
REM
TAC DRISQL  ,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG
TAC DRISQLF ,TYPE=D,STATUS=ON,CALL=NEXT
TAC SQLNEXT ,TYPE=D,CALL=NEXT,TIME=300000
TAC SQLENTER,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVVORG
TAC SQLLIST ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRIVVORG
REM
REM *** Synchronous administration ******************
REM
DEFAULT TAC TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,PROGRAM=KDCADM, DBKEY=UTM
TAC KDCTAC
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCUSER
TAC KDCSEND
TAC KDCAPPL
TAC KDCDIAG
TAC KDCLOG
TAC KDCINF
TAC KDCHELP
```

```
TAC KDCSHUT
TAC KDCTCL
REM
REM *** Asynchronous administration *****************
REM
DEFAULT TAC TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCTACA
TAC KDCLTRMA
TAC KDCPTRMA
TAC KDCSWCHA
TAC KDCUSERA
TAC KDCSENDA
TAC KDCAPPLA
TAC KDCDIAGA
TAC KDCLOGA
TAC KDCINFA
TAC KDCHELPA
TAC KDCSHUTA
TAC KDCTCLA
REM
REM *** PROGRAM/MODULE/TAC/LTAC statements for ******
REM *** C - DRIVE ***********************************
REM
PROGRAM CMAINPR, COMP=ILCS
TAC SNDTACCC,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=CMAINPR
TAC RECTACCC,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=CMAINPR
REM
REM *** PROGRAM/MODULE/TAC/LTAC statements for ******
REM *** DRIVE -C ************************************
REM
LTAC CANTAC
REM
REM *** PROGRAM/MODULE/TAC/LTAC statements for ******
REM *** DRIVE - C via PEND PA local *****************
REM
PROGRAM CTACMAIN, COMP=C
TAC CLOCTAC,TYPE=D,STATUS=ON,CALL=BOTH,PROGRAM=CTACMAIN
REM
REM *** Assign function keys ************************
REM
SFUNC K1,RET=20Z
SFUNC F1,CMD=KDCOFF
SFUNC K2,CMD=KDCOUT
REM
REM *** Permitted USERs *****************************
REM
```

```
USER user1
USER user2
USER user3
USER admin1,STATUS=ON,PERMIT=ADMIN
USER admin2,STATUS=ON,PERMIT=ADMIN
USER admin3,STATUS=ON,PERMIT=ADMIN
REM
REM *** DRIVE remote TAC ****************************
REM
LTAC SQLRMT
LTAC SQLRMTA,TYPE=A
REM
REM *** Address BS2000 partner application **********
REM *** on the same computer ************************
REM
SESCHA SESBSAP1, CONNECT=YES, PLU=Y
LPAP LPABSAP2, SESCHA=SESBSAP1
REM
LSES LSBS1AP1, LPAP=LPABSAP2, RSES=LSBS1AP2
LSES LSBS2AP1, LPAP=LPABSAP2, RSES=LSBS2AP2
LSES LSBS3AP1, LPAP=LPABSAP2, RSES=LSBS3AP2
LSES LSBS4AP1, LPAP=LPABSAP2, RSES=LSBS4AP2
BCAMAPPL BCBS1AP1
BCAMAPPL BCBS2AP1
BCAMAPPL BCBS3AP1
BCAMAPPL BCBS4AP1
CON BCBS1AP2, BCAMAPPL=BCBS1AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
CON BCBS2AP2, BCAMAPPL=BCBS2AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
CON BCBS3AP2, BCAMAPPL=BCBS3AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
CON BCBS4AP2, BCAMAPPL=BCBS4AP1, LPAP=LPABSAP2, PRONAM=D016ZE07
REM
REM *** Define PTERMs and LTERMs ********************
REM
TPOOL LTERM=DRIUSER, NUMBER=9, PRONAM=D255S156, PTYPE=T9750
TPOOL LTERM=DRIUSE , NUMBER=9, PRONAM=D255S247, PTYPE=T9750
REM
TPOOL LTERM=DRI, NUMBER=10, PRONAM=D016KR20, PTYPE=T9750
REM
END
```

Receiving partner

```
REM *** Generate KDCFILE and ROOT ********************
REM
OPTION GEN=ALL
```

```
ROOT root-name
REM
MAX APPLINAME=bs2anw, KDCFILE=utm.bs2anw
REM
REM *** Set maximum values *************************
REM
MAX KB=512, SPAB=32767, NB=32700
MAX TASKS=6,ASYNTASKS=1
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700
MAX PGPOOL=(1000,80,95),RECBUF=(30,4096),REQNR=8
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60,CONRTIME=10
MAX LOGACKWAIT=600,BRETRYNR=10
MAX VGMSIZE=128
REM
REM *** RSET can act locally only *******************
UTMD RSET=LOCAL
REM
REM *** Describe databases used: SESAM V2.2 ***********
REM
DATABASE ENTRY=SESSQL,TYPE=SESAM,LIB=$TSOS.SYSLNK.SESAMSQL.022
DATABASE ENTRY=SESAM,TYPE=SESAM,LIB=$TSOS.SYSLNK.SESAMSQL.022
REM
REM *** Define program units ************************
REM
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM DRIVVORG,COMP=ILCS
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM KDCADM,COMP=ILCS
REM
REM *** Statements for connection of DRIVE submitting ******
REM *** partner with C program units as receiving partner **
REM
PROGRAM CANMAIN,COMP=ILCS
TAC CANTAC,TYPE=D,STATUS=ON,CALL=FIRST,PROGRAM=CANMAIN
REM
REM *** Define USER EXITS ***********************************
REM
EXIT PROGRAM=EXSTRT,USAGE=START
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM *** Load DRIVE modules *************************
REM DRIVE library
MODULE EXTAB,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSTART,...
MODULE EXSHUTE,...
```

```
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
REM
REM *** Transaction codes and Tacclass for DRIVE ****
REM
DEFAULT TAC PROGRAM=DRIVROOT
TAC SQLRMT,TYPE=D,STATUS=ON,CALL=BOTH,TIME=30000
TAC SQLRMTA,TYPE=A,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG,TIME=300000
REM
TAC DRISQL,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG
TAC DRISQLF,TYPE=D,STATUS=ON,CALL=NEXT
TAC SQLNEXT,TYPE=D,CALL=NEXT,TIME=300000
TAC SQLENTER,TYPE=A,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG,TIME=300000
TAC SQLLIST,TYPE=A,STATUS=ON,CALL=FIRST,EXIT=DRIVVORG,TIME=300000
REM
REM *** SYNCHRONOUS ADMINISTRATION ******************
REM
DEFAULT TAC TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,PROGRAM=KDCADM,DBKEY=UTM
TAC KDCTAC
TAC KDCLTERM
TAC KDCPTERM
TAC KDCSWTCH
TAC KDCUSER
TAC KDCSEND
TAC KDCAPPL
TAC KDCDIAG
TAC KDCLOG
TAC KDCINF
TAC KDCHELP
TAC KDCSHUT
TAC KDCTCL
REM
REM *** ASYNCHRONOUS ADMINISTRATION *****************
REM
DEFAULT TAC TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCTACA
TAC KDCLTRMA
TAC KDCPTRMA
TAC KDCSWCHA
TAC KDCUSERA
TAC KDCSENDA
TAC KDCAPPLA
TAC KDCDIAGA
TAC KDCLOGA
TAC KDCINFA
TAC KDCHELPA
TAC KDCSHUTA
```

```
TAC KDCTCLA
REM
REM *** PERMITTED USERS ****************************
REM
USER user1
USER user2
USER user3
USER admin1,STATUS=ON,PERMIT=ADMIN
USER admin2,STATUS=ON,PERMIT=ADMIN
USER admin3,STATUS=ON,PERMIT=ADMIN
REM
REM *** Assign function keys ***********************
REM
SFUNC K1,RET=2OZ              "BREAK key"
SFUNC F1,CMD = KDCOFF
SFUNC K2,CMD = KDCOUT
REM
TPOOL LTERM=DRIUSER,NUMBER=9,PRONAM=D255S156,PTYPE=T9750
TPOOL LTERM=DRIUSE,NUMBER=9,PRONAM=D255S247,PTYPE=T9750
TPOOL LTERM=DRI,NUMBER=10,PRONAM=D016KR19,PTYPE=T9750
REM
REM *** Address partner application *****************
REM *** DRIVE or C as submitting partner in BS2000 **
REM
SESCHA SESBSAP2,CONNECT=Y,PLU=N
REM
LPAP LPABSAP1, SESCHA=SESBSAP2
LSES LSBS1AP2, LPAP=LPABSAP1, RSES=LSBS1AP1
LSES LSBS2AP2, LPAP=LPABSAP1, RSES=LSBS2AP1
LSES LSBS3AP2, LPAP=LPABSAP1, RSES=LSBS3AP1
LSES LSBS4AP2, LPAP=LPABSAP1, RSES=LSBS4AP1
REM
REM
BCAMAPPL BCBS1AP2
BCAMAPPL BCBS2AP2
BCAMAPPL BCBS3AP2
BCAMAPPL BCBS4AP2
REM
CON BCBS1AP1,BCAMAPPL=BCBS1AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
CON BCBS2AP1,BCAMAPPL=BCBS2AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
CON BCBS3AP1,BCAMAPPL=BCBS3AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
CON BCBS4AP1,BCAMAPPL=BCBS4AP2,LPAP=LPABSAP1,PRONAM=D016ZE07
END
```

3.4.8.3 Assembling UTM linkage program KDCROOT

Because of UTM-D, the UTM linkage program must be assembled with the SYSPRG.UTM-D.xxx.KDCDEF procedure.

3.4.8.4 Error handling

For KDCS calls, DRIVE evaluates the fields KCRCCC (KDCS error code) and KCRCDC (internal error code), which are set by UTM in the KB return area.

KDCS error codes for APRO calls

The error codes 40Z,44Z and 46Z cause the program to be aborted; all other error codes cause the conversation to be terminated.

KDCS error codes for MPUT calls

All error codes cause the conversation to be terminated.

KDCS error codes for FPUT calls

The error codes 40Z and 44Z cause the program to be aborted; all other error codes cause the conversation to be terminated.

KDCS error codes for MGET and FGET calls

All error codes cause the conversation to be terminated.

3.5 Starting a DRIVE UTM application

(Changes to chapter 16 of “DRIVE Programming System” [1])

3.5.1 Prerequisites

The following files must be available to the UTM application:

– The file containing the load module of the UTM application program

  (The file name was specified with the parameter &STARTLLM = module-name or &PHASE = module-name in the DRIPRC.INSTALL.DRIVE procedure. See section 3.4.4, “Generating UTM applications”, on page 51, page 56 (mixed operation), page 61 (old-style) and section 3.4.8, “Generating a DTP application”, on page 62).

– KDCFILE (file name extension: KDCA)

– Where applicable, a user log file (file name extension: USLA)

– System log file SYSLOG

– Module library $TSOS.SYSLNK.UTM.xxx (xxx=version)

– Where applicable, user files of the UTM application (only in mixed operation)

If desired, modules can be loaded as shared code in the class 3/4 memory using DSSM (see section 3.2.1, “Loading DRIVE modules as shared code”, on page 29).

3.5.2 Starting an application

To prevent the terminal, from which the UTM production application was started, from being blocked for the duration of the UTM application, the UTM application should be started by a BS2000 batch procedure (see section 3.5.2.1, “Start procedure”, on page 73).

A description of how the start procedure of an existing UTM application must be modified for DRIVE can be found under “Modifying the start procedure of an existing UTM application for DRIVE” on page 76.

A framework start procedure called DRIPRC.START can be found in the SYSPRC.DRIVE.022 library.

The name of the file containing the start procedure must be specified as a UTM start parameter.

3.5.2.1 Start procedure

A BS2000 procedure for starting a UTM application has the following components:

- assignment of the module libraries (see section 3.2.2, “Assigning module libraries”, on page 31)
- creation of the system log file SYSLOG
- indication of the database configuration name (for SESAM)
- if applicable, creation of a list file DRILIST (see section 3.2.5, “Readying a list file for UTM applications”, on page 34)
- call of the UTM application program
- the start parameters for the UTM application
- the start parameters for the formatting system
- the DRIVE start parameters

  The DRIVE start parameters define the DRIVE parameters valid for all users of the UTM application; for example, for assigning the DRIVE library, requesting DRIVE-specific memory areas (DRIVE cache) and loading the K/F keys with DRIVE functions.

  You must also specify the DRIVE statements required to run a DRIVE program (e.g. dynamic parameters) if the user is not permitted to enter these statements in interactive mode see PARAMETER LOCK parameter) or if these statements are not processed in the leader program (see the manual “DRIVE Programming System” [1], section “Data protection in UTM applications“).
- branch statement to the start the procedure following abortion due to error

The UTM application program reads the start parameters via SYSDTA.

Sample start procedure for a UTM production application with DRIVE new-style:

```
/LOGON ————————————————————————————————————————————————————————————————————— (1)
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=base-name.SYSLOG,SHARED-UPDATE=YES ┐
                                                                             ├─── (2)
/SET-FILE-LINK FILE-NAME=base-name.KDCA,SHARED-UPDATE=YES                    ┘
/MODIFY-MSG-FILE-ASSIGNMENT ADD-FILE=$userid.SYSMSA.ESQL-COBOL.022     ┐
/MODIFY-MSG-FILE-ASSIGNMENT ADD-FILE=$userid.SYSMSA.SESAMSQL.022       ├─────── (3)
/MODIFY-MSG-FILE-ASSIGNMENT ADD-FILE=$userid.SYSMSA.SES-SQL-GEM.022    ┘
```

```
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crte-lib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lms-lib
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhs-macro-lib
                                                                    (4)
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=system-macro-lib
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usr-lib
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=format-lib
/SET-FILE-LINK LINK-NAME=RSOML,FILE-NAME=SYSLIB.DRIVE.022
/SET-FILE-LINK LINK-NAME=DRILIST,FILE-NAME=list-file
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesam-lib
                                                                    (5)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesam-conf
/.NEW REMARK
/START-PROGRAM FROM-FILE=*MOD(LIB=root-lib,ELEM=module-name,PROG-MO=ANY,-
/              RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,UN-EXTRNS=DELAY,-    (6)
/              LO-IN=REF))
.UTM START FILEBASE=base-name
.UTM START TASKS=3
                                                                    (7)
.UTM START ASYNTASKS=1
.UTM START STARTNAME=enter-file
.FHS DE=NO
                                                                    (8)
.FHS MAPLIB=format-lib
.UTM END                                                            (9)
.DRIVE PAR DYNAMIC LIBRARY=lib-name
.DRIVE PAR DYNAMIC CATALOG=project SCHEMA=employ
.DRIVE PAR DYNAMIC AUTHORIZATION=secret
.DRIVE PAR KFKEY='K1' ACTION=BREAK UTMRC='20Z'
                                                                    (10)
.DRIVE PAR KFKEY='K3' ACTION=EXIT UTMRC='33Z'
.DRIVE PAR KFKEY='K4' UTMRC='25Z'
.DRIVE ACQUIRE MEMORY 120 USER 5
.DRIVE END
/SKIP-COMMANDS .NEW                                                 (11)
/LOGOFF
```

Explanation:

(1) The start procedure should be a batch procedure so that the terminal from which it is started is not blocked by an interactive task.

(2) Assignment of system files

(3) Assignment of SESAM message files (necessary for SESAM V2.x)

(4) Assignment of module libraries and creation of the list file DRILIST

(5) Assignment of the SESAM module library

(6) Use the /START-PROGRAM command to call the generated UTM application.

(7) UTM start parameters:

- The base name of the KDCFILE and the user log file were defined in the &FILEBASE parameter of SYSPRC.UTM.xxx(GEN) or with the KDCDEF control statement MAX KDCFILE=*base-name*.
- Three tasks are generated, one of them for asynchronous conversations.
- *enter-file* specifies the name of the file containing the batch procedure for the start of the UTM application.

(8) FHS start parameters

(9) End of the input of UTM start parameters and of the start parameter for the form generating system. The UTM application is started, i.e. the start exit is activated next.

(10) DRIVE start parameters (see statements PARAMETER DYNAMIC, PARAMETER KFKEY, PARAMETER STATIC and ACQUIRE in the “DRIVE Directory” [3])

The UTMRC specifications for PAR KFKEY must match the SFUNC specification KDCDEF (see section “SFUNC” on page 46).

(11) Branch to /.NEW

If the UTM application is aborted because of an error, a new start is initiated immediately.

Modifying the start procedure of an existing UTM application for DRIVE

Depending on the DRIVE variant, you can modify and extend the UTM start procedure with the following commands and statements:

```
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crte-lib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lms-lib
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhs-macro-lib
                                                                  (1)
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=system-macro-lib
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usr-lib
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=format-lib
/SET-FILE-LINK LINK-NAME=RSOML,FILE-NAME=SYSLIB.DRIVE.022
/SET-FILE-LINK LINK-NAME=DRILIST,FILE-NAME=list-file
```

The following must be specified for the SESAM database system:

```
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesam-lib
                                                                  (2)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesam-conf
.FHS DE=NO
                                                                  (3)
.FHS MAPLIB=format-lib
.DRIVE start-parameters                                           (4)
.DRIVE END
```

Explanation:

(1) Assignment of module libraries and creation of the list file DRILIST

(2) Assignment of the module library for the database system

(3) Start parameters for the form generating system

(4) DRIVE start parameters (see page 73)

3.5.2.2 Start procedure for mixed operation

Both the new-style and old-style start parameters must be specified in the UTM start procedure for DRIVE mixed operation. If the start parameters contain an error, an error message is output to SYSLST and the start of the UTM application is aborted.

Sample start procedure of a UTM test application (SESAM V2.x variant):

```
/LOGON ------------------------------------------------------------------------  (1)
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=base-name.SYSLOG,SHARED-UPDATE=YES
                                                                          -----  (2)
/SET-FILE-LINK FILE-NAME=base-name,KDCA,SHARED-UPDATE=YES
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022
/SET-FILE-LINK LINK-NAME=LIBOML,FILE-NAME=SYSPRG.DRIVE.022
/SET-FILE-LINK LINK-NAME=BLSLIB01,FILE-NAME=crte-lib
/SET-FILE-LINK LINK-NAME=BLSLIB02,FILE-NAME=lms-lib
                                                                  ----------  (3)
/SET-FILE-LINK LINK-NAME=BLSLIB03,FILE-NAME=fhs-macro-lib
/SET-FILE-LINK LINK-NAME=BLSLIB04,FILE-NAME=system-macro-lib
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usr-lib
/SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=format-lib
/SET-FILE-LINK LINK-NAME=DRILIST,FILE-NAME=list-file.21
/SET-FILE-LINK LINK-NAME=DRUCK,FILE-NAME=list-file.51
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesam-lib
                                                                  ----------  (4)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesam-conf
/.NEW REMARK
/START-PROGRAM FROM-FILE=*MOD(LIB=root-lib,ELEM=module-name,PROG-MO=ANY,-
/              RUN-MO=ADV(ALT-LIB=YES,NAME-COL=ABORT,UN-EXTRNS=DELAY, -       -----  (5)
/              LO-IN=REF))
.UTM START FILEBASE=base-name
.UTM START TASKS=3
                                  ------------------------------------------  (6)
.UTM START ASYNTASKS=1
.UTM START STARTNAME=enter-file
.FHS DE=NO
.FHS MAPLIB=format-lib
.UTM END ----------------------------------------------------------------------  (7)
.DRIVE PAR DYNAMIC LIBRARY=lib-name
.DRIVE PAR DYNAMIC CATALOG=project SCHEMA=employ
.DRIVE PAR DYNAMIC AUTHORIZATION=secret
.DRIVE PAR KFKEY='K1' ACTION=BREAK UTMRC='20Z'  ----------------------------  (8)
.DRIVE PAR KFKEY='K3' ACTION=EXIT UTMRC='33Z'
.DRIVE ACQUIRE MEMORY 120 USER 5
.DRIVE END
```

```
.DRIVE SEQUENCE
.DRIVE PAR PLAMLIB='lib-link',FORMLIB='flib-name
.DRIVE PAR DD=ON,PASSWORD=OFF
.DRIVE PAR KFKEY='K1',ACTION=BREAK,UTMRC='2OZ'
.DRIVE PAR KFKEY='K3',ACTION=EXIT,UTMRC='33Z'
.DRIVE ACQUIRE MEMORY WITH 32 FOR VIEWS ———————————————————— (9)
.DRIVE ACQUIRE MEMORY WITH 120 FOR 5 USER
.DRIVE ACQUIRE LIST FILE WITH DRUCK
.DRIVE ACQUIRE FILE TEST2 WITH T2
.DRIVE ACQUIRE PLAM FILE 'PLAM.BIBL1' WITH PLIB
.DRIVE END SEQUENCE
/SKIP-COMMANDS .NEW ———————————————————————————————————————— (10)
/LOGOFF
```

Explanation:

(1) The start procedure should be a batch procedure so that the terminal from which it is started is not blocked by an interactive task.

(2) Assignment of system files

(3) Assignment of module libraries

(4) Assignment of the database

(5) Use the /START-PROGRAM command to call the generated UTM application.

(6) UTM start parameters:

- The base name for KDCFILE was defined in the &FILEBASE parameter of SYSPRC.UTM.xxx(GEN).
- Three tasks are generated, of which one is reserved for asynchronous conversations.
- *enter-file* specifies the name of the file containing the batch procedure for the start of the UTM application.

(7) End of the input of the UTM start parameters as well as the start parameters for the form generating system. The UTM application is started, i.e. the start exit is activated next.

(8) DRIVE start parameters for new-style operation. See the PARAMETER DYNAMIC, PARAMETER KFKEY, and ACQUIRE statements in the “DRIVE Directory” [3]. The UTMRC specification in PAR KFKEY must match the SFUNC specification in KDCDEF (see section “SFUNC” on page 46).

(9) DRIVE start parameters for old-style operation. See also notes below.

PARAMETER
For the old-style variant, the PLAMLIB specification defines the PLAM library used as the DRIVE library for procedures.

FORMLIB defines the PLAM library in which the FHS forms are stored.

PASSWORD=OFF deactivates password protection within the application for the SESAM variant.

KFKEY specifications are mandatory both in old-style and new-style operation.

ACQUIRE
MEMORY FOR VIEWS defines the initial size of the place holder for user-specific view definitions. DRIVE automatically expands the memory space to the required size.

MEMORY FOR USER defines the size of the DRIVE cache. If this start parameter is also specified for new-style operation, the two specifications must match. The table below lists the formulae for calculating the size of the DRIVE cache:

Implementation variant	**Formula**	**Rounded to**
Old-style	mlength x 1024 x cn +96 +cn x 24	1 Mbyte
New-style	mlength x 1025 x cn	1 Mbyte

mlength: Size of a memory area within the cache memory in Kilobytes

cn: Number of DRIVE UTM users whose internal system data is to be buffered in parallel in the cache memory when switching UTM dialog steps.

LIST FILE defines the list file. The V5.1 list file must be different to the V2.1 list file.

(10) Branch to /.NEW

A new start is initiated immediately if the UTM application is aborted due to an error.

The DRIVE start parameters for new-style operation must precede those for old-style operation.

3.5.2.3 Start procedure for old-style operation

The old-style start parameters have to be entered for a UTM start procedure under old-style operation. If the start parameters contain an error, an error message is output to SYSLST and the start of the UTM application is aborted.

The following start procedure is required if you wish to run a UTM application under DRIVE old-style operation:

```
/LOGON ------------------------------------------------------------------------ (1)
/ASSIGN-SYSDTA *SYSCMD
/SET-FILE-LINK LINK-NAME=SYSLOG,FILE-NAME=base-name.SYSLOG,SHARED-UPDATE=YES
                                                                                 (2)
/SET-FILE-LINK FILE-NAME=base-name.KDCA,SHARED-UPDATE=YES
/SET-FILE-LINK LINK-NAME=USEROML,FILE-NAME=usr-lib
/SET-FILE-LINK LINK-NAME=DRIVEOML,FILE-NAME=SYSLNK.DRIVE.022 ------------------ (3)
/SET-FILE-LINK LINK-NAME=DRUCK,FILE-NAME=list-file.51
/SET-FILE-LINK LINK-NAME=SESAMOML,FILE-NAME=sesam-lib
                                                                                 (4)
/SET-FILE-LINK LINK-NAME=SESCONF,FILE-NAME=sesam-conf
/.NEW REMARK
/START-PROG module-name
.UTM START FILEBASE=base-name
.UTM START TASKS=3
                                                                                 (5)
.UTM START ASYNTASKS=1
.UTM START STARTNAME=enter-file
.UTM END ---------------------------------------------------------------------- (6)
.DRIVE SEQUENCE
.DRIVE PAR PLAMLIB='lib-link',FORMLIB='flib-name
.DRIVE PAR PASSWORD=OFF
.DRIVE PAR KFKEY='K1',ACTION=BREAK,UTMRC='20Z'
.DRIVE PAR KFKEY='K3',ACTION=EXIT,UTMRC='33Z'
.DRIVE ACQUIRE MEMORY WITH 32 FOR VIEWS ---------------------------------------- (7)
.DRIVE ACQUIRE MEMORY WITH 120 FOR 5 USER
.DRIVE ACQUIRE LIST FILE WITH DRUCK
.DRIVE ACQUIRE FILE TEST2 WITH T2
.DRIVE ACQUIRE PLAM FILE 'PLAM.BIBL1' WITH PLIB
.DRIVE END SEQUENCE
/SKIP-COMMANDS .NEW ----------------------------------------------------------- (8)
/LOGOFF
```

Explanation:

(1) The start procedure should be a batch procedure so that the terminal from which it is started is not blocked by an interactive task.

(2) Assignment of system files

(3) Assignment of module libraries (see section 3.2.2, “Assigning module libraries”, on page 31)

(4) Assignment of database

(5) UTM start parameters:

- The base name for KDCFILE was defined in the &FILEBASE parameter of SYSPRC.UTM.xxx(GEN).
- Three tasks are generated, of which one is reserved for asynchronous conversations.
- *enter-file* specifies the name of the file containing the batch procedure for the start of the UTM application.

(6) End of the input of the UTM and SESAM start parameters as well as the start parameters for the form generating system. The UTM application is started, i.e. the start exit is activated next.

(7) DRIVE start parameters for old-style operation. See also notes below.

PARAMETER
For the old-style variant, the PLAMLIB specification defines the PLAM library used as the DRIVE library for procedures. If no value is specified, there is no library for procedures.

FORMLIB defines the PLAM library in which the FHS forms are stored.

PASSWORD=OFF deactivates password protection within the application for the SESAM variant.

ACQUIRE
MEMORY FOR VIEWS defines the initial size of the place holder for user-specific view definitions. DRIVE automatically extends the memory space to the required size.

MEMORY FOR USER defines the size of the DRIVE cache. The table below lists the formula for calculating the size of the DRIVE cache:

Implementation variant	Formula	Rounded to
Old-style	mlength x 1024 x cn +96 +cn x 24	1 Mbyte

mlength: size of a memory area within the cache memory in kilobytes

cn: number of DRIVE UTM users whose internal system data is to be buffered in parallel in the cache memory when switching UTM dialog steps.

LIST FILE defines the list file.

(8) Branch to /.NEW

A new start is initiated immediately if the UTM application is aborted due to an error.

3.5.3 Starting and terminating a dialog

You will find a description of how to start and terminate a dialog with a UTM production application in section 3.1.2, “Dialog structure in UTM applications”, on page 20.

3.5.4 Terminating an application

To terminate the UTM application normally, the following options are available to the system administrator:

Normal termination of the UTM application:

– KDCSHUT NORMAL from a terminal

– BCLOSE from the console (only by the operator)

The UTM application can be **abnormally** terminated by:

– administration command KDCSHUT KILL

– internal UTM errors

– errors in the system environment

– user errors

4 DRIVE SQL statements

This chapter contains all the new and modified DRIVE SQL statements. The following list provides an overview:

New DRIVE SQL statements

ALTER SPACE

ALTER STOGROUP

CREATE INDEX

CREATE SPACE

CREATE STOGROUP

CREATE SYSTEM_USER

CREATE USER

DROP INDEX

DROP SPACE

DROP STOGROUP

DROP SYSTEM_USER

DROP USER

REORG STATISTICS

UTILITY

Modified DRIVE SQL statements

ALTER TABLE

CREATE SCHEMA

CREATE TABLE

CREATE TEMPORARY VIEW

DECLARE CURSOR

DROP SCHEMA

DROP TABLE

DROP VIEW

GRANT

PRAGMA

REVOKE

ALTER SPACE - Modify space parameters

You use ALTER SPACE to modify the parameters of a user space.

The SPACE view of the INFORMATION_SCHEMA provides you with information on which user spaces have been defined (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must own the space. If the storage group is modified, the current authorization identifier must have the special privilege USAGE for the new storage group.

```
ALTER SPACE space
          [ { PCTFREE percent | NO LOG } ...]
          [ USING STOGROUP stogroup ]
```

You must specify at least one of the parameters PCTFREE, NO LOG or USING STOGROUP, and each parameter may only be specified once.

space
Name of the space for which parameters are to be modified.

You can qualify the space name with a database name.

PCTFREE *percent*
Free space reservation in the space file expressed as a percentage. *percent* must be an unsigned integer between 0 and 70. The modified free space reservation is not evaluated until the next time the database is reorganized with the REORG utility statement.

PCTFREE *percent* omitted:
The setting for the free space reservation remains unchanged.

NO LOG
Deactivate logging.

Logging is deactivated immediately after the current transaction is terminated with the COMMIT statement.

NO LOG omitted:
The logging setting remains unchanged.

USING STOGROUP *stogroup*

The name of the storage group containing the volumes to be used for the space file. The new storage group is not evaluated until the next time the database is recovered or reorganized with the utility statements RECOVER and REORG respectively.

You can qualify the name of the storage group with a database name. This database name must be the same as the database name of the space.

USING STOGROUP *stogroup* omitted:

The storage group for the space remains unchanged.

Example

In the example below, the free space reservation and the storage group for a space is modified.

```
ALTER SPACE my_space
    PCTFREE 60
    USING STOGROUP abraxas
```

See also

CREATE SPACE, CREATE STOGROUP

ALTER STOGROUP - Modify storage group

You use ALTER STOGROUP to modify the definition of a storage group.

The STOGROUPS view of the INFORMATION_SCHEMA provides you with information on which storage groups have been defined (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must have the special privilege CREATE STOGROUP and must own the storage group.

```
ALTER STOGROUP stogroup
        { ADD VOLUMES (volume_name,... [ON dev_type]) |
          DROP VOLUMES (volume_name,...) |
          PUBLIC |
          TO catid
        }
```

---

stogroup
: Name of the storage group for which the definition is to be updated. You can qualify the name of the storage group with a database name.

ADD VOLUMES (*volume_name*,...)
: Adds new private volumes to the storage group. *volume_name* is an alphanumeric literal indicating the VSN of the volumes. Each VSN can only be specified once for a storage group.

  If the storage group previously consisted of private volumes, the new volumes being added must have the same device type.

  A storage group can comprise up to 100 volumes.

  ON *dev_type*
  : Alphanumeric literal indicating the device type of the private volumes.
    You must specify the device type if the storage group was previously set up on public volumes (PUBLIC).
    If the storage group previously consisted of private volumes, you can omit ON *dev_type*. If you do specify ON *dev_type*, you must specify the same device as before.

    ON *dev_type* omitted:
    The storage group consists of private volumes which all have the same device type as before.

DROP VOLUMES (*volume_name*,...)
Deletes individual private volumes from the definition of the storage group. *volume_name* is an alphanumeric literal indicating the VSN of the volume.
You cannot delete the last volume in a storage group.

PUBLIC
The storage group is set to the default Public Volume Set (PVS) of the BS2000 user ID under which the DBH is running. All private volumes are deleted from the definition of the storage group.

TO *catid*
The new catalog identifier for the volumes is entered in the definition of the storage group. *catid* is an alphanumeric literal indicating the new catalog ID.

In the case of private volumes, the new catalog ID is only used for cataloging the files. The files themselves are still stored on the private volumes. In the case of a PVS, the catalog ID of the PVS on which the storage group is located is changed.

Effect of ALTER STOGROUP

The ALTER STOGROUP statement only modifies the definition of the storage group. It does not affect existing spaces that the volumes in the storage group use.

Volumes deleted from the storage group are not, however, used for new storage space assignments for the spaces. Volumes can be deleted from the storage group explicitly with DROP VOLUME or implicitly by changing from public volumes (PUBLIC) to private volumes or vice versa.

The new definition of the storage group takes effect when files (spaces or backups) are created in the storage group.

Examples

1. The example below changes the storage group from private volumes to the PVS with the catalog ID O. This is done in two steps:

```
ALTER STOGROUP abraxas PUBLIC
ALTER STOGROUP abraxas TO 'O'
```

2. The example below changes the storage group ORION from PUBLIC to private volumes. The catalog ID for the files in the storage group remains unchanged.

```
ALTER STOGROUP my_db.orion
   ADD VOLUMES ('DXO17A','DXO17B') ON 'D3475'
```

See also

CREATE STOGROUP

ALTER TABLE - Modify base table

You use ALTER TABLE to modify an existing base table. You can add, update or delete columns, or you can add or delete integrity constraints.
The value for the reservation of free space which is specified using CREATE SPACE ... PCTFREE is not taken into account.

If you are using a CALL DML table, you can only add, update or delete columns. The restrictions that apply to CALL DML tables are described in “Special considerations for CALL DML tables” on page 97.

You can use the UTILITY MODE pragma to add, change or delete a column in a table (ADD, ALTER, DROP). When you activate the pragma (UTILITY MODE ON), the associated statement is performed outside a transaction like a utility statement. This suppresses normal transaction logging for the corresponding statement and thus makes it possible to accelerate performance considerably when modifying large data volumes. However, if an error occurs, it is not possible to roll back the statement. The space containing the base table to be changed is defective and must be repaired (see “UTILITY MODE pragma clause” on page 149).

You cannot use ALTER TABLE to change the table type! To change the table type, you use the utility statement MIGRATE (see the “SESAM/SQL-Server - SQL Reference Manual, Part 2” [7]).

The BASE_TABLES view in the INFORMATION_SCHEMA provides you with information on which base tables have been defined (see the chapter “Information schemas” in the manual “SESAM/SQL-Server - SQL Reference, Part 1” [6]).

The current authorization identifier must own the schema to which the base table belongs.

This statement can destroy declaration statements in a DRIVE program. A static cursor must be created in the declaration section of the DRIVE program. If an ALTER TABLE statement in the body of the DRIVE program updates the table for which the cursor was declared, the cursor can no longer be accessed.

The corresponding possibilities for SESAM/SQL represent an extension of the SQL2 standard.

```
ALTER TABLE table

      { ADD [COLUMN] column_definition,... |

        ALTER [COLUMN] { column
                         { DROP DEFAULT |
                           SET [(number)] data_type [CALL DML call_dml_default] |
                           SET DEFAULT { alphanumeric_literal |
                                         numeric_literal |
                                         time_literal |
                                         CURRENT_DATE |
                                         CURRENT_TIME(3) |
                                         CURRENT_TIMESTAMP(3) |
                                         USER |
                                         CURRENT_USER |
                                         SYSTEM_USER |
                                         NULL
                                       }
                         }
                       }, ...
                     [ USING FILE exception_file [ PASSWORD password ] ] } |

        DROP [COLUMN] column,... { CASCADE | RESTRICT } |

        ADD [ CONSTRAINT integrity_constraint_name ] table_constraint |

        DROP CONSTRAINT integrity_constraint_name { CASCADE | RESTRICT }
```

table
Name of a base table.

ADD [COLUMN] *column_definition*,...
Adds new columns to the base table. The new columns are added to the existing columns in the base table. *column_definition* defines the columns.

No primary key must be defined in *column_definition*.

An authorization identifier which possesses table privileges for the underlying base table automatically obtains the corresponding privileges for the newly added columns.

ALTER [COLUMN] *column*
column is the name of the column to be modified.

If temporary views[1] which are based on the base table exist, they are deleted.

[1] Temporary views are no longer supported as of SESAM/SQL V3.0.

Modifications of the column are performed in the following order:

– DROP DEFAULT

– SET *data_type*

– SET default

You can use one and the same modification type only once for a column.

DROP DEFAULT
Deletes the default (SQL default value) for the column.

The underlying base table must not be a CALL DML table.

SET *data_type*
New data type of the column.

The column whose data type is to be changed must not be column of a primary key. In CALL DML only tables, the column of a primary key can also be specified.

The column must not be used in views, indices and integrity constraints.

You can also change the data type of a multiple column. When a data type is changed to a multiple column data type, SESAM/SQL assigns the position number 1 to the first column element. The number of column elements corresponds to the dimension of the new data type.

An atomic column can contain the multiple column data type and vice versa. In this case, SESAM/SQL considers the atomic value to be the same as the value of a multiple column with dimension 1.

The original column data type can only be modified to certain target data types. The following table illustrates which original data types can be modified to which new data types and which combinations are not, or only partly, permitted:

Original data type	**New data type**						
	INTEGER SMALLINT DECIMAL NUMERIC	REAL DOUBLE PRECISION FLOAT	VAR-CHAR	CHAR	DATE	TIME(3)	TIME-STAMP(3)
INTEGER SMALLINT DECIMAL NUMERIC	yes	yes [1]	no	yes	no	no	no
REAL DOUBLE PRECISION FLOAT	yes	yes	no	yes	no	no	no

	New data type						
VARCHAR	no	no	yes [2]	no	no	no	no
CHAR	yes	yes [1]	no	yes	yes [1]	yes [1]	yes [1]
DATE	no	no	no	yes	yes	no	no
TIME(3)	no	no	no	yes	no	yes	no
TIME-STAMP(3)	no	no	no	yes	no	no	yes

[1] A column may only be changed to the numeric data types REAL, DOUBLE PRECISION and FLOAT or the time data types DATE, TIME and TIMESTAMP if the underlying base table is an SQL table.

[2] A column with VARCHAR data type may only be changed to the new VARCHAR data type with $new\_length \geq old\_length$.
The other data types may not be changed to the VARCHAR data type and vice versa.

SESAM/SQL converts all values in *column* to the new data type row by row. In the case of multiple columns, SESAM/SQL converts the significant values of all variants whose position number is smaller than or equal to the new data type dimension. This means that it is possible that an element's position may change within the multiple column: If the result of converting a column is the NULL value, all following elements whose position number is smaller than or equal to the new data type dimension are shifted to the left and the NULL value is appended after them. The same rules apply when converting a column value as when converting a value by means of the CAST expression.

The rules for converting a column value are described in section 6.1, "Data conversion with SQL statements", on page 167.

If a conversion error occurs, an error message or alert is issued.

The rounding of a value does not represent a conversion error.

Example

A column of NUMERIC data type is changed to the data type INTEGER. SESAM/SQL converts the original column value 450.25 to 450 without issuing an alert.

When conversion errors occur, SESAM/SQL differentiates between truncated strings, truncated column elements and non-convertible values:

- truncated strings
  A column with CHARACTER data type is to be changed to a new CHARACTER data type with shorter length. Affected column values which are longer than the new value are truncated to the length of the new data type. If characters which are not spaces are removed, SESAM/SQL issues an alert.

  Example

  The value 'cust_service' in a column which is of data type CHARACTER(12) is to be converted to data type CHARACTER(6). The original column value is replaced by the value 'cust_s'. SESAM/SQL issues an alert.

- truncated column elements
  A multiple column contains at least one column element whose position number is greater than the dimension of the new data type and which contains a significant value not equal to NULL.

  Example

  A multiple column of data type (7) CHARACTER (20) is to be converted to the data type (5) CHARACTER (20). In some table rows, all 7 elements of the multiple row contain an alphanumeric value.

- Non-convertible values
  For certain column values, a change of data type results in the loss of values (data exception).

  Examples

  – The value of an original column of numeric data type is too large for the target numeric data type.

    Example

    The value 9999 in an INTEGER column is to be converted to the data type NUMERIC(2,0).

  – A column of CHARACTER type is converted to a numeric data type. The original value of the column cannot be represented as numeric value.

    Example

    The value 'Otto' in a column with data type CHARACTER(4) is to be converted to the data type INTEGER.

  – The length of the value in an originally numeric column or in a column with a time data type is too large for the alphanumeric target data type CHARACTER.

    Example

    The value 9999 in a column of data type INTEGER is to be converted to the data type CHARACTER(2).

If the column definition for *column* contains a default, the new data type may not contain a dimensional specification.
If the specified SQL default value is an alphanumeric, numeric or time literal, it is converted to the new data type. The conversion must not result in a conversion error.
If the specified SQL default value is a time function, a special literal or the NULL value, it is not changed.
After conversion, the SQL default value for the new data type must conform to the assignment rules for default values.

CALL DML *call_dml_default*

Changes the non-significant value of column in a CALL DML table. This specification may only be used in CALL DML tables.
call_dml_default corresponds to the non-significant attribute value of SESAM/SQL version 1.x.
You must specify *call_dml_default* as an alphanumeric literal.

CALL DML *call_dml_default* not specified:
If the data type modification applies to the column in a CALL DML/SQL table, *column* retains the non-significant attribute value which was assigned to it during column definition.
If the data type modification applies to a column of a CALL DML only table, i.e. a column with “old” attribute formats from SESAM versions < V13.1, *column* is assigned the following non-significant attribute value:

– space if the *column* data type is alphanumeric
– digit 0 if the *column* data type is numeric

SET DEFAULT *default*

Defines a new SQL default value for the column.

The underlying base table cannot be a CALL DML table.

column cannot be a multiple column.

default must conform to the assignment rules for default values.

The default is evaluated when a row is inserted or updated and the default value is used for *column*.

USING FILE *exception_file* [PASSWORD *password*]

Defines the name of the exception file. *exception_file* must be specified as an alphanumeric literal.
SESAM/SQL creates or uses the exception file only if a column conversion performed using SET *data_type* results in one or more conversion errors (see page 92).

If an exception file is specified, a statement which results in a conversion error is continued. SESAM/SQL issues an alert and replaces the original column values by new values in the affected base table:

– truncated strings are replaced by the corresponding truncated value.

– non-convertible values are replaced by the NULL value.

– column items in a multiple column whose position number is larger than the new data type dimension are truncated.

SESAM/SQL logs the original column values and truncated column elements together with the associated alert or error message in the exception file.

Even when UTILITY MODE is switched ON, a statement which results in a conversion error is not interrupted. The space which contains the base table to be updated remains intact.

For a detailed description of the exception file and its contents refer to “Exception file of SQL statement ALTER TABLE” on page 99.

PASSWORD *password*
BS2000 password for the error file. *password* must be specified as an alphanumeric literal.

password can be specified in several different ways:

– 'C"*string*"'
string contains four printable characters.

– 'X"*hex-string*"'
hex-string contains eight hexadecimal characters.

– '*n*'
n is an integer between - 2147483648 and + 2147483647

USING FILE *exception_file* not specified:
If a column conversion performed using SET *data_type* results in a conversion error, SESAM/SQL does not log the affected column values or column elements in an exception file.
Strings are truncated to the length of the new data type and SESAM/SQL issues an alert.
If conversion errors occur because values cannot be converted or column elements have to be truncated, SESAM/SQL aborts the associated statement and issues an error message.

DROP [COLUMN] *column*,... { CASCADE | RESTRICT }

Deletes one or more columns and associated indices in the base table.
column is the name of the column to be deleted. You can specify the same column name only once.

No primary key may be defined for *column*.

You must not specify all columns in the base table.

Deleting a column withdraws the column privileges UPDATE and FOREIGN KEY... REFERENCES for this column from the current authorization key. If these privileges have been passed on, then the passed on privileges are also withdrawn.
In addition, deleting the column also deletes all views where *column* was used in the view definition as well as all views whose definitions contain the name of such a “higher level” view.

The arrangement of the remaining columns in a table can change: if deleting a column results in a gap, all following columns are shifted to the left.

CASCADE

Deletes the specified column(s) and associated indices.
The integrity constraints of other tables or columns which use *column* are also deleted.

You cannot use the UTILITY MODE pragma. If you activate the UTILITY MODE, an error message is output and the statement is aborted.

RESTRICT

Restricts the ways a column can be deleted:
You cannot delete the column if you use it in a view definition. You may only define an index for the column to be deleted if none of the remaining columns in the base table is named in the affected index definition. The same applies to the integrity constraints.

The UTILITY MODE pragma can be activated.

ADD CONSTRAINT clause

Adds an integrity constraint to the base table.

CONSTRAINT *integrity_constraint_name*

Assigns a name to the integrity constraint. You can qualify the name of the integrity constraint with a database and schema name. The database and schema name must be the same as the database and schema name of the base table.

CONSTRAINT *integrity_constraint_name* omitted:
The integrity constraint is assigned a name according to the following pattern:

{ UN | FK | CK } *integrity_constraint_number*

where UN stands for UNIQUE, FK for FOREIGN KEY and CH for CHECK. *integrity_constraint_number* is a 16-digit number.

table_constraint
Specifies an integrity constraint for the table. *table_constraint* cannot define a primary key constraint.

DROP CONSTRAINT *integrity_constraint_name* { CASCADE | RESTRICT }

Deletes the integrity constraint *integrity_constraint_name*. *integrity_constraint_name* cannot name a primary key constraint.

CASCADE
If *integrity_constraint_name* is a uniqueness constraint, and if the reference constraint of another table references the column(s) for which *integrity_constraint_name* was defined, the reference constraint of the other table is also implicitly deleted.

RESTRICT
You must not delete a uniqueness constraint on a column if a referential constraint on another table references this column(s).

Special considerations for CALL DML tables

The ALTER TABLE statement for CALL DML tables must take the following restrictions into account:

- Only the ADD [COLUMN], DROP [COLUMN] and ALTER [COLUMN] clause are permitted with SET *data_type*.
- A newly inserted column must include a CALL DML clause.
- Only the data types CHARACTER, NUMERIC, DECIMAL, INTEGER and SMALLINT are permitted.
- No integrity constraint or default value (DEFAULT) can be defined for the column.
- The column name must be different to the integrity constraint name of the table constraint since this name is used as the name of the compound primary key.
- A column's data type in a CALL DML table may only be changed to the data type of a CALL DML/SQL table.In particular, a CALL DML table's data type must not be changed to an "old attribute format", i.e. to an attribute format of SESAM version < 13.1.
- An "old attribute format" in a CALL DML only table can be changed to the following data types:
  - CHARACTER with $new\_length \geq old\_length$
  - NUMERIC with *old_fraction=new_fraction*
  - DECIMAL with *old_fraction=new_fraction*
  - INTEGER
  - SMALLINT

– You can assign a new non-significant attribute value for columns in a CALL DML table. You may not change the symbolic attribute name.

– If a data type modification results in a value in a CALL DML column receiving the non-significant attribute value, the value of the column in question is considered to be non-convertible. If no exception file was specified, SESAM/SQL issues an error message and aborts the statement. If an exception file is specified, SESAM/SQL reacts as in the case of non-convertible values in an SQL table (see page 94).

– You can neither use the ALTER [COLUMN] clause nor the DROP [COLUMN] clause to change the table type. Even if the columns in a CALL DML only table have been changed or deleted so that none of the columns contains an “old attribute format“, the “CALL DML only” table type remains unchanged. You can change the table type by means of the utility statement MIGRATE (see the “SESAM/SQL-Server - SQL Reference Manual, Part 2” [7]).

Converting “old” attributes in a CALL DML only table

The attribute of a CALL DML only table has no explicit type: the type is simply specified by the way the table is saved. The user must interpret the values correctly.

You cannot use ALTER COLUMN to change the type, but only to transfer it to the specified type. When you do this, values of the corresponding type are transferred and those of different types are rejected (SQLSTATE 22SA5).
You should therefore only specify the appropriate type. Conversion to another type is only possible if you use a second ALTER COLUMN and specify the new data type.
For example, a binary value can only be changed to INTEGER, SMALLINT. After a second ALTER COLUMN you can also convert it to NUMERIC, DECIMAL and CHARACTER.

ALTER COLUMN reads each value and prepares it in accordance with its definition in the CALL DML table. Alignment, fill bytes etc. are not taken into account. However, no conversion is performed. After that, a check is performed to determine whether the read value corresponds to the specified format or not.

Since the attribute of the CALL DML also contains values of different types, it is advisable to always specify USING FILE *exception_file* for “old” attributes when using ALTER COLUMN. All inappropriate values are then entered in the exception file.

If no exception file is present, ALTER COLUMN aborts when the first inappropriate value is encountered.

Exception file of SQL statement ALTER TABLE

When you modify a column (ALTER COLUMN), you can specify the name of an exception file. If necessary, you can protect the exception file using a BS2000 password. The exception file is used to store column values for which conversion errors resulted in data loss because of a change of data type.

If you have specified an exception file and conversion errors occur during the modification of the data type, SESAM/SQL sets up the exception file as a SAM file under the DBH user ID if this does not yet exist.

If the exception file is not to be stored under the DBH user ID, you must use the SDF command CREATE-FILE to set up the file as a SAM file under the required ID (the BS2000 user id of the DBH must possess the necessary access rights).

If an exception file is specified, statements which result in a conversion error are not aborted. SESAM/SQL issues an alert and replaces the original column value by a new value in the affected base table. Depending on the error type, the value is truncated or replaced by the NULL value.

SESAM/SQL logs the original column values together with the associated error message or alert in the exception file. If an exception file exists, its contents are not overwritten. SESAM/SQL appends the new entries to the existing entries.

The exception file is not subject to transaction logging. It remains intact, even if the transaction which SESAM/SQL uses to write entries to the exception file is implicitly or explicitly rolled back.

You can display the contents of the exception file using the SDF command SHOW-FILE.

Contents of the exception file

The exception file contains an entry for each logged column value. The entry consists of the corresponding SQL status code and the components which identify the column value within the associated base table.

---

entry :: =

row_id
column_name [posno]
sql_state
column_value

row_id :: = { *primary_key* | *row_counter* }

---

row_id

Identifies the table rows which contains the *column_value*.
In tables with primary keys, *row_id* is the primary key value which uniquely identifies the corresponding row. Its representation in the error file corresponds to the representation of *column_value* (see under the appropriate information). The same applies to the compound keys.

In tables without primary key, *row_id* is the counter of the row containing *column_value*. SESAM/SQL numbers all table rows sequentially. The first row in the table contains the value 1 as *row_counter*.
row_counter is an unsigned integer.

column_name

Name of the column containing to the *column_value*. In multiple columns, *column_name* also contains the position number, in unsigned integer format, of the affected column element. The first element of the multiple column has the position number 1.

sql_state

SQLSTATE of the associated error message or alert.

column_value

Original column value for which the ALTER TABLE statement resulted in a conversion error.
Depending on the data type of the associated, *column_value* is represented in the following ways in the exception file:

Data type of column containing the original value	**Representation of *column_value* in the exception file**
Data type of a CALL DML table column of SESAM version ≤ 13.1	string with a maximum length of 54 characters
CHARACTER CHARACTER VARYING	string with a maximum length of 54 characters
INTEGER, SMALLINT, NUMERIC, DECIMAL,	corresponding numeric literal (integer or fixed point number)
FLOAT, REAL, DOUBLE PRECISION	corresponding numeric literal (floating point number)
DATE	date time literal
TIME	time time literal
TIMESTAMP	time stamp time literal

Strings are represented without surrounding single quotes in the exception file.

If the original value is a string which contains double quotes, these are represented as single quotes in the exception file.

Example

The following example illustrates an exception file which contains the original column value of the base table “convert“.

The “convert” base table has the following structure:

```
CREATE TABLE CONVERT
( PKEY1             CHAR(1)
 ,PKEY2             SMALLINT
 ,PKEY3_UNIQUE1_REF SMALLINT
.
.
 ,DATE_CHAR        CHAR(10)
    DEFAULT '1994-05-02'
    CONSTRAINT INTEG002 NOT NULL
.
.
 ,CONSTRAINT PK001 PRIMARY KEY
    (PKEY1,PKEY2,PKEY3_UNIQUE1_REF)
)
```

This entries are the result of conversion errors which were caused by the following statements:

```
ALTER TABLE CONVERT -
ALTER COLUMN DATE_CHAR DROP DEFAULT, -
DATE_CHAR SET NUMERIC(10,2) -
USING FILE 'O.CONVERTEST'
```

Excerpt from the exception file O.CONVERTEST:

row_id [1]	*column_name*	*sql_state*	*column_value*
A,2,1	DATE_CHAR	22SA5	1994-05-02
A,3,2	DATE_CHAR	22SA5	1994-05-02
A,4,3	DATE_CHAR	22SA5	1994-05-02
.			
.			
A,23,22	DATE_CHAR	22SA5	1994-05-02
A,24,23	DATE_CHAR	22SA5	1994-05-02

[1] This is a compound key which consists of the fields PKEY1, PKEY2 and PKEY3_UNIQUE1_REF

When DATE_CHAR is converted from CHAR(10) to NUMERIC(10,2), the value 1994-05-02 cannot be converted in the rows for which the compound key has been specified.

Example

The example below deletes the not NULL integrity constraint on the column `company` of the `customers` table. The name of the integrity constraint is in the CHECK_CONSTRAINTS view of the INFORMATION-SCHEMA.

```
ALTER TABLE customers
   DROP CONSTRAINT customers.company_notnull RESTRICT
```

See also

CREATE TABLE

CREATE INDEX - Create index

You use CREATE INDEX to generate an index for a base table. SESAM/SQL can use the index to evaluate constraints on one or more columns of the index without accessing the base table or to output the rows in the table in the order of the values in the index column(s).

The restrictions and special considerations that apply to CALL DML tables are described in “Special considerations for CALL DML tables” on page 104.

The current authorization identifier must own the schema to which the base table belongs.

If you specify the space for the index, the actual authorization identifier must own the space.

```
CREATE INDEX { index ({ column [ LENGTH length ] },...) }
       ON TABLE table [ USING SPACE space ]
```

---

index ({ *column* [LENGTH *length*] },...)
Definition of an index.

If you create an index for only one column, the column cannot be longer than 256 characters. If you create an index involving several columns, the sum of the column lengths plus the total number of columns cannot exceed 256.

index
Name of the new index. The unqualified index name must be unique within the schema. You can qualify the index name with a database and schema name. The database and schema name must be the same as the database and schema name of the base table for which you are creating the index.
If you use the CREATE INDEX statement in a CREATE SCHEMA statement, you can only qualify the index name with the database and schema name from the CREATE SCHEMA statement.

column
Name of the column in the base table you want to index.

A column cannot occur more than once in an index. You can create an index that applies to several columns (compound index). In this case, the index cannot apply to multiple columns.

LENGTH *length*
Indicates the length up to which the column is to be indexed. *length* must be an unsigned integer between 1 and the length of the column. You can only limit the length if the column is of the following data type: CHARACTER, VARCHAR or data types from SESAM ≤ V12.

LENGTH *length* omitted:
The column in its entirety is indexed.

ON TABLE *table*
Name of the base table you are indexing.

If you qualify the table name with a database and schema name, this must be the same as the database and schema name of the index.

If you use the CREATE INDEX statement in a CREATE SCHEMA statement, you can only qualify the table name with the database and schema name from the CREATE SCHEMA statement.

USING SPACE *space*
Name of the space in which the index is to be stored.

You can qualify the space name with the database name. This database name must be the same as the database name of the base table.

The space must already be defined for the database to which the table belongs. The current authorization identifier must own the space.

USING SPACE *space* omitted:
The index is stored in the space for the base table.

Special considerations for CALL DML tables

The CREATE INDEX statement for CALL DML tables must take the following restrictions and special considerations into account:

– Every index can only apply to one column.

– Each column can only occur once in an index.

– You can only specify the name of the primary key constraint of a database with a compound key as the column name in the index. This means that the primary key is indexed.

Indexes and integrity constraints

If you define a UNIQUE constraint for a table, the columns specified in the UNIQUE constraint are implicitly indexed. If you explicitly define an index with CREATE INDEX that applies to the same columns, the implicitly defined index is deleted. The explicit index is then also used for the integrity constraint.

Example

The example below creates an index for the column `company` in the `customers` table.

```
CREATE INDEX ind1 (company)
    ON TABLE customers
    USING SPACE my_space
```

See also

DROP INDEX

CREATE SCHEMA - Create schema

You use CREATE SCHEMA to create a schema. At the same time you can define tables, views, privileges and indexes. You can also modify the schema later with the appropriate CREATE, ALTER and DROP statements.

The current authorization identifier must have the special privilege CREATE SCHEMA.

```
CREATE SCHEMA { schema [ AUTHORIZATION authorization_identifier ] |
                AUTHORIZATION authorization_identifier
              }

              [ { create_table_statement |
                  create_view_statement |
                  create_index_statement |
                  grant_statement
                } ...
              ]
```

---

schema
Name of the schema. The unqualified schema name must be unique within the database. You can also qualify the schema name with a database name.

schema omitted:
The name of the authorization identifier in the AUTHORIZATION clause is used as the schema name.

AUTHORIZATION *authorization_identifier*
The authorization identifier owns the schema.

This authorization identifier is used as the name of the schema if you do not specify a schema name.

AUTHORIZATION *authorization_identifier* omitted:
If an authorization identifier has been defined for the compilation unit, it owns the schema. Otherwise, the current authorization identifier is the owner.

create/grant_statements
If you use unqualified table and index names in the CREATE and GRANT statements, the names are automatically qualified with the database and schema name of the schema.

create_table_statement
CREATE TABLE statement that creates a base table for the schema.

create_view_statement
CREATE VIEW statement that creates a view for the schema.

grant_statement
GRANT statement that grants privileges for a base table or a view. You cannot grant special privileges with the GRANT statement.

create_index_statement
CREATE INDEX statement that creates and index for the schema.

create/grant_statements omitted:
An empty schema is created.

How CREATE SCHEMA functions

The CREATE TABLE, CREATE VIEW, GRANT and CREATE INDEX statements that are specified in the CREATE SCHEMA statement are executed in the order in which they are specified. You must therefore place statements that reference existing tables or views after the statement that creates these tables or views.

Example

The example below creates the schema `andromeda`, defines a table and grants privileges for the schema.

```
CREATE SCHEMA andromeda

CREATE TABLE telephone_list

   (name CHARACTER (25),
   telephone CHARACTER (15),
   fax CHARACTER (15))

GRANT ALL PRIVILEGES ON telephone_list TO hugh
```

See also

CREATE TABLE, CREATE VIEW, CREATE INDEX, GRANT, DROP SCHEMA

CREATE SPACE - Create space

You use CREATE SPACE to create a new entry for a new user space in the database catalog and to generate the corresponding file at operating system level.

You can define up to 199 user spaces for a database.

The current authorization identifier must have the special privilege USAGE for the storage group used.

If the database catalog was created with a separate DB user ID, you must use the SDF command CREATE-FILE to create the user space files for this database before calling the SQL statement CREATE SPACE. Here it is important that you create the user space files under the database user ID *shareable* and assign write permissions to it. If the file of the catalog space was created with a password, you must also specify a password for the user space files. The password must be identical to the BS2000 password for the catalog space file.

```
CREATE SPACE space
          [ AUTHORIZATION authorization_identifier ]
          [ { PRIMARY allocation |
              SECONDARY allocation |
              PCTFREE percent |
              [NO] SHARE |
              [NO] DESTROY |
              NO LOG
            } ...
          ]
          [ USING STOGROUP stogroup ]
```

---

space
Name of the space. The first 12 characters of the unqualified space name must be unique within the database. You can qualify the space name with the database name.

AUTHORIZATION *authorization_identifier*
Name of the authorization identifier to be entered as the owner of the space.

AUTHORIZATION *authorization_identifier* omitted:
The current authorization identifier is entered as the owner.

You can only specify the following parameters once: PRIMARY, SECONDARY, PCTFREE, [NO] SHARE, [NO] DESTROY and NO LOG.

PRIMARY *allocation*
: Primary allocation of the space file in units of 2K (BS2000 halfpage). *allocation* must be an unsigned integer between 1 and 8388607.

  PRIMARY *allocation* omitted:
  PRIMARY 24 is used.

SECONDARY *allocation*
: Secondary allocation of the space file in units of 2K (BS2000 halfpage). *allocation* must be an unsigned integer between 1 and 32767.

  SECONDARY *allocation* omitted:
  SECONDARY 24 is used.

PCTFREE *percent*
: Free space reservation in the space file expressed as a percentage. *percent* must be an unsigned integer between 0 and 70.

  PCTFREE *percent* omitted:
  PCTFREE 20 is used.

[NO] SHARE
: SHARE indicates that the space file is sharable, i.e. that the space file can be accessed from more than one BS2000 user ID of the DBH.
  NO SHARE indicates that the space file is not sharable.

  NO SHARE is recommended for security reasons.

  [NO] SHARE omitted:
  NO SHARE is used.

[NO] DESTROY

: DESTROY indicates that when the space file is deleted the storage space is to be overwritten with binary zeros.
  NO DESTROY means that when the space file is deleted, just the storage space is released.

  [NO] DESTROY omitted:
  DESTROY is used.

NO LOG
: No logging.

  NO LOG omitted:
  The logging setting for the database is used.

USING STOGROUP *stogroup*

Name of the storage group containing the volumes to be used for creating the space file.

If you specify the unqualified name of the storage group, the name is automatically qualified with the database name of the schema. If you qualify the name of the storage group with a database name, this name must be the same as the database name of the space.

USING STOGROUP *stogroup* omitted:
The default storage group D0STOGROUP is used.

Space file at operating system level

The space file is created either under the BS2000 user ID under which the DBH is running or under a separate DB user ID. In the first case, the file is created by DBH; in the latter it must be created by the database administrator using CREATE-FILE.

The space file has the following name:

:*catid*:{ *dbh* | *db* }*_bs2000_userid.database.unqual_spacename*

Only the first 12 characters of the unqualified space name are used for the file name.

Example

The example below creates a space file using the default primary and secondary allocation settings. The file is to be sharable and is not to be overwritten with binary zeros when it is deleted.

```
CREATE SPACE my_space SHARE NO DESTROY
```

See also

ALTER SPACE, CREATE STOGROUP

CREATE STOGROUP - Create storage group

You use CREATE STOGROUP to create a new storage group. A storage group describes either a PVS (Public Volume Set) or a set of private volumes. The private volumes in a storage group must all have the same data type (see also “SESAM/SQL-Server - Core Manual” [8]).

The storage group D0STOGROUP always exists.

The current authorization identifier must have the special privilege CREATE STOGROUP.

```
CREATE STOGROUP stogroup
          { VOLUMES (volume_name,...) ON dev_type | PUBLIC }
          [ ON catid ]
```

---

stogroup
Name of the storage group. The unqualified name of the storage group must be unique within a database. You can qualify the name of the storage group with a database name.

The current authorization identifier will own the storage group and is granted the special privilege USAGE for this storage group.

VOLUMES (*volume_name*,...)
The storage group is created on private volumes. v*olume_name* is an alphanumeric literal indicating the VSN of the volumes. You can only specify each VSN once, and you can specify up to 100 volumes.
All the volumes in a storage group must have the same device type.

ON *dev_type*
Device type of the private volumes.

PUBLIC
The storage group comprises a public volume set (PVS).

ON *catid*
Alphanumeric literal indicating the *catid*.

If you specify PUBLIC, this is the catalog ID of the PVS on which the storage group is defined and on which the files are created. In the case of private volumes (VOLUMES), this is the PVS on which the files are cataloged. The files themselves are located on the specified private volumes.

ON *catid* omitted:
The catalog ID assigned to the BS2000 user ID under which the DBH is running is used.

Example

The example below creates a new storage group `abraxas` with the specified private volumes. The catalog ID `P` is used for cataloging the space files created on the storage group.

```
CREATE STOGROUP abraxas
    VOLUMES ('DY130A','DY130B','DY130C','DY130D') ON 'D3475'
    ON 'P'
```

See also

DROP STOGROUP

CREATE SYSTEM_USER - Create system entry

You use CREATE SYSTEM_USER to define a system entry, i.e. assign authorization identifiers to the system users. You can assign an authorization identifier to more than one user, and a single user may have more than one authorization identifier.

A local UTM system user is identified by the local host name, the local UTM application name and the UTM user ID.

A UTM system user working with SESAM databases via UTM-D is identified by the local host name, the local UTM application name and the local UTM session name (LSES).

A BS2000 (TIAM) system user is identified by the host name and the BS2000 user ID.

Please note that before you move a database to another system, you must first define a valid system entry for the new system.

The current authorization identifier must have the special privilege CREATE USER. If you want to assign a system user an authorization identifier with the special privilege CREATE USER and with GRANT authorization (see “GRANT - Grant privileges” on page 136), the current authorization identifier must also have GRANT authorization.

```
CREATE SYSTEM_USER { utm_user | bs2000_user }
          FOR authorization_identifier
          AT CATALOG catalog

utm_user::=( { hostname|* },{ utm_application_name|* },{ utm_userid|* } )
bs2000_user::=( { hostname|* },[*],{ bs2000_userid|* } )
```

---

utm_user
Defines a system entry for a UTM system user.

hostname
Alphanumeric literal indicating the symbolic host name.

If DCAM is not available on the host, the host is assigned the name “HOMEPROC”.

For UTM-D: Specification of the local host on which the SESAM/SQL database connection was generated.

* All hosts.

utm_application_name
Alphanumeric literal indicating the name of the UTM application.
For UTM-D: Name of the local UTM application.

* All UTM applications.

utm_userid
You specify the UTM user ID as an alphanumeric literal defined with KDCSIGN for local UTM system users. For UTM-D, you specify the local UTM session name (LSES).

* All UTM user IDs.

bs2000_user
Defines a system entry for a BS2000 system user.

hostname
Alphanumeric literal indicating the symbolic host name.
If DCAM is not available on the host, the host is assigned the name “HOMEPROC”.

* All hosts.

bs2000-userid
Alphanumeric literal indicating the BS2000 user ID.

* All BS2000 user IDs.

FOR *authorization_identifier*
Name of the previously defined authorization identifier to be assigned to the system user.

AT CATALOG *catalog*
Name of the database for which the assignment of an authorization identifier to a system user is valid.

Example

In the example below, two previously defined authorization identifiers are assigned to system users.

```
CREATE SYSTEM_USER (*,,'purchasing') FOR hugh AT CATALOG my_db
CREATE SYSTEM_USER (*,,'sales') FOR bertha AT CATALOG my_db
```

The authorization identifier `hugh` can access the database `my_db` from the BS2000 user ID `purchasing`, whereas the authorization identifier `bertha` can access the database `my_db` from the BS2000 user ID `sales`.

See also

DROP SYSTEM_USER, CREATE USER

CREATE TABLE - Create base table

You use CREATE TABLE to create a base table in which the data is permanently stored. SESAM/SQL distinguishes between

– SQL tables that can only be processed with SQL.

– CALL DML/SQL tables that can be processed with CALL DML and to some extent with SQL.

– CALL DML only tables that can only be processed with CALL DML. These CALL DML tables cannot be created with CREATE TABLE. They are created with the MIGRATE statement (see the “SESAM/SQL-Server - SQL Reference, Part 2” [7]).

CALL DML only tables and CALL DML/SQL tables are referred to by the term CALL DML tables.

The restrictions that apply when you use CREATE TABLE to create CALL DML tables are described in “Special considerations for CALL DML tables” on page 117.

The current authorization identifier must own the schema. If you specify the space for the base table, the current authorization identifier must own the space.

```
CREATE [CALL DML] TABLE table
    ( { column_definition | [CONSTRAINT integrity_constraint_name] table_constraint }, ...)
    [USING SPACE space]
```

CALL DML
: Creates a CALL DML table.
  You can only process CALL DML tables with SESAM CALL DML. The column definitions and integrity conditions must observe certain restrictions (see “Special considerations for CALL DML tables” on page 117).

  CALL DML omitted:
  An SQL table is created.
  SQL tables can only be processed with SQL.

TABLE *table*

Name of the new base table. The unqualified table name must be different from all the base table names and view names in the schema and must be different from the unqualified name of a temporary views if such a name is still present[1]. You can qualify the table name with a database and schema name.

If you use the CREATE TABLE statement in a CREATE SCHEMA statement, you can only qualify the table name with the database and schema name from the CREATE SCHEMA statement.

column_definition

Defines columns for the base table.

You must define at least one column. A base table can have up to 26 134 columns of any type except VARCHAR and up to 1000 columns of the type VARCHAR.

The current authorization identifier is granted all table privileges for the defined columns.

CONSTRAINT *integrity_constraint_name*

Assigns an integrity constraint name to the table constraint. The unqualified name of the integrity constraint must be unique within the schema. You can qualify the name of the integrity constraint with a database and schema name. The database and schema name must be the same as the database and schema name of the base table for which the integrity condition is defined.

CONSTRAINT *integrity_constraint_name* omitted:

The integrity constraint is assigned a name according to the following pattern:

{ UN | PK | FK | CH } *integrity_constraint_number*

where UN stands for UNIQUE, FK for FOREIGN KEY and CH for CHECK. *integrity_constraint_number* is a 16-digit number.

table_constraint

Defines an integrity constraint for the base table.

USING SPACE *space*

Name of the space in which that table is to be stored. The space must already be defined for the database to which the table belongs. You can qualify the space name with the database name. This database name must be the same as the database name of the base table.

USING SPACE *space* omitted:
The table is stored in the default space of the current authorization identifier on the storage group D0STOGROUP.

[1] Temporary views are no longer supported as of SESAM/SQL V3.0.

The default space is D0*authorization_identifier* with the first 10 characters of the authorization identifier. If this space does not yet exist, it is created if the current authorization identifier has been granted the special privilege USAGE for the storage group D0STOGROUP.

Special considerations for CALL DML tables

The CREATE TABLE statement for CALL DML tables must take the following restrictions into account:

- Only the data types CHARACTER, NUMERIC, DECIMAL, INTEGER and SMALLINT are permitted.
- No default value can be defined for the column with DEFAULT.
- A column that is not a primary key must have a CALL DML clause.
- The table must contain exactly one primary key restraint as the column or table constraint.
- The table constraint defines a compound primary key and must be given a name that corresponds to the name of the compound primary key in SESAM/SQL V1.x.
- The column name must be different from the integrity constraint name of the table constraint since this name is used as the name of the compound primary key.

Example

1. The example below shows the CREATE TABLE statement for the `orders` table in the demonstration database.

```
CREATE TABLE orders

   (order_num   INTEGER CONSTRAINT order_num_primary PRIMARY KEY,
   cust_num     INTEGER CONSTRAINT o_cust_num_notnull NOT NULL,
   contact_num  INTEGER,
   order_date   DATE DEFAULT CURRENT_DATE,
   order_text   CHARACTER (30),
   actual       DATE,
   target       DATE,
   order_stat   INTEGER DEFAULT 1 CONSTRAINT order_stat_notnull NOT NULL,
   CONSTRAINT o_cust_num_ref_customers FOREIGN KEY (cust_num)
      REFERENCES customers,
   CONSTRAINT contact_num_ref_contacts FOREIGN KEY (contact_num)
      REFERENCES contacts,
   CONSTRAINT order_stat_ref_ordstat FOREIGN KEY (order_stat)
      REFERENCES ordstat(order_stat_num)
   )
```

2. This example shows the CREATE TABLE statement for the CALL DML table `company` in the schema `companykey` of the database `callcompany` (see the manual “SESAM/SQL-Server - CALL DML Applications” [11]).

```
CREATE CALL DML TABLE callcompany.companykey.company

   (key           CHARACTER(006) PRIMARY KEY,
   item_name      CHARACTER(015) CALL DML ' ' AA8,
   item_price     NUMERIC(05,02) CALL DML -0 AB6,
   item_stock     NUMERIC(04) CALL DML -0 AC4,
   cust_surname   CHARACTER(015) CALL DML ' ' AD2,
   cust_firstname CHARACTER(012) CALL DML ' ' AEZ,
   cust_street    CHARACTER(015) CALL DML ' ' AFX,
   cust_zip       CHARACTER(005) CALL DML ' ' AGV,
   cust_city      CHARACTER(015) CALL DML ' ' AHT,
   cust_since     CHARACTER(006) CALL DML ' ' AJR,
   cust_discount  NUMERIC(04,02) CALL DML 0 AKP,
   .
   .
   .
   p_salary (010) NUMERIC(07,02) CALL DML 0 AT5)
   USING SPACE CALLCOMPANY.COMPANY
```

See also

ALTER TABLE, CREATE SCHEMA, CREATE SPACE

CREATE USER - Create authorization identifier

You use CREATE USER to create a new authorization identifier.

The current authorization identifier must have the special privilege CREATE USER.

```
CREATE USER authorization_identifier
       AT CATALOG catalog
```

---

authorization_identifier
Name of the authorization identifier. The first 10 characters of the authorization identifier must be unique within the database.

AT CATALOG *catalog*
Name of the database for which the authorization identifier is to be valid.

Example

The example below defines two authorization identifiers for `my_db`.

```
CREATE USER bertha AT CATALOG my_db
CREATE USER hugh AT CATALOG my_db
```

See also

DROP USER, CREATE SYSTEM_USER

DECLARE - Declare cursor

You use DECLARE to define a cursor. You can use the cursor to access the individual rows in a derived table. The current row on which the cursor is positioned can be read. If the cursor is updatable, you can also update and delete rows.

A static cursor declaration must physically precede any statement that uses the cursor in the program text. All the statements that use this cursor must be located in the same compilation unit.

The DECLARE statement for a static cursor is not an executable statement. This means that the statement declaring a static cursor is only permitted at the beginning of the program, i.e. before the processing statements.

You must specify FOR *cursor_description* if you are declaring a dynamic cursor (see the EXECUTE statement in the “DRIVE Directory” [3]).

In DRIVE, up to 20 dynamic and variable cursors are permitted. The DRIVE program is aborted if more cursors are declared. You can prevent the program from being aborted with WHENEVER &DML_STATE IN ('TOO MANY CURSORS') (see WHENEVER and the “DRIVE Programming Language” manual [2], section 3.1.2, “System variables”).

Scope of validity of a cursor:

A cursor ceases to be valid
- when the program is terminated (the life of a cursor ends when the application or transaction is terminated)
- if the program is aborted
- when DRIVE is terminated (STOP)
- if DROP CURSOR *cursor* (DRIVE statement, only in interactive mode, dynamically or for variable cursors).
- DROP CURSORS (DRIVE statement, only in interactive mode or dynamically)

When you switch from DRIVE interactive mode to program mode, the cursor definition remains valid, but the cursor position is lost because no transaction can be open when this switch takes place. You can, however, save the cursor position with STORE, provided that the cursor involved is not a PREFETCH cursor.

In DRIVE program mode, a cursor defined with PERMANENT remains valid beyond the end of a program invoked with CALL, and its position is retained.
A cursor defined with TEMPORARY is closed when the program is terminated and deleted if a COMMIT WORK statement is issued on a higher program level.

A cursor always ceases to be valid in program mode when you switch to interactive mode, if the program is aborted, and when DRIVE is terminated (STOP or COMMIT WORK WITH STOP).

```
DECLARE cursor [ { PERMANENT | TEMPORARY } ]
               [ SCROLL ] CURSOR
               [ PREFETCH n ]
               [ FOR cursor_description ]

cursor_description::= query_expression [ ORDER BY { column | column(pos_no) | column_no }
                                   [ { ASCENDING | DESCENDING } ] ]
                                 [ FOR { UPDATE [OF column,...] | READ ONLY } ]
```

cursor
Name of the cursor. You cannot define more than one cursor with the same name within a compilation unit. The scope of validity of the cursor is limited to the compilation unit in which the cursor is defined.

PERMANENT
PERMANENT can only be specified within programs called using CALL.
The position of the cursor is retained after a program invoked with CALL is terminated, provided that no COMMIT WORK statement was executed in the called program or in the calling program between the CALLs.
When the program is invoked with CALL runs for the first time, the cursor must be opened with the OPEN statement. Whenever it executes subsequently, no OPEN statement may be issued for that cursor.
The calling program must not contain a COMMIT WORK statement.

TEMPORARY
TEMPORARY is the default value.
TEMPORARY can only be specified in programs called using CALL.
The cursor is closed at the end of the CALLed program and its position is lost (end of the scope of validity of the cursor). The cursor is deleted (end of the life of the cursor) when the next COMMIT WORK statement is issued at a higher program level.

SCROLL
You can position the cursor on any row in the derived table and in any order with FETCH NEXT/PRIOR/FIRST/LAST/RELATIVE/ABSOLUTE.

You can only specify SCROLL if no FOR UPDATE clause was defined in the cursor description of *cursor*.

If you specify SCROLL, *cursor* cannot be changed. The FOR READ ONLY clause applies implicitly.

SCROLL omitted:
You can only position the cursor on the next row. In FETCH, only the position specification NEXT is permitted.

PREFETCH *n*

The PREFETCH clause increases performance by activating block mode.

Instead of the PREFETCH clause, you can also use a PRAGMA statement with the pragma clause PREFETCH to activate block mode (see “PRAGMA - Declare pragma clauses” on page 141). Both ways of activating block mode are functionally equivalent to each other. A prerequisite for its use is that you are working with a SESAM/SQL Version 2.1 or higher.

If you are working with SESAM/SQL V2.0, use of the PREFETCH clause or PREFETCH pragma will result in SESAM errors (SQLSTATE class 01).

A cursor for which block mode is activated using one of the above-mentioned methods is referred to as a **PREFETCH cursor**.

The following statements are not permitted for a PREFETCH cursor:
FETCH PRIOR, FIRST, LAST, RELATIVE, ABSOLUTE (only positioning with FETCH NEXT is permitted)
STORE and RESTORE
DELETE... WHERE CURRENT OF...
UPDATE... WHERE CURRENT OF...

n

Block factor *n*-1 indicates the number of records that SESAM is to read into a buffer when the first FETCH statement after OPEN is executed (block). A subsequent FETCH statement does not have to access the database. *n* is an integer of the type SMALLINT. *n* must be greater than or equal to 2 and less than or equal to 32000. If l is the sum of the lengths of all the selected row elements, then n * l should be less than or equal to 30000. If you are outputting to the screen, the number of rows that can be displayed on a screen can be used as a guideline. Depending on l, less than *n*-1 rows may be read into the block buffer.

FOR clause

The FOR clause can only omitted in program mode in a static cursor declaration. If it is omitted, a **variable cursor** is declared. This type of cursor is not made known to the database system until a subsequent dynamic declaration with a FOR clause is executed. In the case of a dynamic declaration with EXECUTE, you must specify a FOR clause. Except for the FOR clause, the static and dynamic declarations for a cursor must be the same. The block factor *n* within a PREFETCH clause can, however, vary. You can specify any of the other cursor statements (OPEN, FETCH, CLOSE, DROP CURSOR, STORE, RESTORE, UPDATE, DELETE, CYCLE) statically for the variable cursor. This leads to an improvement in performance (see the “DRIVE Programming Language” manual [2], section 4.6.1, “Dynamic SQL statements”).

cursor_description
Declares a static or dynamic cursor.

cursor_description defines the derived table and the attributes of the cursor. The earliest point at which a row in the derived table can be selected is when you open the cursor with OPEN. The latest point at which a row can be selected is when you execute a FETCH statement.

query_expression
Query expression for selecting rows and column from base tables or views.

The values of the host variables in *query_expression* are not determined until the cursor is opened. The literals CURRENT_USER and SYSTEM_USER and time functions that are used in *query_expression* are not evaluated until the cursor is opened.

ORDER BY
The ORDER BY clause indicates the columns according to which the derived table is to be sorted. The rows are sorted according to the values in the column specified first. If two or more rows have the same values in that column according to the comparison rules, these rows are sorted according to the values in the second sort column and so on. In SESAM/SQL, NULL values are considered smaller than all non-NULL values for sorting purposes.

The order of rows with the same value in all the sort columns is undefined.

You can only specify ORDER BY if no FOR UPDATE clause was declared for the cursor description of *cursor*.

If you specify ORDER BY, *cursor* cannot be changed. The FOR READ ONLY clause applies implicitly.

ORDER BY omitted:
The order of the rows in the cursor table is undefined.

column
Name of the column according to which the table is to be sorted. The column must be part of the derived table created by *query_expression*.

You can specify an atomic column for *column*. The column name cannot be qualified with a table specification.

column(*pos_no*)
Element of a multiple column that is to be taken as the basis for the sorting operation. The column element must be part of the derived table created by *query_expression*.

pos_no is an unsigned integer indicating the position number of the column element in the multiple column.

column_number
Number of the column to be used as the basis for sorting.

column_number is an unsigned integer where

1 ≤ *column_number* ≤ number of derived columns.

By specifying a column number, you can also use columns that do not have a name, or which do not have a unique name, as the basis for sorting.

column_number can be an atomic column or a multiple column with the dimension 1.

ASCENDING
The values in the column involved are sorted in ascending order.

DESCENDING
The values of the column involved are sorted in descending order.

FOR READ ONLY
The FOR READ ONLY clause specifies that *cursor* can only be used to read the records of the derived table (read-only cursor).

If the relevant query expression is not updatable, the FOR READ ONLY clause applies implicitly (see the section “Updatability of query expressions” in the “SESAM/SQL-Server - SQL Reference Manual , Part 1” [6]). It also applies if SCROLL or ORDER BY is specified in the cursor declaration.

FOR UPDATE
You can only use the FOR UPDATE clause if the relevant query expression is updatable (see the section “Updatability of query expressions” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]) and neither SCROLL nor ORDER BY was specified. You use a FOR UPDATE clause to specify which columns in the underlying table can be updated via the cursor with UPDATE...WHERE CURRENT OF.

If a PREFETCH pragma has been defined for a cursor, the FOR UPDATE clause disables this pragma (see “PREFETCH pragma clause” on page 148).

FOR UPDATE omitted:
If the cursor is updatable (see below) and the FOR READ ONLY clause is not specified, you can update all the columns of the underlying table with UPDATE...WHERE CURRENT.

OF *column*,...
Only the specified columns can be updated with UPDATE...WHERE CURRENT OF. For *column*, specify the name of a column in the table that the updatable cursor references. *column* is the unqualified name of the column in the underlying table, regardless of whether a new column name was defined in the query expression of the cursor description.

Example

In the example below, an updatable cursor `cur` is declared. The underlying table is `tab`. Only column `col` in table `tab` can be updated via cursor `cur`. To do this, a FOR UPDATE clause with the column name `col` is specified in the cursor description.

```
DECLARE cur CURSOR FOR

    SELECT corr.col AS column FROM tab AS corr

    FOR UPDATE OF col
```

The unqualified, original column name `a` is used in the FOR UPDATE clause although the column is renamed in the SELECT list and the table is renamed in the FROM clause.

OF *column,...* omitted:
Each column in the underlying table can be updated with UPDATE ... WHERE CURRENT OF.

Updatable cursor

Only updatable cursors can be used with the UPDATE... WHERE CURRENT OF... or DELETE... WHERE CURRENT OF statements to perform updates or deletions. A cursor is updatable if its cursor description is updatable, i.e. the underlying query expression is updatable, and no ORDER BY clause is specified (see metavariable *query_expression*). No SCROLL clause can be specified in the cursor declaration. If you specify the PREFETCH clause, an updatable cursor cannot be used for updating or deleting.

Example 1

This example is a cursor declaration with a variable cursor description. A static cursor is specified with a DECLARE ... CURSOR statement without a FOR clause.

```
DECLARE cur_displ SCROLL CURSOR;
```

An EXECUTE statement is used to declare the cursor description dynamically at execution time. (The expression in quotes can be up to 256 characters long, otherwise you will have to use CONCAT. DRIVE includes the blanks for indenting the code in the count.)

```
EXECUTE 'DECLARE cur_disp SCROLL CURSOR FOR '||
              'SELECT country, last_name, first_name, salary '||
        'FROM db_employee '||
        'WHERE (country = &hcountry) ' ||
               ' AND (salary >= &s_lower) AND (salary <= &s_upper);';
```

All statements that reference the cursor can be specified statically in the program.

```
CYCLE cur_disp INTO &disp_row.*;
DISPLAY FORM LINE &disp_row;
END CYCLE;
DROP CURSOR cur_disp;
...
COMMIT WORK;
EXECUTE 'DECLARE cur_update CURSOR FOR ' ||
        'SELECT country, last_name, first_name, salary ' ||
         'FROM db_employee ' ||
         'WHERE (country = &hcountry) AND
               (salary = &highest);';
```

A COMMIT WORK should be included between the DROP statement and an EXECUTE 'DECLARE ...' statement on the same cursor name (in accordance with static cursor declaration without a FOR clause).

Example 2

An updatable cursor `cur` is declared. The underlying table is `tab`. Only column `col` in table `tab` can be updated via cursor `cur`. To do this, a FOR UPDATE clause with the column name `col` is specified in the cursor description.

```
DECLARE cur CURSOR FOR
   SELECT corr.col AS column FROM tab AS corr
   FOR UPDATE OF col;
```

The unqualified, original column name `col` is used in the FOR UPDATE clause although the column is renamed in the SELECT list, and the table is renamed in the FROM clause.

Example 3

A static cursor is declared with an input variable.

```
DECLARE cur_order CURSOR FOR
   SELECT order_num, order_date, order_text, order_stat
   FROM orders
   WHERE cust_num >= &CUST_NUM;
```

The cursor description with the current value of &CUST_NUM is evaluated for OPEN *cur_order*. When RESTORE *cur_order* is executed, the cursor description of the last OPEN *cur_order* statement remains valid.

See also

CLOSE, DELETE, FETCH, INSERT, OPEN, SELECT, UPDATE

DROP INDEX - Delete index

You use DROP INDEX to delete an index. The index may have been created explicitly with a CREATE INDEX statement or implicitly by the definition of an integrity constraint (UNIQUE, PRIMARY KEY).

The INDEXES view of the INFORMATION_SCHEMA provides you with information on which indexes have been defined (chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

If an explicitly defined index is also used by an integrity constraint, the index is not deleted but is renamed as an implicit index. The new index name starts with UI and is followed by a 16-digit number.

Indexes created implicitly by an integrity constraint (UNIQUE, PRIMARY KEY) are not deleted until the relevant integrity constraint is deleted.

The current authorization identifier must own the schema to which the index belongs.

```
DROP INDEX index
```

---

index
Name of the index to be deleted.

You can qualify the name of the index with a database and schema name.

See also

CREATE INDEX

DROP SCHEMA - Delete schema

You use DROP SCHEMA to delete a database schema.

The SCHEMATA view of the INFORMATION_SCHEMA provides you with information on which schemas have been defined (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must own the schema.

```
DROP SCHEMA schema { CASCADE | RESTRICT }
```

---

schema
Name of the schema. The schema must be empty.

You can qualify the name of the schema with a database name.

CASCADE
The schema *schema* and all the objects of the schema are deleted. Views and integrity constraints that reference the base tables or views in *schema* are also deleted.

RESTRICT
The schema *schema* can only be deleted when it is empty. All the schema’s base tables, views and integrity constraints must be deleted beforehand.

See also

CREATE SCHEMA, DROP TABLE, DROP VIEW

DROP SPACE - Delete space

You use DROP SPACE to delete a user space. All the base tables and indexes in the space must be deleted beforehand. The space file is overwritten with binary zeros if the DESTROY parameter was specified when the space was created or updated.

The SPACES view of the INFORMATION_SCHEMA provides you with information on which user spaces have been defined (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must own the space.

```
DROP SPACE space RESTRICT
```

---

space
Name of the space. The space must be empty.

You can qualify the name of the space with a database name.

DROP SPACE and transactions

A DROP SPACE statement cannot be followed by a CREATE SPACE statement within the same transaction.

See also

CREATE SPACE, ALTER SPACE

DROP STOGROUP - Delete storage group

You use DROP STOGROUP to delete a storage group. You cannot delete a storage group if it is being used for spaces or has been entered in the media table (see the “SESAM/SQL-Server - Core Manual” [8]).

The STOGROUPS view of the INFORMATION_SCHEMA provides you with information on which storage groups have been defined (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must own the storage group.

```
DROP STOGROUP stogroup RESTRICT
```

---

stogroup
Name of the storage group. The storage group cannot be deleted if it is being used.

You can qualify the name of the storage group with a database name.

See also

CREATE STOGROUP, ALTER STOGROUP

DROP SYSTEM_USER - Delete system entry

You use DROP SYSTEM_USER to delete a system entry, i.e. the assignment of an authorization identifier to a system user. You must specify the combination of system user and authorization identifier that was defined for a system entry with CREATE SYSTEM_USER.

You cannot delete a system entry if it is the last assignment of a system user to the authorization identifier of the universal user.

If an SQL transaction belonging to the system user is currently active, his or her system entry is only deleted if another system entry exists for the system user.

The SYSTEM_ENTRIES view of the INFORMATION_SCHEMA provides you with information on which authorization identifiers have been assigned to which system users (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must have the special privilege CREATE USER. If the assignment of an authorization identifier with the special privilege CREATE USER and GRANT authorization (see “GRANT - Grant privileges” on page 136) to a system user is to be deleted, the current authorization identifier must also have GRANT authorization.

```
DROP SYSTEM_USER { utm_user | bs2000_user }
            FOR authorization_identifier
            AT CATALOG catalog

utm_user::=( { hostname|* },{ utm_application_name|* },{ utm_userid|* } )
bs2000_user::=( { hostname|* },[*],{ bs2000_userid|* } )
```

---

utm_user

Delete a system entry of a UTM system user.

You must specify the UTM user exactly as it was defined with CREATE SYSTEM_USER. * means the system entry defined with *, not all relevant system entries.

hostname

Alphanumeric literal indicating the symbolic host name.

If DCAM is not available on the host, the host is assigned the name “HOMEPROC”.

* All hosts.

utm_application_name
Alphanumeric literal indicating the name of the UTM application.

* All UTM applications.

utm_userid
You specify the UTM user ID as an alphanumeric literal defined with KDCSIGN for local UTM system users. For UTM-D, you specify the local UTM session name (LSES).

* All UTM user IDs.

bs2000_user
Delete a system entry of a BS2000 system user.
You must specify the BS2000 user exactly as it was defined with CREATE SYSTEM_USER. * means the system entry defined with *, not all relevant system entries.

hostname
Alphanumeric literal indicating the symbolic host name. If DCAM is not available on the host, the host is assigned the name “HOMEPROC”.

* All hosts.

bs2000_userid
Alphanumeric literal indicating the BS2000 user ID.

* All BS2000 user IDs.

FOR *authorization_identifier*
Name of the authorization identifier assigned to the system user.

AT CATALOG *catalog*
Name of the database for which the assignment of the system user to the authorization identifier is to be deleted.

Example

In the example below, two system entries are deleted. The system entries must be specified exactly as they were defined with CREATE SYSTEM_USER. The authorization identifiers `hugh` and `bertha` are not deleted.

```
DROP SYSTEM_USER (*,,'purchasing') FOR hugh AT CATALOG my_db
DROP SYSTEM_USER (*,,'sales') FOR bertha AT CATALOG my_db
```

See also

CREATE SYSTEM_USER, CREATE USER, DROP USER

DROP TABLE - Delete base table

You use DROP TABLE to delete a base table and the associated indexes.

When a base table is deleted, all the table and column privileges for this base table are revoked from the current authorization identifier. Table and column privileges that have been passed on are also revoked.

The BASE_TABLES view of the INFORMATION_SCHEMA provides you with information on which base tables have been defined (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must own the schema to which the table belongs.

This statement can destroy declaration statements in a DRIVE program.

```
DROP TABLE table { CASCADE | RESTRICT }
```

---

table
Name of the base table to be deleted.

CASCADE
The base table *table* and all the associated indexes are deleted. All the views and integrity constraints that reference *table* are deleted.

RESTRICT
The base table *table* cannot be deleted if it is used in a view definition or an integrity constraint of another base table.

See also

CREATE TABLE, ALTER TABLE

DROP USER - Delete authorization identifier

You use DROP USER to delete an authorization identifier and the associated system entries. You cannot delete an authorization identifier if it is the owner of schemas, spaces or storage groups, if it is the grantor of a privilege, or if an SQL transaction is currently active for the authorization identifier.

All the temporary views of the specified authorization identifier in the specified database are deleted if they are still present[1].

You cannot delete the authorization identifier of the universal user.

The USERS view of the INFORMATION_SCHEMA provides you with information on which authorization identifiers have been defined. Information on which authorization identifier is owner is stored in the views SCHEMATA, SPACES and STOGROUPS. The TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES and CATALOG_PRIVILEGES views provide you with information on whether the authorization identifier is grantor of a privilege (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must have the special privilege CREATE USER. If you want to delete an authorization identifier that as been granted the special privilege CREATE USER and GRANT authorization (see “GRANT - Grant privileges” on page 136), the current authorization identifier must also have GRANT authorization.

```
DROP USER authorization_identifier AT CATALOG catalog RESTRICT
```

---

authorization_identifier
Name of the authorization identifier to be deleted.

AT CATALOG *catalog*
Name of the database from which the authorization identifier is to be deleted.

See also

CREATE USER, CREATE SYSTEM_USER, DROP SYSTEM_USER

[1] Temporary views are no longer supported as of SESAM/SQL V3.0.

DROP VIEW - Delete view

You use DROP VIEW to delete the definition of a view.

When a view definition is deleted, all the table and column privileges for this view are revoked from the current authorization identifier. Table and column privileges of the view that have been passed on are also revoked.

The VIEWS view of the INFORMATION_SCHEMA provides you with information on which views have been defined. Information on the tables a view uses is provided in the view VIEW_TABLE_USAGE (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The current authorization identifier must own the schema to which the view belongs.

```
DROP VIEW table { CASCADE | RESTRICT }
```

---

table
Name of the view to be deleted.

CASCADE
The view *table* and all the views in whose definition *table* is used are deleted.

RESTRICT
The view *table* cannot be deleted if it is used in another view definition.

See also

CREATE VIEW

GRANT - Grant privileges

You use GRANT to grant table and column privileges for base tables and views, and special privileges for databases and storage groups. If the GRANT statement is included in a CREATE SCHEMA statement, you cannot grant special privileges with GRANT.

The current authorization identifier must be authorized to grant the specified privileges:

– It is the authorization identifier of the universal user.
– It is the owner of the table, database or storage group.
– It has GRANT authorization for granting the privileges to other users.

Information on which authorization identifiers are owners is stored in the SCHEMATA, SPACES and STOGROUPS views. The TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES and CATALOG_PRIVILEGES provide you with information on whether the authorization identifier has GRANT authorization for a certain privilege (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

Only the authorization identifier that granted a privilege (the “grantor”) can revoke that privilege.

The GRANT statement has two formats: one for granting table and column privileges and one for granting special privileges.

GRANT format for table and column privileges

```
GRANT     { ALL PRIVILEGES |
              { SELECT |
                DELETE |
                INSERT |
                UPDATE [(column,...)] |
                REFERENCES [(column,...)]
              }, ...
          }

          ON [ TABLE ] table

          TO { PUBLIC | authorization_identifier }, ...

          [ WITH GRANT OPTION ]
```

ALL PRIVILEGES
All the table and column privileges that the current authorization identifier can grant are granted. ALL PRIVILEGES comprises the privileges SELECT, DELETE, INSERT, UPDATE and REFERENCES.

SELECT | DELETE ...
The table and column privileges are granted individually. You can specify more than one privilege. The following specifications are possible:

SELECT
Privilege that allows rows in the table to be read.

DELETE
Privilege that allows rows to be deleted from the table.

INSERT
Privilege that allows rows to be inserted into the table.

UPDATE [(*column*,...)]
Privilege that allows rows in the table to be updated.

The update operation can be limited to the specified columns. *column* must be the name of a column in the specified table. You can specify more than one column.

(*column*,...) omitted:
All the columns in the table can be updated including columns inserted later.

REFERENCES [(*column*,...)]
Privilege that allows the definition of referential constraints that reference the table. The reference can be limited to the specified columns. *column* must be the name of a column in the specified table. You can specify more than one column.

(*column*,...) omitted:
All the column in the table can be referenced including columns inserted later.

ON [TABLE] *table*
Name of the table for which you want to grant privileges.

If you use the GRANT statement in a CREATE SCHEMA statement, you can only qualify the table name with the database and schema name from the CREATE SCHEMA statement.

The table can be a base table or a view. You can only grant the SELECT privilege for a table that cannot be updated.

TO PUBLIC
The privileges are extended to all authorization identifiers, both current and future. Each authorization identifier is granted the privileges extended to PUBLIC in addition to its own privileges.

TO *authorization_identifier*
The privileges are granted to *authorization_identifier*. You may specify more than one authorization identifier.

WITH GRANT OPTION
The specified authorization identifier(s) is granted not only the specified privileges but also GRANT authorization. This means that the authorization identifier(s) is authorized to grant the privileges it has been extended to other authorization identifiers. You cannot specify WITH GRANT OPTION together with PUBLIC.

WITH GRANT OPTION omitted:
The specified authorization identifier(s) cannot grant the privileges it has been extended to other authorization identifiers.

GRANT format for special privileges

```
GRANT     { ALL SPECIAL PRIVILEGES |
              { CREATE USER |
                CREATE SCHEMA |
                CREATE STOGROUP |
                UTILITY |
                USAGE
              }, ...
          }

          ON { CATALOG catalog | STOGROUP stogroup }

          TO { PUBLIC | authorization_identifier }, ...

          [ WITH GRANT OPTION ]
```

ALL SPECIAL PRIVILEGES
All the special privileges that the current authorization identifier can grant are granted. ALL SPECIAL PRIVILEGES comprises the special privileges CREATE USER, CREATE SCHEMA, CREATE STOGROUP, UTILITY and USAGE.

CREATE USER | CREATE SCHEMA ...
The special privileges are granted individually. You can specify more than one privilege. The following special privileges can be specified:

CREATE USER
Special privilege that allows you to define and delete authorization identifiers. You can only grant the CREATE USER privilege for a database.

CREATE SCHEMA
Special privilege that allows you to define database schemas.
You can only grant the CREATE SCHEMA privilege for a database.

CREATE STOGROUP
Special privilege that allows you to define storage groups.
You can only grant the CREATE STOGROUP privilege for a database.

UTILITY
Special privilege that allows you to use utility statements
You can only grant the UTILITY privilege for a database.

USAGE
Special privilege that allows you to use a storage group.
You can only grant the USAGE privilege for a storage group.

ON CATALOG *catalog*
Name of the database for which you are granting special privileges.

ON STOGROUP *stogroup*
Name of the storage group for which you want to grant the USAGE privilege. You can qualify the name of the storage group with a database name.

TO PUBLIC
The privileges are extended to all authorization identifiers, both current and future. Each authorization identifier is granted the privileges extended to PUBLIC in addition to its own privileges.

TO *authorization_identifier*
The privileges are granted to *authorization_identifier*. You may specify more than one authorization identifier.

WITH GRANT OPTION
The specified authorization identifier(s) is granted not only the specified privileges but also GRANT authorization. This means that the authorization identifier(s) is authorized to grant the privileges it has been extended to other authorization identifiers. You cannot specify WITH GRANT OPTION together with PUBLIC.

WITH GRANT OPTION omitted:
The specified authorization identifier(s) cannot grant the privileges it has been extended to other authorization identifiers.

Example

In the example below, the first GRANT statement grants several table privileges, the second grants the special privilege CREATE SCHEMA to an existing authorization identifier.

```
GRANT SELECT,INSERT,UPDATE ON TABLE telephone_list TO bertha

GRANT CREATE SCHEMA ON CATALOG my_db TO hugh
```

See also

REVOKE, CREATE SCHEMA

PRAGMA - Declare pragma clauses

While in ESQL/COBOL and in the utility monitor, pragmas are entered as special SQL comments together with the SQL statement that they are to influence, in DRIVE they are declared using a separate DRIVE SQL statement, the PRAGMA statement.

DRIVE converts the declared clauses into pragmas for SESAM/SQL, i.e. special SQL comments that can be used to influence or monitor the execution of SQL statements.

Application possibilities and advantages

The SESAM V2 user can use pragmas to

– activate block mode, i.e. specify the maximum number of rows in a (static, dynamic or variable) cursor table that can be read “in advance” by a FETCH statement in order to accelerate execution of subsequent FETCH statements considerably.

This functionality is only available if you are using SESAM/SQL V2.1 (or higher). SESAM/SQL V2.0 does not support block mode, which means that use of the block mode pragma clause PREFETCH in DRIVE will result in an error (see “Error handling when using pragmas” on page 150).

– output a readable representation of the internal access plan of the SQL optimizer to a file for new-style DML statements (SELECT or cursor processing, INSERT, UPDATE, DELETE)

– influence the execution rule (SQL access plan) for processing a new-style DML statement.

– select the join method used (sort, merge join or nested-loop join).

– set lock mode for SQL DML statements.

– specify the isolation level for database accesses by a new-style DML statement independent of the isolation level of the transaction in which the statement is executed.

– insert new oldest-style columns into an oldest-style table (CALL DML table) (see the utility statement MIGRATE and the DDL statement ALTER TABLE).

– process base tables in the state “check pending” (see the “SESAM/SQL-Server - SQL Reference Manual, Part 2” [7]).

Characteristics of the PRAGMA statement

A PRAGMA statement is a program statement that is evaluated at compilation time. Pragmas can be defined both statically and dynamically:

– A static PRAGMA statement can appear anywhere between PROCEDURE and END PROCEDURE, and only affects the (textually) next static SQL statement. It is evaluated at the time of explicit source compilation (COMPILE statement) or at the time of implicit source compilation (DO or CALL statement: preliminary step of procedure execution).

– A dynamic PRAGMA statement can appear anywhere that EXECUTE is permitted. It only affects the (chronologically) next dynamic SQL statement and is evaluated during the implicit statement compilation (EXECUTE statement: generation and compilation as preliminary step of statement execution).

A PRAGMA statement cannot be executed and does not therefore initiate any transactions.

In DRIVE, this causes the SQL statement is prefixed with the special comment

"--%PRAGMA { pragma_clause },..."

The effect of the PRAGMA statement in DRIVE therefore corresponds to a conversion of the pragma clauses into SESAM-compliant usage as in ESQL/COBOL programs and in the utility monitor.

Whether the effect in DRIVE leads to an effect in SESAM depends on whether the specific pragma clauses influence the SQL statement:

– PREFETCH only has an effect in SESAM for DECLARE statements and indirectly for the corresponding FETCH statements.

– EXPLAIN INTO, IGNORE INDEX, OPTIMIZATION LEVEL, SIMPLIFICATION and ISOLATION LEVEL only influence the statements DECLARE, SELECT, INSERT, UPDATE and DELETE in SESAM.

– JOIN only has an effect in SESAM for SELECT statements.

– LOCKMODE only has an effect in SESAM for DML statements.

– UTILITY MODE only has an effect in SESAM for ALTER TABLE statements.

If a *literal* in a pragma clause does not have any effect in SESAM, an error with &WARNING = SQL WARNING' and &SQL_CLASS = '01' occurs in DRIVE when the corresponding SQL statement is compiled (static PRAGMA) or executed (dynamic PRAGMA), see also page 150.

PRAGMA clauses

```
PRAGMA '{ CHECK { ON | OFF } |
          DATA TYPE OLDEST |
          EXPLAIN INTO file_name|
          IGNORE INDEX index |
          ISOLATION LEVEL { READ UNCOMMITTED |
                            READ COMMITTED |
                            REPEATABLE READ |
                            SERIALIZABLE
                          } |
          JOIN [ { SORT MERGE | NESTED LOOP } ] |
          LOCK MODE EXCLUSIVE |
          OPTIMIZATION LEVEL level |
          PREFETCH n |
          SIMPLIFICATION { ON | OFF } |
          UTILITY MODE { ON | OFF } |
       }, ...'
```

CHECK pragma clause

The CHECK pragma clause has the following syntax:

```
CHECK { ON | OFF }
```

The default value is ON. Under certain conditions, you can use the clause 'CHECK OFF' to access base tables that are in the state “check pending” with DML statements (see the “SESAM/SQL-Server - SQL Reference Manual, Part 2” [7]).

DATA TYPE pragma clause

DATA TYPE defines that a column in attribute format is created for exclusive CALL DML tables.

The clause only affects SESAM if it is specified with the statement ALTER TABLE ... ADD COLUMN ... and the table is an exclusive CALL DML table.

```
DATA TYPE OLDEST
```

EXPLAIN pragma clause

EXPLAIN is used to output the access plan selected by the optimizer. You can only use this pragma if the current authorization identifier has the special privilege UTILITY (see the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

This pragma is only effective in the following SQL statements in SESAM:
– DECLARE
– DELETE
– INSERT
– SELECT
– UPDATE

This pragma is only effective in a static statement if you precompile the program while the database is online. This is always the case in DRIVE.

```
EXPLAIN INTO file
```

---

file

Name of the SAM file into which the explanation is to be output. If the file already exists, the explanation is appended to the file.

If *file* includes a BS2000 user ID, this user ID is used. If not, the ID of the DBH (Data Base Handler) for the database referenced in the SQL statement is used. In both cases the DBH must have write permission for the file.

You specify an alphanumeric literal for *file*.

In the case of dynamic statements, the explanation is output when the EXECUTE statement is executed. For static statements, the explanation is output during precompilation.

The explanation comprises the SQL statement and an edited representation of the access plan. The representation of access plans is described in the manual “SESAM/SQL-Server - Performance” [12].

You can display the contents of the file with SHOW-FILE. If you want to read the file with EDT, you must enter the following command:

```
SET-FILE-LINK LINK-NAME=EDTSAM,FILE-NAME=file,...,BUFFER-LENGTH=(STD,2),...
```

As of EDT Version 16.5A, you can also enter:

```
@OPEN F=file,TYPE=CATALOG
```
or
```
@OP F=file,T=C
```

Example

```
SET &explainfile = '$YDRI20.EXPLAINFILE';

/* The file $YDRI20.EXPLAINFILE must be cataloged with */
/* USER-ACC = ALL-USERS */

EXECUTE 'SET SESSION AUTHORIZATION ''"DRI-USER1" ''';

/* The authorization identifier DRI-USER1 must have the special */
/* privilege UTILITY for the default database */

EXECUTE 'PRAGMA ''EXPLAIN INTO ''''|| &explainfile || ''''''';

DISPLAY FORM 'Output of the SQL access plan in file ' || &explainfile;

EXECUTE 'DECLARE C1 CURSOR FOR cursor_description';
```

IGNORE INDEX pragma clause

The specified index is ignored when the join order and join algorithm are specified and when the optimum access path (to base relations) is selected.

```
IGNORE INDEX index
```

ISOLATION LEVEL pragma clause

ISOLATION LEVEL determines the isolation level for database accesses performed by an SQL statement.

This pragma is only effective in the following SQL statements in SESAM:
– DECLARE
– DELETE
– INSERT
– SELECT
– UPDATE

```
ISOLATION LEVEL
              { READ UNCOMMITTED |
                 READ COMMITTED |
                 REPEATABLE READ |
                 SERIALIZABLE }
```

The isolation levels are described under the SET TRANSACTION statement.

If you have specified the ISOLATION LEVEL pragma, any database access performed in connection with this statement takes place under this isolation level.

If you specify a lower isolation level than specified for the transaction, the isolation level defined for the transaction is no longer guaranteed.

JOIN pragma clause

When this pragma is entered, the appropriate join method (sort, merge, join or nested loop) is selected.

```
JOIN [SORT MERGE | NESTED LOOP]
```

If the OPTIMIZATION LEVEL pragma restricts the number of possible plan alternatives such that nested-loop joins are not taken into account (OPTIMIZATION LEVEL < 6), the JOIN pragma can likewise not force a nested-loop join.

In the case of multiple joins, the pragma is also taken into account in conjunction with any intermediate joins. If the JOIN pragma is not specified, the optimizer selects the join method that is to be used.

LOCK MODE pragma clause

The LOCK MODE pragma sets the lock mode. It is only effective in SQL-DML statements.

```
LOCK MODE EXCLUSIVE
```

If LOCK MODE EXCLUSIVE is specified, every access to the database connected directly or indirectly with this SQL statement involves exclusive locks. Otherwise the lock mode is defined by the system.

OPTIMIZATION LEVEL pragma clause

The option n controls the number of plan alternatives that can be generated and evaluated during access path selection.

```
OPTIMIZATION LEVEL n
```

Firstly, all plan variants are examined, and only the most favorable variants that in general promise an improvement of the evaluation costs are selected by means of heuristic techniques. The number of plan variants that are followed up on depends on the value of n. The value of n can be between 1 and 10; the default value is 9. If a value $n \leq 5$ is specified, only one plan variant is examined in each optimization step.

A distinction is made between the following levels:

– $n \leq 9$
  Various join orders are considered.

– $n \leq 8$
  In the case of a nested-loop join, this statement checks whether switching the two join partners would be advantageous.

– $n \leq 7$
  Execution of sort minimization.

– $n \leq 6$
  In the case of join optimization, not only the sort merge but also the nested-loop join is considered.

  When selecting the access path, all the possibilities for achieving the required sort are considered (physical sorting in the DBH kernel, sorting via index scan).

– $n \leq 5$
  Execution of subquery optimization and storage of intermediate result relations that are needed more than once.

– $n \leq 4$
  Execution of the range construction, i.e. several atomic predicates on the same column are grouped together as one index access.

PREFETCH pragma clause

PREFETCH controls the block mode of the SQL statement FETCH (position cursor). Block mode accelerates execution of the FETCH statement. It is only effective if FETCH is used to position the cursor on the next row in the cursor table (FETCH NEXT).

You can use the PREFETCH pragma to activate block mode and specify a blocking factor (n). When the first FETCH NEXT... statement is executed, the column values of the current row are read and the next n-1 rows of the associated cursor table are stored in a buffer. When the next n-1 FETCH NEXT... statements that relate to the same cursor are executed, the next row can be accessed directly, without DBH contact.

If the cursor description of a DECLARE statement for static or dynamic cursors includes a FOR UPDATE clause, the PREFETCH pragma is ignored (i.e. it does not have any effect in SESAM), and block mode is not activated.

PREFETCH *n*

---

n Blocking factor. You must specify the blocking factor as an unsigned integer (of the type SMALLINT).

If the blocking factor (*n*) has a value > 0, up to n-1 rows in the specified cursor table are stored in a buffer. If the blocking factor is the value 0, the PREFETCH pragma has no effect. This means that you can activate the effect of the pragma and thus block mode by specifying a value > 0 for *n* and deactivate it by specifying the value 0.

The following restrictions apply if block mode has been activated:

Only the FETCH NEXT statement is permitted for the PREFETCH cursor in the same compilation unit. The following SQL statements are no longer executable:

– UPDATE ... WHERE CURRENT OF *cursor*
– DELETE ... WHERE CURRENT OF *cursor*
– STORE *cursor*
– FETCH *cursor* with a cursor position that is different to NEXT
– FETCH *cursor* with an INTO clause that is different to the first FETCH NEXT statement if *cursor* is static.

UTILITY MODE pragma clause

The UTILITY MODE pragma determines whether transaction logging is effective in the SQL statement in which this pragma is specified. Transaction logging makes it possible to roll a transaction back to a consistent state

The UTILITY MODE pragma is only effective in the SQL statement ALTER TABLE.

It only works if the ALTER TABLE statement adds, changes or deletes columns in a base table. In an ALTER TABLE statement which adds or deletes integrity constraints, the UTILITY MODE pragma has no effect.

```
UTILITY MODE { ON | OFF }
```

ON
: Transaction logging is deactivated during the execution of the SQL statement. The associated - ALTER TABLE statement does not open a transaction.
No save data for the ALTER TABLE statement is stored. If an error occurs which results in an interruption of the statement, the transaction cannot be rolled back to a consistent state. When an error occurs, the space containing the base table is damaged and must be repaired using the RECOVER utility statement (see the “SESAM/SQL-Server - SQL Reference Manual, Part 2” [7]).

OFF
: The pragma has no effect and transaction logging remains active.

An ALTER TABLE statement, for which the UTILITY MODE pragma is switched ON and is effective, is aborted with an error message in the following cases:

– when a transaction is active
– if the ALTER TABLE table deletes a column, i.e. using DROP COLUMN *column* CASCADE.

If no UTILITY MODE pragma is specified for an ALTER TABLE statement then the default setting, UTILITY MODE OFF, is effective.

If you use the UTILITY MODE ON pragma then, after an error or consistency check, the space containing the base table to be changed is defective. To avoid data loss, you should save the space before issuing the ALTER TABLE statement. The save is necessary if you want to use the utility statement RECOVER to repair it.

Error handling when using pragmas

When using pragmas, the following errors may occur:

– The syntax of PRAGMA statement is incorrect.

  In this case, DRIVE reports a syntax error during compilation of the PRAGMA statement. In the case of a dynamic PRAGMA statement, &ERROR is then assigned the value 'SYNTAX ERROR'.

Otherwise, the PRAGMA statement takes effect (so-called DRIVE effect) when the next SQL statement is executed and the contents of the specified literal are passed to SESAM as an SQL comment with the prefix %PRAGMA. Errors may occur that SESAM detects and which mean that the statement has no affect in SESAM:

– The syntax of the contents of the literal in the PRAGMA statement is incorrect.

  In this case, SESAM reports an SQLSTATE of the class 01 (warning) to DRIVE and precompiles (static pragma) or prepares (dynamic pragma) the SQL statement without the pragma.

– The syntax of the contents of the literal in the PRAGMA statement is correct, but the pragma clause is assigned to an SQL statement that it cannot influence (see description of the individual clauses).

  This is the case, for example, if the PREFETCH pragma clause is used with SESAM/SQL V2.0. SESAM reports an SQLSTATE of the class 01 (warning) to DRIVE and precompiles (static pragma) or prepares (dynamic pragma) the SQL statement without the pragma.

– The syntax of the contents of the literal in the PRAGMA statement is correct and the pragma clause is assigned to an SQL statement that it can influence in SESAM, but this influence is hindered for some other reason (see description of the individual clauses).

  This is the case, for example, if a BS2000 file that cannot be shared or which has not been cataloged as SHAREABLE is specified in the EXPLAIN pragma clause.

  In this case, SESAM also reports an SQLSTATE of the class 01 (warning) to DRIVE and precompiles or prepares the SQL statement without the pragma.

DRIVE converts all warnings from SESAM (SQLSTATE class = 01) into the following DRIVE warning:

```
&WARNING='SQL WARNING'
```

If a WHENEVER action is defined for this exception condition (see page 160), then this determines the reaction to the SQL warning, e.g. continue, reset or close transaction.

If no WHENEVER action is defined for this exception condition, the warning is ignored because the error exit ‘CONTINUE’ is set by DRIVE for warnings. This means that the SQL statement is executed without pragma effect.

REORG STATISTICS

You use REORG STATISTICS to re-generate global statistics on the distribution of values over the column in an index. These statistics are used to optimize table accesses with search conditions and should be updated whenever extensive changes are made to the data.

The current authorization identifier must either be the owner of the schema to which the index belongs or must have the special privilege UTILITY for the database to which the index belongs.

```
REORG STATISTICS FOR INDEX index
```

---

index
Name of the index for which the statistics are to be re-generated.

You can qualify the name of the index with a database and schema name.

See also

CREATE INDEX

REVOKE - Revoke privileges

You use REVOKE to revoke table and column privileges or special privileges from authorization identifiers. If temporary views[1] of the authorization identifier are still based on the table, they are deleted.

Only the authorization identifier that granted a privilege (the “grantor”) can revoke that privilege from an authorization identifier (see “GRANT - Grant privileges” on page 136).

The TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGE and CATALOG_PRIVILEGES views of the INFORMATION_SCHEMA provide you with information on the privileges assigned to the authorization identifiers (see the chapter “Information schemas” in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6]).

The revoke statement has two formats: one format for table and column privileges and another for special privileges.

REVOKE format for table and column privileges

```
REVOKE { ALL PRIVILEGES |
           { SELECT |
             DELETE |
             INSERT |
             UPDATE [(column,...)] |
             REFERENCES [(column,...)]
           }, ...
       }

       ON [ TABLE ] table

       FROM { PUBLIC | authorization_identifier }, ...

       { CASCADE | RESTRICT }
```

ALL PRIVILEGES
: All the table privileges that the current authorization identifier can revoke are revoked. ALL PRIVILEGES comprises the privileges SELECT, DELETE, INSERT, UPDATE and REFERENCES.

SELECT | DELETE | INSERT | UPDATE [(*column*,...)] REFERENCES [(*column*,...)]
: The table and column privileges are revoked individually. You can specify more than one of the following privileges:

[1] Temporary views are no longer supported as of SESAM/SQL V3.0.

SELECT
Privilege that allows rows in the table to be read.

DELETE
Privilege that allows rows to be deleted from the table.

INSERT
Privilege that allows rows to be inserted into the table.

UPDATE [(*column*,...)]
Privilege that allows rows in the table to be updated.

The revoke operation can be limited to the specified columns. *column* must be the name of a column in the specified table. You can specify more than one column.

(*column*,...) omitted:
The privilege for updating all the columns in the table is revoked.

REFERENCES [(*column*,...)]
Privilege that allows the definition of referential constraints that reference the table.

The revoke operation can be limited to the specified columns. *column* must be the name of a column in the specified table. You can specify more than one column.

(*column*,...) omitted:
The privilege for referencing all the columns in the table is revoked.

ON [TABLE] *table*
Name of the table for which you want to revoke privileges.

The table can be a base table or a view. You can only revoke the SELECT privilege for a table that cannot be updated.

FROM PUBLIC
The privileges are revoked from all authorization identifiers. The individual privileges of the individual authorization identifiers are not affected.

FROM *authorization_identifier*
The privileges are revoked from the user with the authorization identifier *authorization_identifier*. You may specify more than one authorization identifier.

CASCADE
An authorization identifier can revoke any privileges it has granted:

– All the specified privileges are revoked.

– If a specified privilege has been forwarded to other authorization identifiers, all forwarded privileges are deleted implicitly.

– Views defined on the basis of the specified privilege are deleted.

– Referential constraints defined on the basis of the specified privilege are deleted.

RESTRICT
The following restrictions apply to the revoking of privileges:

– A privilege forwarded to other authorization identifiers cannot be revoked for as long as a forwarded privilege like this still exists.

– A privilege on the basis of which a view or referential constraint has been defined cannot be revoked if the view or referential constraint still exists.

REVOKE format for special privileges

```
REVOKE { ALL SPECIAL PRIVILEGES |
          { CREATE USER |
            CREATE SCHEMA |
            CREATE STOGROUP |
            UTILITY |
            USAGE
          }, ...
       }

       ON { CATALOG catalog | STOGROUP stogroup }

       FROM {PUBLIC | authorization_identifier }, ...

       { CASCADE | RESTRICT }
```

ALL SPECIAL PRIVILEGES
All the special privileges that the current authorization identifier can revoke are revoked. ALL SPECIAL PRIVILEGES comprises the special privileges CREATE USER, CREATE SCHEMA, CREATE STOGROUP, UTILITY and USAGE.

CREATE USER | CREATE SCHEMA | CREATE STOGROUP | UTILITY | USAGE
The special privileges are revoked individually. You can specify more than one of the following special privileges:

CREATE USER
Special privilege that allows you to define authorization identifiers.
You can only revoke the CREATE USER privilege for a database.

CREATE SCHEMA
Special privilege that allows you to define database schemas.
You can only revoke the CREATE SCHEMA privilege for a database.

CREATE STOGROUP
: Special privilege that allows you to define storage groups.
You can only revoke the CREATE STOGROUP privilege for a database.

UTILITY
: Special privilege that allows you to use utility statements.
You can only revoke the UTILITY privilege for a database.

USAGE
: Special privilege that allows you to use a storage group.
You can only revoke the USAGE privilege for a storage group.

ON CATALOG *catalog*
: Name of the database for which you want to revoke special privileges.

ON STOGROUP *stogroup*
: Name of the storage group for which you want to revoke the USAGE privilege. You can qualify the name of the storage group with a database name.

FROM *authorization_identifier*
: The privileges are revoked from the user with the authorization identifier *authorization_identifier*. You may specify more than one authorization identifier.

CASCADE
: An authorization identifier can revoke any privileges it has granted:

  – All the specified privileges are revoked.

  – If a specified privilege has been forwarded to other authorization identifiers, all forwarded privileges are deleted implicitly.

  – Views defined on the basis of the specified privilege are deleted.

  – Referential constraints defined on the basis of the specified privilege are deleted.

RESTRICT
: The following restrictions apply to the revoking of privileges:

  – A privilege forwarded to other authorization identifiers cannot be revoked for as long as a forwarded privilege like this still exists.

  – A privilege on the basis of which a view or referential constraint has been defined cannot be revoked if the view or referential constraint still exists.

Example

The REVOKE statement in the example below revokes the UPDATE privilege for all the columns in the table `telephone_list`.

```
REVOKE UPDATE ON TABLE telephone_list FROM bertha
```

See also

GRANT

UTILITY - Forward UTILITY statement

This statement is used to forward UTILITY statements to SESAM/SQL.

`UTILITY` *utility-statement*

The UTILITY statements are forwarded as DRIVE strings. However, the string is not fully checked in this case.

The following UTILITY statements are possible:

Statement	**Meaning**
ALTER MEDIA DESCRIPTION	Modify media table
CHECK CONSTRAINTS	Check integrity constraints
CHECK FORMAL	Check format of tables and indexes
COPY	Create backup copies
CREATE CATALOG	Create database (catalog space)
CREATE MEDIA DESCRIPTION	Define file attributes of database-specific files
CREATE REPLICATION	Create replica from backup copy
DROP MEDIA DESCRIPTION	Delete all media table entries for database-specific file type
LOAD	Load user data into base table
MIGRATE	Convert databases and tables
MODIFY	Maintain information (metadata) on the backup data
RECOVER	Full and partial recovery, rebuild indexes
REFRESH REPLICATION	Update replication
REORG	Reorganize catalog space and user spaces
UNLOAD	Unload user data from base table

Further information on these statements can be found in the “SESAM/SQL-Server - SQL Reference Manual, Part 2” [7].

5 DRIVE statements

This chapter contains the changes and supplements for “DRIVE Directory” [3]. The specified section and chapter numbers refer to this manual.

Overview

- WHENEVER statement:

  New &WARNING operand for handling warnings in DRIVE.

- PAGE PRINT statement:

  The keyword POSITION is mandatory.

- DRIVE metavariable charprim.

- New string function SQLMSGSTRING for determining the message text of the SQL interface.

- DRIVE statements and parameters for FHS-DE:

  Since FHS-DE is not supported, the statements ADD BOX, REMOVE BOX and REPLACE BOX are omitted, as are the FHS-DE-specific operands of the statement DISPLAY screenformat.

5.1 WHENEVER - Define error exit

(See chapter 3, description of the WHENEVER statement on page 206)

This application is valid

– in TIAM and UTM mode
– in program mode

WHENEVER is used to define an error exit in case a semantic error occurs in a program. WHENEVER must be defined in the declaration section of the program after the definitions of internal subprograms. If more than one WHENEVER is included for a given event, the most recent specification is used.

The WHENEVER statement polls the entries in the system variables &KFKEY, &ERROR (= &ERROR_STATE.ERROR), &WARNING (&ERROR_STATE.WARNING) and &DML_STATE (= &ERROR_STATE.DML_STATE) and defines error exits. For a description of the system variables and their entries, refer to the “DRIVE Programming Language” manual [2], section “System variables“.

If entries for &ERROR and for &DML_STATE are queried, and if both events occur simultaneously, the error exit defined for &ERROR is executed if &SQL_CODE > 0, otherwise the error exit for &DML_STATE is executed.

If an error occurs, the corresponding counter is incremented.

If no error exit is defined, DRIVE aborts the program. (Exception: the program is continued with the &ERROR entries "OK END", "TOO LONG" and "TOO SHORT", with the &DML_STATE entries "TABLE END", "DIRTY READ" and "SQL CONV WARNING", and with all &WARNING entries.)

```
WHENEVER { &KFKEY [ IN ( literal, ... ) ] |
           &ERROR [ IN ( error, ... ) ] |
           &DML_STATE [ IN ( status, ... ) ]
           &WARNING [ IN ( warning, ...) ]
         }

         { CONTINUE | CALL subprog-name | BREAK }
```

&KFKEY IN

The condition takes effect when the *literal* key is pressed. This is only evaluated in programs without a graphic user interface. DRIVE then sets the CONTINUE operation.

The condition is only executed if no other error condition occurs.

If an overflow occurs on the execution of a DISPLAY statement then pressing the *literal* key aborts output. CONTINUE, CALL or BREAK is executed as the next statement.

literal

Key designation (K1, K3 - K14 or F1 - F20)

&ERROR IN

The entry in &ERROR can be queried after the following statements:

CALL (not CALL *subprog-name*)
CASE
CYCLE FOR / WHILE
DISPLAY [FORM / LIST / SCREEN]
DO
ENTER
END CYCLE of a CYCLE WHILE or CYCLE FOR loop
END CYCLE of a CYCLE *cursor-name* loop with remote access
END DISPATCH (&ERROR cannot be polled following CALL statements which call programs in the remote system.)
END IF
EXECUTE (after EXECUTE and after EXECUTE with one of the executed statements)
FILL {FORM / LIST}
IF
PROCEDURE
SEND MESSAGE
SET
SYSTEM
SQL statements with an INTO clause
File processing statements
Remote access to a SESAM or UDS database

If IN (*error*, ...) is not specified, this has the same effect as specifying all possible entries for *error*.

error

error specified the entry defined for an error exit.

Refer to the “DRIVE Programming Language” manual [2] and the section on system variables for details on the entries in &ERROR which can be queried.

A literal must be specified for *error*.

&DML_STATE IN	The entry in &DML_STATE can be queried for all SQL statements and after any EXECUTE statement that executes an SQL statement. It can also be queried after END CYCLE within a "CYCLE cursor-name INTO" loop if the value of SQLCODE is less than 0. If IN (*status*, ...) is not specified, this has the same effect as specifying all possible entries for *status*.
status	*status* defines the entry for which an error exit is defined. Refer to the “DRIVE Programming Language” manual [2] and the section on system variables for details on the entries in &DML_STATE which can be queried. A literal must be specified for *status*. The message text of the SQL error can be determined using the string function SQLMSGSTRING; for more details, see page 165.
&WARNING IN	Warnings from the SQL2 interface can be evaluated with &WARNING. If IN (*warning*, ...) is not specified, this has the same meaning as specifying all possible specifications for *warning*.
warning	*warning* is used to define the entry for which an error exit is to be defined. Possible values: SQL WARNING: An SQL2 interface warning has been issued. This value is only set if no serious error exists, i.e. &ERROR and &DML_STATE have the value OK. The message text of the warning can be determined using the string function SQLMSGSTRING; for more details, see page 165.
CONTINUE	If a defined error event occurs, the program is continued. The system variable &ERROR_STATE is then supplied with the error information described above.
CALL *subprog-name*	The internal subprogram *subprog-name* is called when the defined error event occurs. The &ERROR_STATE system variable is then supplied with the error information described above. The program is aborted if, during processing of the internal subprogram, another error occurs for which an error exit has been defined with WHENEVER. The &ERROR_STATE system variable is then not updated in the internal subprogram.
BREAK	The program is aborted if a defined error event occurs.

Example

Statement	**Event**	**&ERROR=**	**&DML_STATE=**	**&WARNING**
SET &v=&a(&i)	INDEX ERROR	'INDEX ERROR'	unchanged	unchanged
OPEN *cursor-name*	SQL ERROR	unchanged	'SQL ERROR'	'OK'
FETCH *cursor-name* INTO ...	DIRTY READ	unchanged	'DIRTY READ'	'OK'
SELECT * INTO	SQL CONV WARNING	unchanged	'SQL CONV WARNING'	'OK'
SELECT ...	SQL WARNING	unchanged	'OK'	'SQL WARNING'

5.2 PAGE PRINT - Describe page background pattern

(See chapter 4, description of the PAGE PRINT statement on page 247)

The key word POSITION must be specified with the output position.

```
PAGE PRINT {print POSITION x y [ CM | INCH | UNITS ] ...
           ...
           }
```

Example

```
PAGE PRINT 'Sample-Report' POSITION 2 5 CM;
```

5.3 char-prim - String functions

(See chapter 5, description of the metavariable *char-prim* on page 283)

The DRIVE metavariable *charprim* is assigned the new function SQLMSGSTRING. This can be used to determine the SESAM message text in the event of SQL errors or SQL warnings.

```
charprim::={ ...|
             SQLMSGSTRING |
             ... }
```

SQLMSGSTRING — Determines the SQL error text or SQL warning text.

If this function is used in a SET statement, a DRIVE variable of data type CHAR(240) must be declared as the receive variable.

If an SQL statement is executed correctly, the receive variable is assigned blanks.

Example

```
...
FETCH CURSOR INTO &CURSOR.*;
DISPLAY FORM SQLMSGSTRING;
...
```

5.4 Omission of DRIVE statements and operands for FHS-DE

(See chapter 3, “DRIVE statements”)

Since FHS-DE is not supported, the associated DRIVE statements and operands have been omitted. Details are provided in the table below:

Statement	Page	Remark
ADD BOX	15	statement omitted
DISPLAY screenformat	92	CURSOR and MESSAGE operands omitted
REMOVE BOX	181	statement omitted
REPLACE BOX	184	statement omitted

6 Information on DRIVE programming

This chapter contains the changes and supplements to the manual “DRIVE Programming Language” [2]. The specified section and chapter numbers refer to this manual.

6.1 Data conversion with SQL statements

(New section)

6.1.1 Compatibility of data types and values

If values are used in calculations, predicates and assignments, the data types of the operands involved must be compatible.

Two data types are compatible if they fulfill the following conditions:

– Both data types are alphanumeric (CHARACTER or CHARACTER VARYING).

– Both data types are numeric (SMALLINT, INTEGER, NUMERIC, DECIMAL, XDEC, REAL, DOUBLE PRECISION or FLOAT).

– Both data types are DATE.

– Both data types are TIME.

– Both data types are TIMESTAMP.

If the data types are compatible, the compatibility of the value ranges is checked. A value is considered compatible with the value range of the target data type if it is an element of the value range.

Applying data conversion rules

Conversion rules are applied in the following situations:

– for all static or dynamic SQL statements with at least one DRIVE input or output variable

– for all SQL statements that refer to one value (constant, variable, expression, subquery).

6.1.2 Violation of conversion rules

If a value cannot be converted, the DRIVE program receives a corresponding value in &SQL_STATE. The following table explains all the values of SQLSTATE that can occur in association with the conversion.

SQLSTATE	SQL message text, meaning, and message time
00 000	“Execution of SQL statements successful”: Meaning: Value conversion was possible without restriction. Message time: DO
01 004	“Characters truncated at end of character string” Meaning: A problem exists in the class of the alphanumeric data types when reading. The length of the SQL value differs by at least one character from the length of the DRIVE variables. An abbreviated conversion is performed, indicated by an SQL warning. The DRIVE variable thus contains a right-justified, truncated alphanumeric character string. Message time: DO
22 001	“Significant characters of a string truncated to the right” Meaning: The value range is incompatible. A problem exists in the class of the alphanumeric data types when writing. The length of the assigned value differs from the length of the SQL column (at least 1 character too many). The conversion is aborted and the SQL statement is not executed. Message time: DO

SQLSTATE	SQL message text, meaning, and message time
22 003	“Numeric value too large or too small” Meaning: The value range is incompatible. A problem exists in the class of the numeric data types (reading or writing). The absolute value of the number to be converted does not fall within the value range of the target data type. The conversion is aborted and the SQL statement is not executed. Message time: DO
42 SR1	“Values cannot be compared” Meaning: The data types are incompatible. Value conversion between the SQL column and the reference data type is not possible (reading or writing). The SQL statement cannot be executed. Message time: Statically with COMPILE, dynamically with DO.

Reaction to conversion errors

The following list shows how DRIVE reacts to conversion errors and indicates the measures that can be taken in the program.

- SQLSTATE=01004

  With this SQLSTATE, the SQL statement was executed but an abbreviated data type was transferred.

  DRIVE redirects the warning to &DML_STATE exception handling SQL CONV WARNING. The default error exit for this warning is CONTINUE, i.e. the program is continued by default.

  Using WHENEVER &DML_STATE IN (’SQL CONV WARNING’) *action*, the program can take control and react to the warning as follows in an error exit, for example:
  – CONTINUE, if the program logic expects a warning
  – BREAK PROCEDURE, if the program logic does not expect a warning

- SQLSTATE=22001, 22003 and 42SR1 (dynamic)

  With these SQLSTATEs, the SQL statement was not executed.

  DRIVE redirects the information to &DML_STATE exception handling SQL ERROR. The default error exit for this warning is BREAK PROCEDURE, i.e. the program is aborted by default.

The program can take control with WHENEVER &DML_STATE IN ('SQL ERROR') *action*. However, it is not advisable to continue the program with CONTINUE. It is far more sensible to check the defined DRIVE variables and correct them if necessary. An implementation error may have occurred or an uncoordinated database modification may have been made.

6.1.3 Numeric data types in the SQL environment

(See section 3.2.1.2 on page 30)

The data types NUMERIC and DECIMAL, which provide accuracy of up to 31 places, are used for fixed point numbers with the highest possible accuracy. On the DRIVE side, the data type XDEC (eXtended DECimal) is available for these situations with the corresponding properties.

You should therefore always use XDEC when an SQL data field of type NUMERIC or DECIMAL is read or written using a DRIVE variable, because no other DRIVE data type provides the same level of accuracy; even DOUBLE, for example, is only accurate up to 16 places.

If you want to transfer values using variables in cases where maximum accuracy is required, please note the following:

- With static SQL statements, the data type XDEC can be used without restriction up to a maximum accuracy of 31 places.
- In dynamic SQL statements, the DRIVE data type XDEC and the SQL data types NUMERIC and DECIMAL only provide maximum accuracy when no more than 18 places are declared. This is because the ESQL-COBOL interface is used internally for dynamic SQL, and this interface only permits a maximum of 18 places.

  If more than 18 places are declared, the maximum possible number of predecimal places are first transferred to the COBOL declaration and only then are the decimal places transferred. If the statement is executed, behavior depends on the value transferred; see also examples 2 and 3:
  – If it is not possible to transfer all decimal places, the value is rounded off.
  – If it is not possible to transfer all predecimal places, the call is rejected with SQLSTATE=22003.

  To avoid these difficulties, you should issue a static call in such cases. This affects the SQL statements SELECT and OPEN CURSOR/FETCH for reading from the database, as well as INSERT INTO and UPDATE for writing to the database.

The restrictions described above do not apply when you transfer the DRIVE value using a numeric literal or a constant.

Examples

1. An SQL database field of type NUMERIC (30,0) is read statically with DRIVE. If you want to guarantee maximum accuracy, you must declare the DRIVE variable as follows:

   ```
   DECLARE VAR &VAR1 XDEC (30,0)
   ```

   If you declare &VAR1 as INTEGER, the read action is rejected with SQLSTATE=22003 (value too large).
   If you declare &VAR as DOUBLE, the value is read (SQLSTATE=OK), but only with an accuracy of 16 places!

2. An SQL database field of type NUMERIC (24,4) is read with a dynamic SELECT statement. This type is converted to COBOL-DECIMAL (18,0) at the ESQL-COBOL interface, i.e. 18 of the 20 possible predecimal places are permitted. This results in the following behavior:

   – A value with a maximum of 18 predecimal places and without decimal places is read without modification.

   – A value with a maximum of 18 predecimal places and with one or more decimal places is rounded off such that all decimal places are dropped.

   – A value with more than 18 predecimal places cannot be read, and the call is rejected with SQLSTATE 22003.

   The same applies when writing to the database.

3. A DRIVE field of type XDEC (20,3) is written to the database with a dynamic UPDATE statement. This type is converted to COBOL-DECIMAL (18,1) at the ESQL-COBOL interface, i.e. all 17 predecimal places and one decimal place are permitted. This has the following effect:

   – A value with a maximum of one decimal place is written to the database without modification.

   – A value with 2 or 3 decimal places is always rounded off to one decimal place.

   The same applies to reading from the database.

6.2 Intercepting errors and warnings, end criteria

You can use the DRIVE statement WHENEVER (see page 160) to intercept errors and react to warnings.

6.2.1 Reactions to errors

(Changes and supplements to section 4.2)

If an execution error occurs in a program, the program is aborted unless otherwise specified.

If you want to prevent the program from being aborted, you must define an error exit using the WHENEVER statement.

6.2.1.1 Reaction to execution errors

There are two types of error classes for execution errors:
ERROR (DRIVE error messages) and DML_STATE (database status messages)

You must use one of the following variants of the WHENEVER statement to define the error exit. The variant you use will depend on whether the error is of the class ERROR or DML_STATE:

– 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 subprog-name

– WHENEVER &DML_STATE IN (status, ...) CALL subprog-name

All values of the system variables &ERROR and &DML_STATE can be queried using WHENEVER (see the section “System variables” in the manual “DRIVE Programming Language” [2]).

Example

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

Reactions of the error classes ERROR and DML_STATE

If the statements are executed without error, then the fields &ERROR and &DML_STATE are assigned the value “OK”. &SQL_STATE is assigned the value “00000”.

If an execution error that belongs to the error classes described in the WHENEVER statement occurs, one of the following actions take place:

– Execution of the current statement is aborted regardless of the error class. Modified variable values are no longer valid, and the variables are assigned their old values.

  The DISPLAY SCREEN statement is an exception, since technical considerations require the data to be kept in a format specific to partial forms, rather than to statements.

– If no action has been defined for an error occurrence, then the program will abort, except for “TABLE END”, “DIRTY READ”, “SQL CONV WARNING, “TOO LONG” and “TOO SHORT”. The result is the same as for the BREAK statement.

– If CONTINUE has been specified, the program continues to execute with the statement that follows the incorrect statement.

– If CALL *subprog-name* has been specified, first the internal subprogram *subprog-name* is called and then the calling program continues to execute with the statement that follows the incorrect statement.

  If, however, a further error occurs during processing of the internal subprogram *subprog-name*, the program is aborted, regardless of any error exits that have been defined. When the SYSTEM statement is executed, the corresponding system variable is given the return code of the subsystem called, regardless of whether the statement executes with or without error (see the WHENEVER statement on page 160).

  During processing of the internal subprogram *subprog-name* the system variable &ERROR_STATE does not change. It is thus possible to query the error information contained in it in this subprogram. Even if an external subprogram is called in *subprog-name*, the contents of &ERROR_STATE remain unchanged.

– If the database system SESAM rolls back a transaction internally (SQLSTATE=40xxx), the DRIVE program cannot react to the SQLSTATE that initiated the internal rolling back of the transaction. In this case, you cannot program an error exit with WHENEVER.

6.2.1.2 Error exit for assigning invalid variable values

If a variable is assigned an invalid value, the program is aborted by default.
If you wish to prevent a program from being aborted, you must define a ‘CHECK ERROR’ or a ‘CONVERSION ERROR’ error exit with one of the following statements:

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

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

The following applies to both statements:

– the errored assignment is not executed

– the original contents of the variable are retained

– DRIVE supplies the system variable &ERROR_STATE with the following value: the &ERROR component is supplied with the value ‘CHECK ERROR’ or ‘CONVERSION ERROR’, the &VAR_NAME component is supplied with the name of the variable which caused the error and with LINE_NR, DISPLACEMENT STATEMENT.

6.2.1.3 Mapping SQLSTATE to &DML_STATE and &SQL_STATE

(New section)

The following table shows how the standardized SQL return code SQLSTATE is mapped to the DRIVE system variables &DML_STATE and &SQL_STATE.

SQLSTATE	&DML_STATE	&SQL_STATE	&SQL_CLASS	&SUB_CLASS
00000	'OK'	00000	00	000
01004	'SQL CONV WARNING'	01004	01	004
01SA1	'DIRTY READ'	01SA1	01	SA1
01xxx	'OK'	01xxx	01	xxx
02xxx	'TABLE END'	02xxx	02	xxx
07xxx	'SQL ERROR'	07xxx	07	xxx
21xxx	'SQL ERROR'	21xxx	21	xxx
22020	RAISE 4100	22020	22	020
22021	RAISE 4100	22021	22	021
22xxx	'SQL ERROR'	22xxx	22	xxx
23xxx	'SQL ERROR'	23xxx	23	xxx
24xxx	'CURSOR SQL ERROR'	24xxx	24	xxx
25xxx	'SQL ERROR'	25xxx	25	xxx
26xxx	'SQL ERROR'	26xxx	26	xxx

SQLSTATE	&DML_STATE	&SQL_STATE	&SQL_CLASS	&SUB_CLASS
28xxx	'SQL ERROR'	28xxx	28	xxx
2Dxxx	RAISE 4100	2Dxxx	2D	xxx
33xxx	RAISE 4100	33xxx	33	xxx
34xxx	RAISE 4100	34xxx	34	xxx
3Dxxx	'SQL ERROR'	3Dxxx	3D	xxx
3Fxxx	'SQL ERROR'	3Fxxx	3F	xxx
40xxx	'TA CANCELLED'	40xxx	40	xxx
42SE2	RAISE 4100	42SE2	42	SE2
42SH2	RAISE 4100	42SH2	42	SH2
42xxx	'SQL ERROR'	42xxx	42	xxx
44xxx	'SQL ERROR'	44xxx	44	xxx
51xxx	'ADMIN SYS ERROR'	51xxx	51	xxx
52xxx	'ADMIN SYS ERROR'	52xxx	52	xxx
55xxx	'ADMIN SYS ERROR'	55xxx	55	xxx
56xxx	'ADMIN SYS ERROR'	56xxx	56	xxx
57xxx	'ADMIN SYS ERROR'	57xxx	57	xxx
58xxx	'ADMIN SYS ERROR'	58xxx	58	xxx
59xxx	'SYSTEM ERROR'	59xxx	59	xxx
81SA2	'ADMIN SYS ERROR'	81SA2	81	SA2
81SA6	'TEMP SYS ERROR'	81SA6	81	SA6
81SB0	'TEMP SYS ERROR'	81SB0	81	SB0
81SB1	'TEMP SYS ERROR'	81SB1	81	SB1
81SB2	'TEMP SYS ERROR'	81SB2	81	SB2
81SB5	'DB NOT AVAILABLE'	81SB5	81	SB5
81SB7	'SQL ERROR'	81SB7	81	SB7
81SBA	'TEMP SYS ERROR'	81SBA	81	SBA
81SC7	'TEMP SYS ERROR'	81SC7	81	SC7
81SCA	'TEMP SYS ERROR'	81SCA	81	SCA
81SD2	'ADMIN SYS ERROR'	81SD2	81	SD2
81SP3	'TEMP SYS ERROR'	81SP3	81	SP3
81xxx	'SYSTEM ERROR'	81xxx	81	xxx
91xxx	'LIMIT REACHED'	91xxx	91	xxx
95xxx	'SQL ERROR'	95xxx	95	xxx

6.2.2 Reaction to SQL warnings

(New section)

SQL warnings occur when SQLSTATE=01xxx (xxx = subclass) is returned following the correct execution of an SQL statement. The warning can have two possible causes:

– warning relating to a read database record

– warning from the SQL interface

You can use the WHENEVER statement to react to these SQL warnings in DRIVE programs.

Warnings relating to a read database record

The following warnings can occur in relation to a database record read by DRIVE:

SQLSTATE=01004: “Character string truncated to the right”
SQLSTATE=01SA1: “Record modified by update transaction”

DRIVE redirects these warnings to the following &DML_STATE exception handling:

SQLSTATE=01004: &DML_STATE = ‘SQL CONV WARNING’, see also page 169
SQLSTATE=01SA1: &DML_STATE = ‘DIRTY READ’

The default error exit for both warnings is CONTINUE, i.e. the program is continued by default without further action.

You can react to these warnings using WHENEVER &DML_STATE IN (’...’) *action*.

Warnings from the SQL interface

From the point of view of SQL, a warning from the SQL interface indicates success, i.e. &DML_STATE=’OK’. 01xxx is returned in SQLSTATE (xxx not equal to 01004 or 01SA1). Here, xxx is the subclass that provides precise information on the cause of the warning.

DRIVE redirects these warnings to the following &WARNING exception handling:

&WARNING = ‘SQL WARNING’

The default error exit is CONTINUE, i.e. the program is continued by default without further action.

You can use WHENEVER &WARNING IN ('SQL WARNING') *action* to program an error exit. The following must be noted in this case:

– The WHENEVER statement in this form is used to react explicitly to warnings from the SQL2 subsystem. If you omit the IN clause, warnings are processed from all subsystems that can currently use this interface. Since other subsystems may be added in the future, you should always program the IN clause.

– If no warning scenario exists, &WARNING is assigned 'WARNING OK'.

– The warning exit can react to the warning as follows, for example:
  – CONTINUE, if the program logic expects a warning
  – BREAK PROCEDURE, if the program logic does not expect a warning

– The text of the SQL warning can be determined using the SQLMSGSTRING function; for more details, see page 165.

6.3 Dynamic SQL statements

(Supplement for section 4.6.1)

With EXECUTE, all the statements listed in chapter 4 can be executed dynamically. In this case, you use the string function to create a string at execution time which contains a dynamically executable SQL statement (see section 4.5.1, “String functions”, in “DRIVE Programming Language” [2]).

6.4 Abbreviation ".*"

(Supplement for section 3.5.1 on page 73)

In a structured variable &variable, the string &variable.* represents the abbreviated notation for the list of all components on the next level.

Example

```
DCL VAR
   1 &a,
      2 a1,
         3 a11 CHAR (1),
         3 a12 INT,
         3 a13 DATE,
      2 a2 CHAR (2),
      2 a3 INT;
```

`&a.*` is thus equivalent to `&a.a1,&a.a2,&a.a3;`

Usage of ".*"

The following table lists the DRIVE statements where ".*" can be used and indicates the restrictions that apply.

DRIVE statement	Restriction
CALL	not with RETURN
CASE	(1)
CYCLE	(1)
DCL FORM	not with RETURN
DCL LIST	not with RETURN
DISPLAY FORM	not with RETURN
DISPLAY LIST	(no restriction)
DO	(no restriction)
ENTER	(no restriction)
FILL	not with RETURN
IF	(1)
READ FILE	(no restriction)
SEND MESSAGE	(no restriction)
SET	only on the right-hand side within an aggregate
WRITE FILE	(no restriction)

(1) A * variable is only permitted in the condition in the following cases:
with '=' only on the right-hand side within an aggregate
with 'IN' only on the right-hand side as a list of values

6.5 Restrictions and incompatibilities

6.5.1 Declaration of variables with the LIKE TABLE construct

(Supplement for section 3.3.3 on page 49)

Up to 36 variables can be declared in a DRIVE program with the component structure of an SQL table:

```
DECLARE VARIABLE &variable LIKE TABLE table
```

If more than 36 variables are declared in this way, compilation of the DRIVE program (COMPILE) is terminated with error DRI0032. The compilation list contains the following error text from the 36th variable declaration with LIKE TABLE:

```
DRI0536 42SF0 Cursorname not unique
```

6.5.2 Unique names of DRIVE SQL programs

(Supplement for “DRIVE Directory” [3], chapter 3, description of the COMPILE statement)

If a DRIVE program is compiled with static SQL statements, this program creates an intermediate SQL module.

The name of this intermediate module can be up to 7 characters long and is formed from the original program name (= name of the PLAM library element). If the name is longer than 7 characters, it is abbreviated in accordance with the 4-3 rule, i.e. the first 4 and last 3 characters are used. The name is supplemented by an internal suffix so that the module can be uniquely identified in a DRIVE session despite the fact that it has been abbreviated.

Up to 191 variants are possible for this suffix.

```
COMPILE program ...
```

If more than 191 programs are executed in a DRIVE session with the same SQL module name, DRIVE outputs the following error message as of the 192nd program:

```
DRI0536 81SD1 Modulname used for two single sql-modules
```

6.5.3 Modified behavior of the WHENEVER statement

If a conversion error occurs with SQLSTATE=01004, the WHENEVER statement behaves differently than in the preceding version:

- If a DRIVE program contains the statement `WHENEVER &DMLSTATE` *action*, then *action* is now performed.
- If a DRIVE program contains the statement `WHENEVER &ERROR` *action*, then *action* is now not performed because SQLSTATE=01004 is no longer converted into a CONVERSION ERROR.

More details on SQLSTATE=01004 can be found on page 169.

6.5.4 Multiple variables in the SELECT list of an SQL statement

(Supplement for “DRIVE SQL Directory” [4], chapter 3, SELECT statement on page 134)

Multiple variables in the SELECT list are only permitted in a static SELECT statement.

```
SELECT select-list INTO,...
```

If a multiple variable such as `...,&var(1-3),...` is specified with a dynamic SELECT statement, this is rejected with SQLSTATE=42SL3 and the message text “?” placeholder in select list.

6.5.5 Insert column list with the SQL statement INSERT

(Supplement for “DRIVE SQL Directory” [4], chapter 3, INSERT statement on page 111)

In the case of the SQL statement INSERT, the total number of columns and variants (multiple columns) can be 1000 in the insert column list. This restriction applies both to static and dynamic INSERT statements.

```
INSERT INTO table (insert-column-list) VALUES ( ... )
```

If the list has more than 1000 columns/variants, the INSERT statement is rejected with SQLSTATE=07009. This limit may be exceeded in particular if multiple columns are used as shown in the following example.

Example

The following INSERT statement contains 1001 terms in the list and is therefore rejected with SQLSTATE=07009:

```
INSERT INTO table1 (
                     field1,
                     multi-field1(250),
                     multi-field2(250),
                     multi-field3(250),
                     multi-field4(250)
                    )
       VALUES       (
                     wert1,
                     &multivar1,
                     &multivar2,
                     &multivar3,
                     &multivar4
                    )
```

7 Databases

7.1 Processing databases

You will find an introduction to the structure of a relational database and to SQL terminology and concepts in the SESAM manuals “SESAM/SQL-Server V2 Reference Manual, Part 1” [6] and “Core Manual” [8].

This chapter provides you with an overview of the most important processing functions.

A database is a “physical” database that can be queried and updated. It comprises not only the rows but also a description of the rows and their logical organization (metadata). A relational database is logically organized in base tables and system tables.

Base tables are defined in the relational schema of the database.

The rows in a database can be read (selection), or inserted, deleted and modified (manipulation).

Selection means reading a row from the database by means of a SELECT statement. If more than one row is found, DRIVE issues an error message.

If you wish to read several rows, a result table (cursor table) must be specified, which contains the selected rows. This result table is generated by declaring a cursor using the *cursor description* metavariable (see the “DRIVE-SQL Directory” [4], *cursor description* metavariable).

Manipulation means that it is possible to use a **single** SQL statement to update several rows, all satisfying given conditions in the **same** way (**set-oriented update**, see the “DRIVE-SQL Directory” [4], *query_expression* metavariable).

If, however, you wish to update **several** rows in **different** ways, the rows must be included in a result table (cursor table), and the cursor positioned on the individual rows (**single-row update**). The base table for which the cursor was declared is updated.

You also need a cursor table when you change or delete one or more rows and do not wish to set the conditions until execution.

When **inserting** records into the database, the RETURN clause of the INSERT statement allows the DRIVE user to output the number assigned by the database system to identify a record in a DRIVE variable using the contents of the count field with the compound key (specification of * in the VALUES clause).

Rows are selected from a table by specifying conditions in the WHERE clause of a SELECT statement or in the *query_expression* metavariable. If you wish to specify several conditions, use the logical operators AND and OR (see the “DRIVE-SQL Directory” [4], *condition* metavariable). It is also possible for the result tables (views, cursor tables) to contain rows from several tables. The tables will be joined (Join) if:

1. all tables involved are listed in the FROM clause of the SELECT statement or the metavariables *select_expression* and
2. either a record element of a table is joined with the record element of another table by the relational operator “=” (whereby the record elements which are compared must have the same data type)

   or, if inside the FROM clause, a join condition is referenced through a ON clause.

Before you can select or manipulate the rows in a cursor table, you must open the cursor table with OPEN and position the cursor on the individual rows with FETCH.

The current cursor position can be stored for use after the end of the transaction with STORE and restored in a later transaction with RESTORE.

When working with a cursor table, the statements must be specified in the following order:

DECLARE *cursor*....

OPEN *cursor*...

FETCH *cursor*...

Use DELETE, UPDATE to edit the cursor table, if the cursor can be modified

STORE *cursor*

RESTORE *cursor*

CLOSE *cursor*

DROP CURSOR *cursor* (as required)

Views are virtual tables that define a section of one or more base tables.

When a view is declared, it is given a name, and a database query is specified with a *query_expression*. The view comprises the result table of this database query. The contents of the view are not determined until a statement referencing this view is executed. If the view can be updated, you can use the view to insert, update or delete rows in the base table.

Transactions ensure that the data in the database remains consistent. A transaction is a logical sequence of statements that are located between two successive synchronization points. The statements in a transaction are either executed in their entirety or not at all. The start of a transaction is not defined by a particular SQL statement. The first SQL statement that initiates a transaction (see “DRIVE Programming Language” [2], section 10.1.2) after the program start or after the previous transaction has been committed or reset is evaluated by the database system as the start of the transaction.

Error handling

In program mode, the response to errors for certain error classes and DRIVE statements can be defined with the WHENEVER statement (see section 6.2, “Intercepting errors and warnings, end criteria”, on page 172). This means that DRIVE does not automatically respond by aborting the program but instead executes the actions specified for the WHENEVER statement.

You can query errors in distributed processing by using the system variable ERROR_STATE.ERROR set to the value 'DIS ERROR.' The variable &DISTRIBUTION_STATE is also supplied (see “DRIVE Programming Language” [2], chapters 12 and 13).

Representing the null value

To set one character to represent the null value on the screen as an alphanumeric and/or numeric data field, use the statement PARAMETER DYNAMIC NULL FORM (see also the statement PARAMETER DYNAMIC in the “DRIVE Directory” [3]).

Any character can be used as a null value character in alphanumeric data fields. However, in a numeric data field, only a numeral or one of the special characters * + - , . is permitted.

The null value character for alphanumeric data fields is also valid for fields of the Time data type. The null value character for numeric data fields is also valid for fields of the INTERVAL data type.

The null value character set with PARAMETER DYNAMIC can be written over for DRIVE screen forms by substituting another null value character in the NULL clause of the DECLARE FORM statement.
A null value character which was set using DECLARE FORM NULL is only valid for the particular screen form defined by DECLARE FORM.

If you do not explicitly set a null value character, then the @ character becomes the default value for alphanumeric or numeric data fields.

Correspondingly, you can use the PARAMETER DYNAMIC NULL LIST statement to set a character to represent the null value for printer output. The default value is a period.

7.2 Database support

This chapter describes:

– the objects, which the individual database systems recognize and how they can be addressed (page 186)
– special characteristics of SESAM V2 (page 189)
– syntax differences between SQL dialects and DRIVE, especially DRIVE support of cursor processing (page 192)
– data security using access control and user passwords in SESAM/SQL (page 196).

Database applications can be quickly and easily developed using the fourth-generation language (4GL) DRIVE. Databases are accessed using SQL (Structured Query Language), which is the most widely used relational database language.

DB servers supported

Using SQL statements DRIVE makes it possible to access the DB server SESAM/SQL V2.

You can only work with one BS2000 server in a DRIVE session.

The distributed transaction processing function can be used to program distributed DRIVE applications, which can access all BS2000 servers (see “DRIVE Programming Language” [2], chapter “Distributed transaction processing (DTP)”).

References to other manuals

An extensive description of SQL statements can be found in the “SESAM/SQL-Server - SQL Reference Manual, Part 1” [6].

The exact syntax for DRIVE-SQL statements for each of the interfaces can be found in condensed form in “DRIVE-SQL Directory” [4].

Complex components of SQL statements which are found in both DRIVE and SQL statements are described separately in the chapter dealing with metavariables in the “DRIVE Directory” [3] and also in “DRIVE-SQL Directory” [4]).The DRIVE metavariables largely correspond to those described for SESAM V2, i.e. there is a unique assignment.

7.2.1 SQL objects in DRIVE

A DB server can be addressed using transaction statements as well as references to databases or SQL objects which are managed by this DB server (DBH). **Persistent** objects, i.e. those which are stored in the database, consist of the base tables of applications, DB server system tables, columns, constraints, indexes, views and synonyms. The names and structures of persistent SQL objects are metadata of the particular database and are

managed by the database system's system tables. As an application, DRIVE manages no information about persistent SQL objects. It is possible to define variables with the structure of base tables, system tables or views using the statement DECLARE VARIABLE... LIKE TABLE.

Names and structures of **temporary SQL objects** belong to the application-specific data and are managed by the SQL runtime system of the DB server. They no longer exist after the end of the DRIVE session.Temporary SQL objects are cursors, as well as temporary tables and correlations. It is possible to define variables with the structure of DB tables or cursor tables using the statement DECLARE VARIABLE... LIKE TABLE.

Assigning a database to DRIVE supplies all necessary data and metadata, i.e. all base and system tables and system views of an SQL schema. Use the statements PARAMETER DYNAMIC or OPTION with the operands CATALOG, SCHEMA and AUTHORIZATION to assign the current database (see the manual "DRIVE-SQL Directory" [4]).

Use the table name and the table structure to specify a **base table**. Use the columns and the table constraints to specify the table structure. Optionally, the kind of physical storage can be specified. The following table shows how to qualify the table names for the DB server supported by DRIVE:

Database server	**Table name**
SESAM V2	[catalog-name .] [schema-name .] tablename two relations are identical if they have the same catalog, schema and table names

The following table shows the storage options for the DB server supported by DRIVE:

Database server	**Storage options**
SESAM V2	USING SPACE [catalogname .] space

The (possibly structured) **column** of a base table is specified by the column name, the data type, an optional default value and an optional column constraint. The column name is unique across all tables. A column (or one or more column components) is specified outside of a table specification using the metavariable record element (see the manual "DRIVE-SQL Directory" [4]).

The following access control is used: an authorized user (grantor) accords **privileges** and allows another user (grantee) to apply a particular statement to a particular SQL object, e.g. to apply SELECT to the table of a schema. The GRANT statement accords privileges and the REVOKE statement revokes privileges and, together, they regulate access in SQL.

A database can be divided into **schemas**, which are established by users or by the database system. A schema defined by a user is assigned to that user who is its owner. It includes metadata of base tables, views, indexes and privileges. System-defined schemas summarize system views and contain database-specific metadata.

Views are virtual tables which define a subset of one or more base tables. In SESAM V2 there are persistent views (see the following section). The view name is unique with respect to all persistent views and tables throughout the database.

Temporary **synonyms** for tables and view names (called “correlations”) can be used in a search query. These synonyms only exist for the life of the query.

Table and column **constraints,** which consist of optional constraint names and the column identifier, can be defined (see CREATE TABLE on page 115 and ALTER TABLE on page 89). The constraint name is unique across the database. For SESAM V2, DRIVE syntax does not currently support indexes. Base tables can be used in DRIVE even though they have been specified with indexes outside of DRIVE.

With SESAM V2 you can define **persistent views** with a somewhat limited SELECT statement which references base tables and other persistent views (see the manual “DRIVE-SQL Directory” [4], CREATE VIEW). The “SQL syntax of SESAM V2” [8] contains a summary of system views in SESAM V2.

7.2.1.1 Defining SQL objects with DDL statements

All user data are organized logically in base tables, which the SESAM V2 user creates using DDL (Data Definition Language) statements.

DRIVE offers additional DDL statements for defining data structures and integrity conditions. These statements are executable SQL statements and must therefore appear in the program body (see chapter 4, “DRIVE SQL statements”, on page 83 and the manual “DRIVE-SQL Directory” [4]).

Pay attention that a DDL statement in the program body does not destroy a declarative statement in the declaration section: e.g. setting a cursor in the declaration section and updating a table on which the cursor has been declared using ALTER TABLE statement in the program body. Attempting to use the cursor to access the table can result in an SQL error.

The DDL statements can also be specified dynamically, i.e. at the time of execution (see the manual “DRIVE Directory” [3], EXECUTE statement).

DML and DDL statements cannot be combined in the same transaction; this means that a transaction must contain only DDL statements or no DDL statements at all.

7.2.2 Information on database support

DRIVE V2 supports the DB server SESAM V2 with

– all numeric, alphanumeric and time expressions (see the manual “DRIVE-SQL Directory” [4], metavariable *sql_expression*)
– all conditions (metavariable *condition*)
– all query blocks (metavariable *query_expression*).

The statement DECLARE VARIABLE...LIKE TABLE/CURSOR... maps SESAM data types to DRIVE data types (see “DRIVE Programming Language” [2], chapter “Using variables and constants“. DRIVE data types are assigned to the SELECT list of cursor descriptions which correspond to the SESAM V2 conventions, i.e. in agreement with the SQL norm (see the manual “DRIVE-SQL Directory” [4], the DECLARE CURSOR statement and metavariable *query_expression*).

SESAM V2 uses both of the data schemas

– INFORMATION_SCHEMA (user-specific access)
– SYS_INFO_SCHEMA (access limited to the universal user)

To query data from these schemas use SELECT statements and cursor declarations (see the manual “DRIVE-SQL Directory” [4]).

Databases with SESAM V2 also involve catalogs. A catalog can contain several SQL schemas with several base tables. There are simple table names and table names which are qualified by catalog and schema names (see page 191).

The PERMIT statement has no effect in SESAM V2 in the new style; all it does is set the SQLSTATE. However, in the old style it is needed to access password-protected CALL DML tables. The GRANT statement accords privileges and REVOKE withdraws them.

There are the following statements for setting up a session:

– SET CATALOG - sets database names (only for dynamic SQL)
– SET SCHEMA - sets schema names (only for dynamic SQL)
– SET SESSION AUTHORIZATION (sets the routing code for the current SQL session)

For information on the topics “consistency level” and “starting a transaction” see “DRIVE Programming Language” [2], chapter “Transaction processing“.

7.2.2.1 Allocating the SESAM V2-DBH

DRIVE accesses a SESAM V2 database using an allocated DBH (Database Handler). The following conditions for use need to be observed:

– local access in BS2000 (in a TIAM or UTM application)

  DRIVE is connected using a connection module (SESMOD in TIAM, SESUTMC in UTM mode) which is supplied at startup with the name of an “independent DBH” and the configuration name. This DBH was previously started with a suitable configuration file. The DBH allocated can manage up to 254 databases and the data and metadata in them can be accessed in a DRIVE session (TIAM mode) or in a UTM conversation (UTM mode). This presupposes that the necessary access authorizations are present.

  Using SESAM DCN with distributed databases allows one to work with this “home DBH” as well as with additional “remote DBHs” and to have access to the databases managed by them. The access remains transparent to the DRIVE user. At any given point, a database can be managed by only one DBH at a time.

– distributed DRIVE applications with UTM-D (Distributed Transaction Processing)

  A UTM application that addresses the UTM-D application on the other BS2000 server, and vice versa, is used as the server in BS2000. The assignment of DBH and databases is determined by the generation of the UTM-D application in the client/server (job submitter/receiver).

Authorizing access to a SESAM V2 database

When a bona fide BS2000 system user creates a database with the SESAM V2 UTILITY statement CREATE CATALOG, a BS2000 password can and a routing code must be entered. The password is inherited by all files in the database and must be entered whenever starting a DBH which manages the database. The routing code indicates a SQL user who possesses a BS2000 system user ID. The default is the ID used when the CREATE CATALOG statement was entered.

A routing code together with a system user ID permits system access. The system access established with CREATE CATALOG is the first for the database and identifies the so-called universal user. Using the UDL statement CREATE USER, this user can establish additional routing codes, while with the CREATE SYSTEM statement routing codes can be expanded for system access.

In order to access a SESAM V2 database with a DRIVE program, system access must meet the following criteria, depending on the form of access employed:

– for local access in BS2000 the system user ID must match the ID in DRIVE as either a TIAM or a UTM application

– for distributed applications only the client must be identified as a SQL user

In DRIVE the identity of SESAM V2 users is verified using the AUTHORIZATION operand of the PARAMETER DYNAMIC or OPTION statement. The specification AUTHORIZATION is valid for the DRIVE session and can be changed when there is no transaction using the DRIVE-SQL statement SET SESSION (see the manual “DRIVE-SQL Directory” [4]).

7.2.2.2 Listing catalogs and SQL schemas

The organization of tables, views and indexes in databases and schemas means that each of these SQL objects can only be uniquely identified in relation to a specific catalog and a specific SQL schema: the simple object name is qualified with the schema name and the catalog name `[ [catalog-name . ] [schema-name . ] ]`.

If the catalog name and/or the schema name are not given, then the standard default settings hold. For static SQL these are taken from the CATALOG and SCHEMA operands of the OPTION statement; for dynamic SQL they are set with the SET CATALOG and SET SCHEMA statements or using the PARAMETER DYNAMIC statement (see the manual “DRIVE-SQL System Directory” [4]).

To maintain clarity in a DRIVE application with SESAM/SQL V2 it is recommended to structure the application into individual programs, so that each program chiefly accesses SQL objects of one schema on a static basis. This schema and its catalog can then be selected using the OPTION statement. Access to SQL objects of another schema of the catalog selected or of schemas of another catalog must then use qualified object names.

7.2.3 Syntax differences between SQL dialects and DRIVE

This section lists the essential functions and syntax elements which DRIVE supports in addition to or instead of those of the particular SQL language descriptions of the DB server.

- Variables

  In DRIVE the term “variable” stands for a user variable. i.e. for a variable which the programer defines in the declaration section. The DRIVE meta-variable *varname* designates a user variable. It must have the prefix “&”.

  In DRIVE statements the term “indicator variable” designates a variable which, together with the USING clause, is necessary in parameter transfers into other programming languages (see “DRIVE Programming Language” [2], chapter “Using variables and constants“). The indicator value shows whether the assigned parameter value is to be interpreted as null value or not.
  The semantics of the indicator variables is expanded in SESAM (see “DRIVE-SQL Directory” [4]). No indicator variables are necessary within SQL statements in DRIVE. What is more, DRIVE variables always show a null value.

  With the clauses LIKE TABLE and LIKE CURSOR of the DECLARE VARIABLE statement you can easily define a structured variable &varname whose structure corresponds to the table (base table, system table, view) selected or to the cursor (derived table) selected. It is then a simple matter to specify a variable list which corresponds to the column list of the table or cursor using &varname.* within SQL statements.

  Examples

  ```
  INSERT INTO TABLE table VALUES (&varname.*);
  FETCH cursor INTO &varname.*;
  ```

- Interactive and program modes

  There is a difference in the statements FETCH and SELECT in the interactive and program modes of DRIVE:
  In interactive mode the values of record elements are displayed on the screen. The INTO clause is not permitted. The INTO clause must be set in program mode. The INTO clause carries over the values of record elements into variables.

- Transaction management

  DRIVE offers the WITH RESET clause for the statement ROLLBACK WORK. The simple ROLLBACK statement sets all database statements in the present transaction back to their initial state. Using the RESET clause results in the DRIVE transaction being rolled back as well. This means that the contents of variables and forms will be rolled back to their state at the previous COMMIT statement and program execution will be resumed after the next COMMIT WORK (see “DRIVE Programming Language” [2], chapter “Transaction processing” and, in a distributed environment, see the corresponding chapter “Distributed transaction processing (DTP)“).

- DRIVE statements and clauses for cursor

  DRIVE offers a DECLARE CURSOR statement. This statement has been expanded with regard to the SQL norm and leads to different kinds of cursor types:

Cursor type	**DECLARE statement**
Permanent cursor	with PERMANENT
Temporary cursor	with TEMPORARY (default)
SCROLL cursor	with SCROLL
Block mode cursor	with PREFETCH n
UPDATE cursor	with FOR UPDATE
Static cursor	in the declaration section of a DRIVE program
Dynamic cursor	in interactive mode or with an EXECUTE statement upon executing
Variable cursor	in the declaration section of a DRIVE program without the FOR clause and upon executing with an EXECUTE statement supplemented by a dynamic FOR clause

  – The DRIVE statement CYCLE *cursorname* INTO *variable* in a cursor loop makes it easy to read database records into a variable list (see the manual “DRIVE System Directory” [3]).

  – block mode

    The PREFETCH clause for editing cursors in the block mode with SESAM V1 uses the same semantics as with SESAM V2. In addition, DRIVE with SESAM V2 offers the option of activating block mode with a PRAGMA statement (see page 141).

  – For information about the variable cursor see “DRIVE-SQL Directory” [4]. For information on the scope of the variable cursor see “DRIVE Programming Language” [2], chapter “Effects of transaction processing on definitions“.

  – The DRIVE statements DROP CURSOR *cursor* and DROP CURSORS delete the cursor. The statements can be carried out dynamically, i.e. created and executed at execution time. This means that DROP can be used to delete the variable cursor and dynamic cursor.

  – In program mode the cursor defined with PERMANENT is retained with its cursor position even after the program is ended, if the program was started with CALL. Any cursor defined with TEMPORARY is lost along with its position when the program ends.

  – A maximum of twenty non-static SESAM cursors can be defined in a DRIVE session using the interactive and program modes.

- Representing null values

  DRIVE offers the choice of using the statement PARAMETER DYNAMIC NULL to output the display of **null values** in other than standard form (in screen forms the character @, in printer output the period (.), see “DRIVE Directory” [3], PARAMETER DYNAMIC statement). The null value output display set with PARAMETER DYNAMIC can be overwritten in DRIVE screen forms by choosing another null value display in the NULL clause of the DECLARE FORM statement. If the null value display is not set to match the PARAMETER or DECLARE FORM statement, then the standard default will be used.

- Scope of the cursor

  A dialog cursor is valid between the DECLARE and the DROP statements. It is not valid in any program.

  A static or variable program cursor is valid in the program in which it was declared: in the declaration section after its DECLARE statement, in the body and in each internal subprogram.

  At the time a program executes, a dynamic program cursor is valid for all statements which execute after the program is declared. Variable program cursors behave in the same way as dynamic cursors with respect to the DB server.

  See also “DRIVE Programming Language” [2], chapter “Transaction processing“.

7.2.4 Life span of the cursor

The life span of a dialog cursor lasts from the DECLARE statement until deletion by the user with DROP or by DRIVE at the end of the DRIVE session.

The life span of a static program cursor begins at compilation time of its DECLARE statement and at execution time when the program in which it is declared is called with DO or CALL.

The life span of a variable program cursor in DRIVE begins with the static declaration of its name, in the DB server with the dynamic declaration of its FOR clause.

During a straight compile run using the COMPILE statement, all program cursors cease to exist upon compilation. When a compilation is followed by execution (DO, DEBUG, CALL statements), then the cursors are still present at execution time.

Upon ending a transaction with COMMIT WORK, DRIVE deletes temporary static, dynamic or variable program cursors at the next higher program level or at the latest upon switching to interactive mode. Upon switching to interactive mode DRIVE deletes permanent program cursors if the program was called with CALL and no COMMIT or ROLLBACK WORK occurred in the calling program. That means that permanent, open cursors keep their

position unchanged at program termination. Dynamic and variable cursors can also be deleted by users (see the DROP CURSOR statement in the manual “DRIVE-SQL Directory” [4]).

For more information about the life span of cursors see “DRIVE Programming Language” [2], section “Effects of transaction processing on definitions“.

7.2.4.1 Cursor position

The cursor position is a reference to a position before at or just after exactly one row in the cursor result table. It is defined only as long as the cursor exists and is influenced by the following statements:

Statement	**Cursor position after successful execution of the statement**
BREAK CYCLE	unspecified
CLOSE	unspecified
CYCLE	on the next row or unspecified
DECLARE CURSOR	unspecified
DELETE ...WHERE CURRENT OF...	in front of the next row or behind the previous row
DROP CURSOR(S)	undefined
END CYCLE	unspecified
FETCH	on the positioned row
OPEN	before the first row
RESTORE	after the last row in FETCH
STORE	before the next or after the last row and saved (no further FETCH possible))
UPDATE...WHERE CURRENT OF...	on the last row in the FETCH

DRIVE closes any open temporary static program cursors when the program in which they were declared closes or if the program aborts. Open permanent cursors, however, remain open and thereby retain their cursor position. The consequences of transaction statements for the life span and position of a cursor are described in “DRIVE Programming Language” [2], chapter “Transaction processing“.

7.2.5 Access control

DRIVE supports the control mechanisms of database system SESAM V2 and contains instructions for granting or producing access authorization.

The following DRIVE statements are valid for these database systems:

Statement	**Function**
PERMIT	Specification of user identifications that are checked with old-style access to CALL DML tables; only affects old style.
GRANT	Assignment of access rights to databases, base tables or views, as well as for storage groups; only affects new style.
SET SESSION AUTHORIZATION	Definition of current authorization key for dynamic SQL statements of a DRIVE program; only affects new style.
PAR DYN AUTHORIZATION	Definition of current authorization key for SQL statements of a DRIVE session.

Transaction recovery, which also involves access control, is discussed in detail in “DRIVE Programming Language” [2], chapter “Transaction processing”.

8 Appendix

8.1 Y2K support for old-style DRIVE

In the old-style variant, DRIVE provides the following functions and date variables:

Language	Interface	Four-digit year representation possible	Work-around
DRIVE old-style	$GDATE	no	Use SF2GDAT4 program
	$IDATE	no	Use SF2IDAT4 program
	&DATE	no	Use SF2DATE program

In its old-style variant, DRIVE provides the programs SF2GDAT4 and SF2IDAT4 for converting two-digit years to four digits, as well as the program SF2DATE that returns the current date with a four-digit year.

Program name	Parameters	Function
SF2GDAT4	P1, P2, P3	Converts the P2 value from a standard date with two-digit year (YYMMDD) to DD.MM.YYYY format, depending on the value of P3.
SF2IDAT4	P1, P2, P3	Converts the P2 value from a German-style date format with two-digit year to the format YYYYMMDD, depending on the value of P3.
SF2DATE	P1	Returns the current date in YYYYMMDD format.

8.1.1 SF2GDAT4 - Year conversion

The program SF2GDAT4 converts the date in *parameter-2* from the standard date format with a two-digit year (YYMMDD) to a German-style date format with a four-digit year (DD.MM.YYYY).

The century is determined by the value in *parameter-3* according to the following algorithm ('sliding window'):

- Last year of the 100-year interval:
  Y1Y2Y3Y4 = current year (4-digit) + value in *parameter-3*

  Example: If the current year is 1999 and the value in parameter-3 is 30, then this results in a 100-year interval from 1930 through 2029.
- When *parameter-1* is returned, YYYY is computed as follows:

  100 * Y1Y2 + YY if Y3Y4 ≥ YY
  100 * (Y1Y2 - 1) + YY if Y3Y4 < YY

See below for examples.

```
CALL SF2GDAT4 USING &par1=&par1, &par2, &par3;
```

&par1 Returns the date entered with *&par2* with four-digit year in the (German-style) format DD.MM.YYYY where DD and MM correspond to the MMDD specification in *&par2*.

&par1 must be an alphanumeric variable with a length of 10 bytes.

In case of an error, '0' is returned in *&par1*.

&par2 Specifies a date in format (YYMMDD).

&par2 must be alphanumeric variable or a literal with a length of 6 bytes. The value of *&par2* must be a positive integer smaller than 1000000. The program does not check if the value of *&par2* is a valid date.

&par3 Specifies a value for how far the 100-year interval should reach into the future from the current date.

&par3 must be an alphanumeric variable or a literal with a length of ≤ 4 bytes. The value of &par3 must be an integer. The sum of the current year and the value of *&par3* must be less than 10000.

Example

```
PROC;
CRE VAR &AKTDAT  CHAR(8);
CRE VAR &AKTJAHR NUM(2);
CRE VAR &DATUM6  CHAR(6);
CRE VAR &DATUM10 CHAR(10);
CRE VAR &DELTA   NUM(4);

/*  Computing the date with a sliding window:                              */
/*                                                                         */
/* The 100-year interval that includes the computed year should range from */
/* (current year - 49) to (current year + 50)                              */

SET &DATUM6  = '721201';
SET &DELTA =  50;
...
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (1)
...

/* The 100-year interval that includes the computed year should range from */
/* (current year - 109) to (current year - 10)                             */

SET &DATUM6  = '921201';
SET &DELTA =  -10;

CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (2)

/*  Computing the date with a fixed window:                                */
/*                                                                         */
/* The 100-year interval that includes the computed year should range from */
/* 1950 to 2049.                                                           */
/* The last year of the 100-year interval is computed relative to the      */
/* current year.                                                           */

CALL SF2DATE USING &AKTDAT=&AKTDAT;      /* Current date with 4-digit year */
SET &AKTJAHR = &AKTDAT(P=1,L=4);
SET &DELTA  = 2049 - &AKTJAHR;
SET &DATUM6   = '501201';

CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (3)
.
SET &DATUM6  = '011201';
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (4)

...
```

```
/*                                                                    */
/* The 100-year interval that includes the computed year should range 1890 */
/* to 1989.                                                           */
/* The last year of the 100-year interval is computed relative to the      */
/* current year.                                                      */

CALL SF2DATE USING &AKTDAT=&AKTDAT;      /* Current date with 4-digit year */
SET &AKTJAHR = &AKTDAT(P=1,L=4);
SET &DELTA  = 1989 - &AKTJAHR;
SET &DATUM6   = '891201';
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (5)
...
SET &DATUM6  = '901201';
CALL SF2GDAT4 USING &DATUM10=&DATUM10, &DATUM6, &DELTA;                  (6)
```

In the year 1997, the program returns the following results:

```
(1) 12.01.1972
(2) 12.01.1892
(3) 12.01.1950
(4) 12.01.2001
(5) 12.01.1989
(6) 12.01.1890
```

In the year 2050, the program returns the following results:

```
(1) 12.01.2072
(2) 12.01.1992
(3) 12.01.1950
(4) 12.01.2001
(5) 12.01.1989
(6) 12.01.1890
```

8.1.2 SF2IDAT4 - Year digits conversion

The program SF2IDAT4 converts the date specified in *parameter-2* with a German-style format and a two-digit year (DD.MM.YY) into a standard date with a four-digit year (YYYYMMDD).

The century is determined by the value in *parameter-3* according to the following algorithm ('sliding window'):

- Last year of the 100-year interval:
  Y1Y2Y3Y4 = current year (4-digit) + value in *parameter-3*

  Example: If the current year is 1999 and the value in *parameter-3* is 30, then this results in a 100-year interval from 1930 through 2029.
- When parameter-1 is returned, YYYY is computed as follows:

  100 * Y1Y2 + YY if Y3Y4 ≥ YY
  100 * (Y1Y2 - 1) + YY if Y3Y4 < YY

See below for examples.

```
CALL SFIGDAT4 USING &par1=&par1, &par2, &par3;
```

&par1 Return the date entered with *&par2* with a four-digit year in the format YYYYMMDD where MM and DD correspond to the MMDD specification in *&par2*.

&par1 must be an alphanumeric variable with a length of 8 bytes.

In case of an error, '0' is returned in *&par1*.

&par2 Specifies a date in format DD.MM.YY.

&par2 must be an alphanumeric variable or a literal with a length of 8 bytes. The value of *&par2* must be date specification in the format DD.MM.YY. The program does not check if the value of *&par2* is a valid date.

&par3 Specifies a value for how far the 100-year interval should reach into the future from the current date.

&par3 must be an alphanumeric variable or a literal with a length ≤ 4 bytes. The value of *&par3* must be an integer. The sum of the current year and the value of *&par3* must be less than 10000.

Example

```
PROC;
CRE VAR &DDAT8  CHAR(8);
CRE VAR &DATUM8 CHAR(8);
CRE VAR &DELTA  NUM(4);
...

SET &DDAT8 = '01.12.59';
SET &DELTA =  50;
...
CALL SF2IDAT4 USING &DATUM8=&DATUM8, &DDAT8, &DELTA;                    (1)
...
SET &DDAT8 = '01.12.47'
SET &DELTA =  -50;
...
CALL SF2IDAT4 USING &DATUM8=&DATUM8, &DDAT8, &DELTA;                    (2)
...
```

In the year 1997, the program returns the following results:

```
(1)  19591201
(2)  18471201
```

In the year 2009, the program returns the following results:

```
(1)  20591201
(2)  19471201
```

You can find more detailed examples in the description for the SFGDAT4 program.

8.1.3 SF2DATE - Current date

The program SF2DATE returns the current date in the format YYYYMMDD.

```
CALL SF2DATE USING &par1=&par1;
```

&par1 Returns the current date with a four-digit year in the format (YYYYMMDD).

&par1 must be an alphanumeric variable with a length of 8 bytes.

Example

```
PROC;
CRE VAR &AKTDAT CHAR(8);
...

CALL SF2DATE USING &AKTDAT=&AKTDAT;
```

On December 1, 2000 the program returns the following result: `20001201`

8.2 DRIVE web interface

Using the Web*Transactions* product line, data from existing DRIVE applications can be presented on the internet. This means that all data that can be generated from SESAM, from LEASY and from BS2000 files is also available over the internet using standard web browsers.

Starting with the automatic generations of HTML pages, you can improve the graphic interfaces bit by bit.

You can keep the traditional look & feel or offer new interfaces together with new interactive techniques. All user interfaces are maintained centrally on the server and are therefore available to all clients immediately, even after a change.

Web*Transactions* - the WWW access also for DRIVE server applications

You can choose any browser you wish. Whether Netscape, Microsoft Internet Explorer or Mosaic, Web*Transactions* works with all of them.

The Web server software (HTTP daemon) provides in connection with Web*Transactions* a communications link to server applications. Besides static HTML documents, Web*Transactions* can also generate information that changes or is interactive as a result of user input, i.e. dynamic content.

The application user interfaces can be easily an dynamically converted into HTML formats that are then passed on to the browsers for display. There is no need to change the server application's logic since Web*Transactions* represents merely an additional user as far as the application is concerned. A big advantage, since the old application with its alphanumeric interface remains usable despite the introduction of new graphical WWW interfaces.

Furthermore, you can optionally change the presentation logic of your DRIVE application with Web*Transactions*, i.e. you can combine screens, separate or even skip them.

Web*Transactions* is also available for the BS2000/OSD system platforms and can be linked via the CGI interface to the HTTP daemon that is integrated into BS2000/OSD V2 and higher.

For further information about Web*Transactions*, check
`http://www.siemens.de/webta.html`.

8.2.1 Procedure

The following steps must be carried out in order to convert a DRIVE application for web operation:

1. Design screens with IFG or use existing screens
2. Implement the DRIVE application or convert existing DRIVE application to web operation.Please note the following:
   - Screen output allowed only via FHS-EUA masks (= #-formats)
   - The DRIVE application must be closed with STOP WITH DISPLAY SCREEN
   - It is recommended to intercept all exceptions, if possible, with WHENEVER and to output the error message via an error screen. Events that are not intercepted are acknowledged by DRIVE via the DRIVE-standard HTML page WEBERR.htm.

   For further information see section 8.2.2, “Preparations and tips for use”, on page 206.
3. Conversion of FHS formats to corresponding HTML documents

   To convert the various format, Web*Transactions* provides standard services. From an FHS format, you must create via IFG a C addressing help and a list printout in file form. Both are used as input for the Web*Transactions* program **ifg2fld**. This program then creates the net data specification (.fld file) and the first HTML page (.htm file).

   For more detailed information, consult the manual “Web*Transactions* V3.0 - Connection to *open*UTM Applications via UPIC” [16].
4. The HTML page can optionally be enhanced

   The HTML template that fhs2fld (Web*Transactions* ≤ V2.0) or Web*Lab* generates can be processed with any HTML editor, whose output then serves as input for Web*Transactions*.
5. Install the HTTP server and Web*Transactions*

   Web*Transactions* can
   - either be run on a SINIX or Windows NT computer, in which case it is linked to the UTM application via UPIC and UTM-D (BS2000),
   - or Web*Transactions* is installed in the POSIX subsystem of the BS2000.

   The manual “Web*Transactions* V3.0 - Connection to *open*UTM Applications via UPIC” [16] contains information about both systems.

6. Install the browser

   The following tips apply to the browser settings:

   – Cache settings should be avoided for such interactive applications (the UTM application is an interactive application that generally requires fixed dialog sequences) since the browser may otherwise display old pages.

   – Proxy servers should likewise be avoided since they also cache page contents and may transfer old pages.

8.2.2 Preparations and tips for use

In the BS2000 server application, in Web operation no simultaneous interpreter and object use is possible in the same UTM application: object and interpreter operation must be strictly separated.

UTM version specifics

Using the server application in the BS2000 requires UTM-D Version 3.4. Versions prior to 3.4 are not Web*Transactions*-capable.

For the DRIVE interpreter and the DRIVE compiler, the following steps must be performed:

- Drive interpreter

Item	**Specification in UTM V3.4 or higher**	**Also possible with UTM V4.0 or higher**
First TAC	DRIWEB	<any>
Follow-up TAC	WEBNEXT	SQLNEXT
User ID record (Y/N)	Yes, DRIWEB@@@@@@@@@@	Yes
Program name of the first procedure called	any procedure name	any procedure name
Error procedure	Error screen display or standard error screen	No error procedure since the system does not recognize the WebTA operation.

- Drive compiler

Item	**Specification in UTM V3.4 or higher**	**Also possible with UTM V4.0 or higher**
First TAC	DRIWEB	First object TAC
Follow-up TAC	DRT#W###	Follow-up object TAC
User ID record (Y/N)	No	No
Program name of the first procedure called	DRIWEB (required)	
Error procedure	Error screen display or standard error screen	No error procedure since the system does not recognize the WebTA operation.

Interpreter operation

For the interpreter operation with Web connection, the following files or assignments are required:

– in the DRIVE library: X-element SMO.ERROR
– in the DRIVE library: S-element DRIWEB@@@@@@@@@@ (only needed with First TAC DRIWEB)
– in the FHS library: R-element WEBERR

The Drive library is the library assigned with PAR DYN. The library elements listed ship only in SIPLIB.DRIVE.022.

Object operation

For the object operation with Web connection, the following files or assignments are required:

– in applications for object operation, the R-module SMO#ROR@ must also be linked.
– in the FHS library: R-element WEBERR

The library elements listed ship in SIPLIB.DRIVE.022.

KDCDEF file and application generation

For interpreters and compilers, KDCDEF general-purpose files are shipped for web operation.

– Interpreter: library SYSPRC.DRIVE.022, element DRIKDCDEF.WEB
– Compiler: library SYSPRC.DRIVE-COMP-LZS.022, element DRCKDCDEF.WEB

Note: In WebTA operation, fixed user names (PTERM names) do not make sense, of course. It is recommended to operate with a terminal pool (TPOOL statement).

DRIVE standard HTML template for exceptions

If exceptions occur at runtime in a DRIVE application that are not intercepted by the exception handling (whenever statement), the DRIVE runtime system outputs in web operation the included HTML page WEBERR.htm. The reason for the exception is listed on this page (for example, conversion error in case of faulty data assignments).

This data that ships with the DRIVE correction must be copied to the respective Web*Transactions* directories as follows:

– WEBERR.HTM to ...../*web application name*/`config/improved` or to the directory in which all Web pages must be stored (depending on the Web*Transactions* installation).

– WEBERR.fld to ...../*web application name*/`config`.

Error handling in the BS2000 server application

In the server environment, you must import the intermediate code or, in object operation, the X- or R-element into the DRIVE library:

– SMO.ERROR: X-element in the DRIVE library for interpreter operation
– SMO#ROR@: R-element for object operation

The WEBERR (R-element) format must be imported into the FHS library.

i Errors can only be processed if the First TAC is DRIWEB and the DRIVE application therefore detects WebTA operation.

In case of an error, the above-mentioned elements are then used to return an error message to Web*Transactions* and shut down the server application orderly.

So far, only a USER- or SYSTEMRAISE in the DRIVE server cause Web*Transactions* to abort.

As of UTM Version 4.0, you can access the server application with any TAC. For these TACs (compiler or interpreter), no WebTA-capable error processing is possible since the WebTA operation is not detected. Outputs in LINE mode (dynamic formats), for example and transaction resets (with a prompting message) cause both the client and the server application to abort.

It is therefore recommended to use DRIWEB as First TAC also in UTM V4.0 or higher (both as interpreter and compiler).

8.2.3 Sample BS2000 server application

KDCDEF: Object operation

```
OPTION GEN=ALL
ROOT  DRTWEB
MAX APPLINAME=DRIWEB, APPLIMODE=SECURE, KDCFILE=DRIWEB
FORMSYS ENTRY=KDCFHS,LIB=SYSLIB.DRIVE.022, TYPE=FHS
REM
REM
REM ************************************************************************
REM ***                                                                  **
REM ***                        KDCDEF FRAME FOR DRIVE V2.2               **
REM ***                             NEWSTYLE-VARIANT                     **
REM ***                                FOR WEB                           **
REM ***                    ******* OBJECT OPERATION *********            **
REM ************************************************************************
REM
REM ******** SET MAXIMUM VALUES *******************************
REM
MAX KB=1024,SPAB=32767,NB=32700
MAX TASKS=6,ASYNTASKS=5
MAX KEYVALUE=32,GSSBS=0,LSSBS=200,TRMSGLTH=32700
MAX PGPOOL=(10000,80,95),RECBUF=(30,4096),REQNR=8
MAX TRACEREC=512,TERMWAIT=18000,RESWAIT=60,CONRTIME=10
MAX LOGACKWAIT=600,BRETRYNR=10,VGMSIZE=128
REM
REM ******** CONTROL STATEMENT KDCDEF-LAUF ************
REM
REM ******** DEFINE KDCROOT *******************************
REM
DATABASE TYPE=SESAM,ENTRY=SESSQL,LIB=SYS.MOD.SQL21
DATABASE TYPE=SESAM,ENTRY=SESAM,LIB=SYS.MOD.SQL21
REM
MPOOL <mempool>,SIZE=200,LIB=SYSLNK.DRIVE.022,SHARETAB=<sharetab>
REM
REM ******** DEFINE PROGRAM UNITS *************************
REM
PROGRAM KDCADM,COMP=ILCS
PROGRAM DRIVROOT,COMP=ILCS
PROGRAM EXSTRT,COMP=ILCS
PROGRAM EXSHUT,COMP=ILCS
PROGRAM DRTROOT,COMP=ILCS
PROGRAM DRTVORG,COMP=ILCS
REM
REM ******** DEFINE USER EXITS ****************************
REM
EXIT PROGRAM=EXSTRT,USAGE=START
```

```
EXIT PROGRAM=EXSHUT,USAGE=SHUT
REM
REM ****************** LOAD DRIVE MODULES **********************
REM
MODULE EXTAB,LIB=SYSLNK.DRIVE-COMP-LZS.022,LOAD=STATIC
MODULE EXSTART,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE EXSHUTE,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRIDUM51,LIB=SYSLNK.DRIVE.022,LOAD=STATIC
MODULE DRTMAT20,LIB=SYSLNK.DRIVE-COMP-LZS.022,LOAD=(POOL,<mempool>)
MODULE DRI#ERS@,LIB=SYSLNK.DRIVE-COMP-LZS.022,LOAD=(POOL,<mempool>)
REM
REM ****************** LOAD DRIVE MODULES FOR OBJECT OPERATION ***
REM
MODULE DRIWEB@@,LIB=<useroml>,LOAD=(POOL,<mempool>)
MODULE <userspecif. DRIVE object>,LIB=<useroml>,LOAD=(POOL,<mempool>)
.
.
MODULE <userspecif. DRIVE object>,LIB=<useroml>,LOAD=(POOL,<mempool>)
MODULE SMO#ROR@,LIB=SIPLIB.DRIVE.022,LOAD=(POOL,<mempool>)
REM
REM ******** TRANSACTION CODES FOR DRIVE **********************
REM
DEFAULT TAC PROGRAM=DRIVROOT
REM
TAC DRISQL   ,TYPE=D,STATUS=ON,CALL=FIRST,EXIT=DRTVORG
TAC DRISQLF  ,TYPE=D,STATUS=ON,CALL=NEXT
TAC SQLNEXT  ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC SQLENTER ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRTVORG
TAC SQLLIST  ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRTVORG
REM
REM ******** TRANSACTION CODES FOR COMPILER *******************
REM
DEFAULT TAC PROGRAM=DRTROOT
REM
TAC DRTXSCXX ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRTXSCAX ,TYPE=A,STATUS=ON,CALL=FIRST,TIME=300000,EXIT=DRTVORG
TAC DRT#I### ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRT#O### ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRTXCRXX ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
TAC DRTXCLXX ,TYPE=D,STATUS=ON,CALL=NEXT,TIME=300000
REM
REM ******** TRANSACTION CODES FOR OBJECTS ********************
REM
TAC DRIWEB ,CALL=FIRST,TYPE=D,EXIT=DRTVORG,TIME=300000
TAC <userspecif. Tacname>,CALL=NEXT,TYPE=D,TIME=300000
.
TAC <userspecif. Tacname>,CALL=NEXT,TYPE=D,TIME=300000
TAC SMO#ROR,CALL=NEXT,TYPE=D,TIME=300000
```

```
REM
REM ******** SYNCHRONOUS ADMINISTRATION ************************** *
REM
DEFAULT TAC PROGRAM=KDCADM
TAC KDCTAC,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCLTERM,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCPTERM,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCSWTCH,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCUSER,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCSEND,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCAPPL,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCDIAG,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCLOG,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCINF,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCHELP,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCSHUT,TYPE=D,STATUS=ON,CALL=BOTH,ADMIN=Y,DBKEY=UTM
TAC KDCTCL,TYPE=D,STATUS=ON,CALL=BOTH,DBKEY=UTM,ADMIN=Y
REM
REM ******** ASYNCHRONOUS ADMINISTRATION ************************
REM
TAC KDCTACA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCLTRMA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCPTRMA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCSWCHA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCUSERA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCSENDA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCAPPLA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCDIAGA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCLOGA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCINFA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCHELPA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCSHUTA,TYPE=A,STATUS=ON,CALL=FIRST,ADMIN=Y,DBKEY=UTM
TAC KDCTCLA,TYPE=A,STATUS=ON,CALL=FIRST,DBKEY=UTM,ADMIN=Y
REM
REM ******** FUNCTION KEY ASSIGNMENT **********************
REM
SFUNC K1,RET=2OZ                          "BREAK KEY"
SFUNC F1,CMD=KDCOFF
SFUNC K2,CMD=KDCOUT
REM
REM ******** CONNECTION TO SINIX-COMPUTER                 **
REM ******** PARTNER NAME                                 **
BCAMAPPL  <bcamname>,T-PROT=ISO
REM
TPOOL  BCAMAPPL=<bcamname>,    -
              LTERM=LTRM,NUMBER=50,PRONAM=<rechnername>,PTYPE=UPIC-R
REM
END
```

Related publications

[1] **DRIVE/WINDOWS V2.1** (BS2000)
Programming System
User Guide

Target group
Application programmers
Contents
Introduction to the programming system DRIVE/WINDOWS and Explanation of the functions available in interactive mode. Description of the DRIVE/WINDOWS installation, generation and administration.

[2] **DRIVE/WINDOWS V2.1** (BS2000)
Programming Language
Reference Guide

Target group
Application programmers
Contents
Description of program creation including alpha screen forms, as well as the use fo DRIVE list forms and the report generator.

[3] **DRIVE/WINDOWS V2.1** (BS2000)
Directory of DRIVE Statements
Reference Manual

Target group
Applications programmers
Contents
Syntax and range of functions of all DRIVE statements. DRIVE messages and keywords.

[4] **DRIVE/WINDOWS V2.1** (BS2000)
Directory of DRIVE SQL Statements for SESAM/SQL 2
Reference Manual

Target group
Application programmers

Contents
A concise description of the syntax and scope of functions of all the DRIVE SQL statements for SESAM V2.x.

[5] **DRIVE/WINDOWS-COMP V2.1** (BS2000)
Compiler
User Guide

Contents
The differences the compiler and the interpreter, and the compilation process. Generating and starting TIAM and UTM applications with compiled DRIVE objects, with special consideration of mixed version operation.

[6] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
SQL Reference Manual Part 1: SQL Statements
User Guide

Target group
The manual is intended for all users who wish to process an SESAM/SQL database by means of SESAM/SQL statements.
Contents
The manual describes how to embed SQL statements in COBOL, and the SQL language constructs. The entire set of SQL statements is listed in an alphabetical directory.

[7] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
SQL Reference Manual Part 2: Utilities
User Guide

Target group
The manual is intended for all users responsible for SESAM/SQL database administration.
Contents
An alphabetical directory of all utility statements, i.e. statements in SQL syntax implementing the SESAM/SQL utility functions.

[8] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
Core Manual
User Guide

Target group
The manual is intended for all users and to anyone seeking information on SESAM/SQL.
Contents
The manual gives an overview of the database system. It describes the basic concepts. It is the foundation for understanding the other SESAM/SQL manuals.

[9] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
Utility Monitor
User Guide

Target group
The manual is intended for SESAM/SQL-Server database and system administrators.
Contents
The manual describes the utility monitor. The utility monitor can be used to administer the database and the system. One aspect covered is its interactive menu interface.

[10] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
Messages
User Guide

Target group
All users of SESAM/SQL.
Contents
All SESAM/SQL messages, sorted by message number.

[11] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
CALL DML Applications
User Guide

Target group
SESAM application programmers
Contents
– CALL DML statements for processing SESAM databases using application programs
– Transaction mode with UTM and DCAM
– Utility routines SEDI61 and SEDI63 for data retrieval and direct updating
– Notes on using both CALL DML and SQL modes

[12] **SESAM/SQL-Server V2.2A** (BS2000/OSD)
Performance
User Guide

Target group
Experienced users of SESAM/SQL.
Contents
The manual covers how to recognize bottlenecks in the behavior of SESAM/SQL and how to remedy this behavior.

[13] ***open*UTM** (BS2000/OSD)
Generating and Handling Applications
User Guide

Target group
This manual is intended for application planners, technical programmers, administrators and users of UTM applications.

[14] **FHS** (TRANSDATA)
User Guide

Target group
Programmers
Contents
Program interfaces of FHS for TIAM, DCAM and UTM applications. Generation, application and management of formats.

[15] **LMS** (BS2000)
ISP Format
Reference Manual

Target group
BS2000 users
Contents
Description of the LMS statements in ISP format for creating and managing PLAM libraries and the members these contain.
Frequent applications are illustrated by means of examples.

[16] **Web*Transactions***
Connection to *open*UTM Applications via UPIC
User Guide

Target group
Anyone who wishes to use Web*Transactions* to connect UTM dialog applications to the Web.
Contents
The manual describes all the steps required for connecting UTM dialog applications to the Web. It supplements the intoductory manual “Concepts and Functions“ and the Reference Manual “Template Language“ by providing all the information relating to UTM.

[17] **BS2000/OSD-BC**
System Installation
User Guide

Target group
BS2000/OSD system administration
Contents
This manual describes
– the generation of the hardware and software configuration with UGEN
– the following installation services:
  – disk organization with MPVS
  – program system SIR
  – volume installation with SIR
  – configuration update (CONFUPD)
  – utility routine IOCFCOPY

Ordering manuals

Please apply to your local office for ordering the manuals.

Index

E

Contents

DRIVE V2.2

Target group

Programmers und administrators of DRIVE applications in BS2000.

Contents

This manual describes the new and modified functions of DRIVE V2.2:
– Information on DRIVE implementing
– New and modified DRIVE SQL statements
– Modified DRIVE statements
– Information on DRIVE programming
– Database support
– Y2K support for old-style DRIVE
– DRIVE web interface

Edition: January 2000

File: drive_22.pdf

Fujitsu Siemens computers GmbH
User Documentation
81730 Munich
Germany

Fax: (0 89) 6 36-4 04 43

e-mail: DOCetc@mchp.siemens.de
http://manuals.mchp.siemens.de

Comments
Suggestions
Corrections

Submitted by

Comments on DRIVE V2.2
Supplement

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.