1 Preface

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

1.1 Brief product description

DRIVE/WINDOWS is a fourth generation programming language (4GL) for the development of commercial client-server applications. It is the 4GL used to access files or the BS2000 database systems SESAM/SQL V1, SESAM/SQL V2 and UDS. DRIVE applications can be distributed across different computer types in accordance with various client-server architectures since DRIVE/WINDOWS is available for BS2000, SINIX and MS-Windows platforms and provides optimum support for client-server connections.

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/WINDOWS automatically provides system-specific interfaces to components, thus relieving the programmer of this task.

DRIVE/WINDOWS 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/WINDOWS-COMP compiler.

DRIVE server applications running under SINIX allow you to access INFORMIX files and databases and can also be used as part of a distributed application using DRIVE applications under BS2000 or MS-Windows. Here again, as in BS2000, there is an integrated report function and a compiler designed to ensure that the created server applications run at top performance.

In addition to facilities for creating applications, DRIVE/WINDOWS under MS-Windows provides a range of convenient tools which are fully integrated into the development process. case/4/0 supports application design and is complemented by DRIVE/DESIGNER which ensures a seamless transition to the coding phase and guarantees that the DRIVE source programs are generated and installed correctly on the basis of the design results. The DRIVE/WINDOWS software production environment thus provides a convenient, graphical, menu-driven user environment for program creation, testing and application control.

1.2 Target group

This manual is aimed at programmers who develop DRIVE applications or components of client-server applications using DRIVE/WINDOWS 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 UDS database system

– the SESAM database system

– the UTM transaction monitor

– the FHS Format Handling System for creating screen forms

– the client operating system (MS-Windows or SINIX)

1.3 Organization of manual suite

DRIVE/WINDOWS is described in three manuals:

– The "Programming System" [1] manual provides a general overview of the DRIVE/WINDOWS system and explains the functions which are used to prepare, save, test and execute DRIVE programs. It also contains information required by the system administrator to prepare DRIVE/WINDOWS for use as well as information on configuring client/server architectures.

– The "Programming Language" [2] manual describes the rules governing a DRIVE program. It deals with the programming logic, the creation of screen and list forms, report design using the report generator and discusses client-server architectures.

– The "System Directory" [3] contains all the DRIVE statements in alphabetical order together with their syntax and a description of their functional scope. The SQL statements are described in separate manuals (see below).

  The statements are arranged in three sections: DRIVE statements, report statements for the generation of lists, forms and reports, and the complex statement elements known as “metavariables”.
  The system directory also includes an introduction to the syntax of DRIVE statements and a list of all DRIVE/WINDOWS messages and keywords.

Directories containing the DRIVE-SQL statements for the various database systems are also available:

– "Directory of DRIVE-SQL statements for SESAM V1" [4]

– "Directory of DRIVE-SQL statements for SESAM V2" [5]

– "Directory of DRIVE-SQL statements for UDS" [6]

DRIVE/WINDOWS also offers the full functionality of DRIVE V5.1 in so-called “old style” and mixed mode operation.
For a description of the old style functions, refer to the DRIVE V5.1 User Guide [14] and System Directory [15]. Techniques for integrating old style programs into new DRIVE applications are described in the Programming Language" [2] manual. The "Programming System" [1] manual describes how you generate DRIVE/WINDOWS for old style and mixed mode operation.

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 filename SYSRME.*product*.*version*.*language*. Please ask your system administrator 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 3.0A:

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

1.5 Changes compared to the version of December 1993 (DRIVE/WINDOWS V1.1)

Components

– DRIVE/WINDOWS supports SESAM V2 and therefore complies with the SQL2 language standard.

– SAM and ISAM files can be edited using DRIVE programs.

– Distributed DRIVE applications allow you to connect to client graphic user interfaces (MS-Windows and SINIX).

– Old-style program can be integrated into new DRIVE applications using the CALL program call. Parameters can be returned to the calling program.

– In old-style mode it is now possible to access SESAM V2, LEASY and DMS as well as SESAM V1.

– DRIVE programs containing SQL statements for SESAM V2 can now only be compiled if there are no open transactions.

– The interfaces to the products TOM-REF and QUERY are no longer supported.

Data types

– DRIVE/WINDOWS supports the TIME(3) and TIMESTAMP(3) data types.

– The VARCHAR data type occupies one byte more than previously. It is no longer possible to redefine items as VARCHAR variables or manipulate the length field.

Functions

– The HELP statement is no longer present. Instead the DRIVE system directory is provided as a file (softbook).

– You can specify background patterns (bases) for reports both for pages and for individual lines.

– The call sequence for CALL and DO has changed: whereas DRIVE/WINDOWS previously searched for the intermediate code before looking for the source, it now always looks for the member with the most recent date.

– The COMPILE statement does not generate an error list. Error messages are inserted in the source section of the compiler listing.

– The PARAMETER DYNAMIC LIBRARY statement no longer automatically creates a PLAM library. Instead, it can only assign an existing PLAM library.

- User-specific data types can be passed to called programs as transfer parameters. The DECLARE TYPE statement may come in front of the PROCEDURE statement.
- The COMMIT WORK and ROLLBACK WORK statements are permitted without the OPTION DBSYSTEM= statement and with the OPTION DBSYSTEM=OFF statement. If no database transaction is open when the statements are executed, they refer exclusively to UTM transactions.
- You can use the OPTION SCREENCHECK compiler option to specify whether or not DRIVE/WINDOWS is to evaluate CHECK clauses in addressing aids.
- The DRIVE system variable &SQL_STATE contains the SQLSTATE of SESAM V2.
- The PERMIT statement is ineffective for SESAM V2. It simply sets the SQLSTATE.
- CURRENT TIMESTAMP outputs the current timestamp.
- You can specify fractions of a second (FRACTION) as a unit for intervals or in time specifications.
- Date expressions can be linked to a result with the TIMESTAMP(3) data type.
- The LENGTH function returns the last non-space character in a string.
- The UPPERSTRING and LOWERSTRING functions convert characters in accordance with the country-specific settings, while the TRSTRING converts them on the basis of user-specific definitions.
- When calculating intervals and times (TIME), DRIVE/WINDOWS does not calculate across changes of day. Instead it reports negative hours.
- In the absence of any declaration to the contrary, null values are represented on screen by the special character @.
- PASCAL program calls are not supported.

Keywords

- DRIVE/WINDOWS uses new keywords. The Appendix to the DRIVE System Directory [3] contains a list of all keywords.

Compatibility

- DRIVE programs whose code members were created using an earlier version of DRIVE/WINDOWS must be recompiled before being executable.
- In the case of DRIVE programs which access SESAM databases, observe the notes on migration in the "Directory of DRIVE-SQL statements for SESAM V2" [5].

1.6 Notational conventions

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

`type-written text`

is used for fixed names (e.g. operating system commands, filenames) and error messages in the body 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 chapter “Statement format and syntax” on page 7.

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

2 Statement format and syntax

2.1 Format

DRIVE and SQL statements consist of the following elements:

– keywords
– names
– literals
– metavariables
– delimiters
– comments

Example

```
CYCLE cursor-name INTO variable WHILE char-name='literal' /*Loop*/
```

Keywords:	CYCLE, INTO, WHILE
Names:	cursor-name, char-name
Literals:	literal
Metavariables:	variable
Delimiters:	blank, equals sign (=)
Comments:	Loop

Keywords

Keywords are words that have to be specified as shown in the manual. A list of all the keywords used in DRIVE/WINDOWS together with their abbreviations can be found in the appendix.

Names

Names identify variable values that the user must replace with current values when entering a statement.

Names can contain alphabetics, numerics and special characters provided that no special restrictions are described.

Names containing alphabetics, numerics and underscore characters (_), do not need to be specially marked. Names which additionally contain further special characters must be enclosed in double quotes (").

Literals

Literals are constants that are passed to the language processor in the form specified.Numeric literals are specified directly and hexadecimal literals are specified with X'*literal*'.
Alphanumeric literals must be enclosed in single quotes.
In the case of date and time literals, you must specify whether the literal contains a date, a time or a timestamp. In the case of interval literals, you must specify a unit for the time range.

Any single quotes (') contained in literals must be cancelled by a second single quote.

Example

The literal "That's it" is expressed as follows:

```
'That''s it'
```

Metavariables

Metavariables are complex parts of statements that have been omitted from a statement to facilitate comprehension. They are covered in a separate chapter (see chapter “DRIVE metavariables” on page 271).

Delimiters

Delimiters must be specified between keywords, names, literals and metavariables in order to uniquely identify them. The following can be used as delimiters:

– blank
– comma (,)
– concatenation operator ||
– all comparison operators = < > <= >= <>
– all arithmetic operators + - * / % **

A comment or an end-of-line character also acts as a delimiter outside strings enclosed in single quotes (') or double quotes (").

Comments

A comment is introduced by the character string /* and terminated by the character string */. Any text may be written between these characters, even extending over more than one line.

The character strings /* and */ do not indicate comments when they are enclosed in single quotes (') or double quotes (").

2.2 Syntax

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`

( )	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 }`

Formal representation	Meaning	Example
...	An ellipsis indicates that the variable which immediately precedes the ellipsis can be repeated several times.	`AT line ...`
	If the ellipsis is preceded by a unit enclosed in brackets, the entire unit must be entered.	`USING { [ RETURN ] [ level ] var-name data-type }, ...`
	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.	`(attribute, ...)`

3 DRIVE statements

ACQUIRE
Request memory area

This application is valid

– in the UTM start procedure as DRIVE start parameter

ACQUIRE is used to request dimensionable memory areas (= cache) for DRIVE UTM applications. These cache areas are used as buffers for internal, user-specific DRIVE information and result in improved performance.

```
ACQUIRE MEMORY mlength USER n
```

---

mlength

Size of a memory area in the cache, in Kbytes.

Value range: 0 < *mlength* < 2147483647.
mlength must be an integer.

The size of the cache in bytes is determined as:
n * *mlength* * 1024

This size is then rounded up to a whole multiple of:

– 64 Kbytes, if DRIVE/WINDOWS is loaded in the lower address space (< 16 Mbytes)
– 1 Mbyte, if DRIVE/WINDOWS is loaded in the upper address space (> 16 Mbytes)

If the same cache is to be accessed in mixed operation with DRIVE V5.1, the total sizes must be the same.

n — Number of DRIVE UTM users whose internal, user-specific DRIVE information is to be simultaneously buffered in the common memory pool (class 6 memory) between UTM dialog steps.
Value range: $0 < n < 32767$.

Example

100 Kbytes of memory space is to be allocated to each of 15 DRIVE UTM users for intermediate storage of user-specific data.

```
ACQUIRE MEMORY 100 USER 15
```

ADD BOX
Output dialog box

This statement is valid

– in UTM mode but not in ansynchronous UTM mode and not in the receiving partner environment in DTP mode
– in program mode

ADD BOX outputs a dialog box which you have previously created in IFG (see IFG [28]).

Any screen forms which have already been output (partial forms and dialog boxes) continue to be displayed but are overlaid by the dialog box and are locked, i.e. the user cannot make any input to these screen forms.

The last dialog box to be output is the current dialog box. Users can only input to the current dialog box.

You may only enter the ADD BOX statement if an FHS-DE partial form has already been ouput using the DISPLAY screenform statement as otherwise DRIVE/WINDOWS aborts the program.

```
ADD BOX dialogbox
        [ POSITION ( line1 , column1 ) ] [ TO field1 ]
        [ CURSOR { POSITION ( line2 , column2 ) | TO field2 }
        [ MESSAGE key[ POSITION ( line3 , column3 ) | TO field3 ]
```

---

dialogbox — Name of the FHS-DE form (max. 7 characters).

The form must have been created in IFG and it must be intended for display in a message box.

The form must be defined in the declaration section of the program using the DECLARE SCREEN statement.

POSITION — Specifies the position of the dialog box.

The position is specified using a starting or reference point
The starting point is the first character (top left) of the dialog box.
The reference point is the first character of *field1* if *field1* is specified or, otherwise, the top left-hand corner of the dialog box/FHS-DE partial form which is located under the current dialog box.

	If you do not specify POSITION or if you enter the value (0,0), DRIVE/WINDOWS attempts to position the dialog box with the default offset (+2,+2) to the reference point. If this is not possible, the dialog box is moved so that it fits on the screen. If you specify the POSITION but there is not enough space for the dialog box at the defined position, UTM aborts the operation with PEND ER.
line1	Line spacing between the reference point and the starting point of the dialog box. *line1* must be a whole number.
column1	Column spacing between the reference point and the starting point of the dialog box. *column1* must be a whole number.
field1	Field in the last FHS-DE form to be output (partial form and dialog box). *field1* must be a simple component of the associated screen variable. If *variable* is not a component of the screen variable of the last screen form to have been output, UTM aborts the conversation with PEND ER. The first 8 characters of the field names in the last screen form to have been output must differ in order to make the unambiguous assignment of screen variable components possible.
CURSOR	The cursor is set to a specific position in the dialog box. You may not specify CURSOR unless the global attribute ““dialog cursor position” was set for *dialogbox* in IFG (see IFG [28]).
POSITION	Specifies the absolute position (line/column) of the cursor.
line2	Line ($1 \leq$ *line2* $\leq$ number of screen lines). *line2* must be a whole number.
column2	Column ($1 \leq$ *column2* $\leq$ number of screen columns). *column2* must be a whole number.
TO	The cursor is set to the first character of field *field2*. In the case of lists, the cursor is set to the first column and first line of the list area.
field2	Field in the dialog box which is to be output. *field2* must be a component of the screen variable for *dialogbox*. The first 8 characters of the field names in the current dialog box must differ to make the unambiguous assignment of screen variable components possible.

MESSAGE	This outputs the FHS-DE message with the message key *key* which you created with IFG along with the dialog box. Depending on the IFG specification, output either takes place in a message box or in a message area in the dialog box. You may not specify MESSAGE unless the global attribute “Message Identifier” was set for *dialogbox* in IFG (see IFG [28]).
key	Message key of the FHS-DE message. You can specify *key* either as a variable (see the metavariable *variable*) or as an alphanumeric literal (see *char-literal* in the metavariable *literal*). *key* must be specified in the form AAAAnnn, where A is a letter (A-Z) and n is a digit (0-9). AAAA may not have the value IDHS.
POSITION	Specifies the absolute position of the message box. The message box is positioned with an additional offset (+2,+2) to *line3*, *column3*. POSITION is only evaluated if the IFG specification stipulates that the message is to be output in a message box. If a message is intended for output in a message box and you have not specified either POSITION or TO then the message is output in the middle of the screen. If a message box which has been positioned using MESSAGE POSITION covers a cursor which has been set with CURSOR POSITION then MESSAGE POSITION is ignored.
line3	Line ($1 \leq line3 \leq$ screen lines). *line3* must be a whole number.
column3	Column ($1 \leq column3 \leq$ screen lines). *column3* must be a whole number.
TO	Specifies that the message box is to positioned with the default offset (+2,+2) to *field3*. TO is only evaluated if the IFG specification stipulates that the message is to be output in a message box. If a message is intended for output in a message box and you have not specified either TO or POSITION then the message is output in the middle of the screen.
field3	Field in the dialog box which is to be output. *field3* must be a component of the screen variable for *dialogbox*. The first 8 characters of the field names in the current dialog box must differ to make the unambiguous assignment of screen variable components possible.

AT
Declare testpoint and operation

This application is valid

– in TIAM mode
– in debugging mode

The AT statement can be used to define testpoints and operations for a program in debugging mode.

DRIVE/WINDOWS always assigns each user-defined testpoint to a statement of the DRIVE program.

The AT statement refers either to the program which is currently running in debugging mode or, in the *library* or *member-name* specification, to a subprogram from the DRIVE library.

If no operations are specified, the program halts at the testpoint (= implicit operation [STOP]). The testpoint becomes the breakpoint. You can specify one of the following debugging statements at breakpoints:

– AT
– BREAK
– BREAK DEBUG
– CONTINUE
– DISPLAY FORM
– DISPLAY LIST
– REMOVE
– SET
– TRACE

If the CONTINUE statement is entered, debugging is resumed.

Multiple operations may be specified for the same testpoint by using several AT statements. When a testpoint is reached, the operations specified are processed in the following order: first, all DISPLAY and SET statements are processed in the order in which they were entered. Then, either the CONTINUE or the TRACE statement is processed, or the program stops (= implicit [STOP]), depending on which statement was declared last. The COUNT statement is not executed until the program statement associated with the testpoint was completed without errors.

The operations CONTINUE, TRACE and [STOP] override each other.

Operations that remain valid for a testpoint are processed whenever the statement to which the testpoint is assigned is executed.

```
AT { [ library(member-name) | member-name ] { line ... | line1 - line2 | ALL } |
     * }
   [ { CONTINUE | COUNT | DISPLAY FORM | DISPLAY LIST | SET | TRACE } ... ]
```

library	Specifies the DRIVE library (max. 54 characters) from which the program is read. *library* may also be the file link name of the DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name. If the DRIVE library has been predefined with the PARAMETER DYNAMIC LIBRARY statement, *library* need not be specified.
member-name	Name of the member (max. 31 characters) which contains the program. DRIVE/WINDOWS searches for the last member to have been processed, irrespective of whether this contains a source (S-member) or an intermediate code (X-member). If you do not enter a *library* specification then the library specified in PARAMETER DYNAMIC LIBRARY is used. If you do not enter a *member-name* specification then the testpoint is set in the program which is running in debugging mode.
*	The specification * designates the last testpoint entered.
line	Refers to a line number in the interpreter listing. An executable statement for which a testpoint is set begins in this line. Exception: no testpoint can be defined for the PROCEDURE statement of the program called with DEBUG. Multiple line numbers may be specified. If more than one statement begins in a given *line*, a testpoint is declared for each such statement.

line1-line2	Refers to line numbers in the interpreter listing. A testpoint is declared for every program statement that begins in a line of this section. *line1* must be smaller than *line2*.
ALL	A testpoint is declared for all executable program statements.
CONTINUE	When the testpoint has been reached and all statements included there have been executed, the program running in debugging mode is continued.
COUNT	A counter is declared for each program statement which corresponds to a testpoint . This counter is initialized with the value 0 and is incremented each time the statement is successfully executed. Counter values are output in the debugging list at the end of the debugging session. You can use the REMOVE statement to delete counters.
DISPLAY FORM	See the DISPLAY FORM statement. The identification of variables using RETURN is ignored.
DISPLAY LIST	See the DISPLAY LIST statement.
SET	See the SET statement.
TRACE	See the TRACE statement.

AT statements refer to the current program or subprogram. They are ineffective for any successor programs which are called with DO.

Example

The "test" program is tested in debugging mode.

All the executable program statements are counted. The statements in lines 15, 17 and 20 to 55 of the interpreter listing for the main program, "test", are testpoints. The variable &var1 is first set to 1, output at the printer and then the screen. The program run is then continued.

At the statement in line 33 of the interpreter listing for the"test2" subprogram which is located in the predefined library, the variable &subvar1 is set to 2 and output at a printer.

As of line 99 of the main program, "test", program tracing is activated and the debugging run continues.

```
DEBUG test;          /* DRIVE halts at the first executable statement*/
                     /* in the program body                          */
AT ALL COUNT;
AT 15 SET &var1 = 1
AT 17 DISPLAY LIST &var1
AT 20 - 55 DISPLAY FORM &var1
AT * CONTINUE
AT test2 33 SET &subvar1 = 2
AT test2 33 DISPLAY LIST &subvar1
AT 99 TRACE
CONTINUE             /* Only now does the debugging run continue   */
...
```

BREAK
Clear screen or abort logical program unit

This application is valid

– in TIAM and UTM applications
– in interactive, program and debugging mode with distinct functions (v.i.)

BREAK has three different functions depending on the active mode:

– In interactive mode the function which is currently active is aborted and the screen is cleared. If you specify BREAK as part of a nested program run, all the programs are aborted and the system switches to interactive mode.
– In debugging mode, following BREAK, control is passed to the final breakpoint of the program being tested (after the END PROCEDURE statement).

  If a counter was declared (with the AT ... COUNT statement), the result list is displayed after BREAK DEBUG. The system then exits debugging mode and switches to interactive mode.
– In program mode, BREAK aborts logical program units (a loop, a branch, a program, a program hierarchy, an internal subprogram).

There are two other ways to abort a program:

– by assigning the BREAK function to a K key (the default is the K1 key), or
– by entering the BS2000 command `SEND-MESSAGE TO=PROGRAM,MESSAGE=BREAK` (This method is only possible in TIAM mode).

`BREAK [ CYCLE | DEBUG | PROCEDURE | SUBPROCEDURE ]`

CYCLE

A loop is aborted and the program continues with the statement following the corresponding END CYCLE.

An implicit CLOSE cursor-name is executed if BREAK CYCLE is specified within a cursor loop (CYCLE *cursor-name* ... through END CYCLE).

BREAK CYCLE must appear within cycle boundaries (CYCLE through END CYCLE). A BREAK CYCLE within an internal subprogram cannot abort a loop in the calling program.

DEBUG	The BREAK DEBUG statement is only permitted in debugging mode. Following this statement, the result list is output, debugging mode is aborted and the system switches to interactive mode.
PROCEDURE	A program is aborted. If the program was called with CALL, control returns to the calling program and processing continues with the statement following the CALL. If the program was called with DO, DRIVE/WINDOWS switches to interactive mode. All necessary resources are freed and all open transactions are reset. It is not possible to restart the program. If BREAK PROCEDURE is specified within a nested program sequence, only the current program level is aborted. A BREAK PROCEDURE within an internal subprogram has the same effect as END PROCEDURE. The same rules apply (see the END statement). If BREAK PROCEDURE is specified for a program called with DO, DRIVE/WINDOWS closes all files opened with the OPEN FILE statement. If BREAK PROCEDURE is specified for a program called with CALL, files opened with the OPEN FILE statement remain open. BREAK PROCEDURE may not occur within a DISPATCH block.
SUBPROCEDURE	An internal subprogram is aborted and processing continues with the statement following the CALL in the calling program. The calling program may likewise be an internal subprogram. BREAK SUBPROCEDURE must be specified as a static statement within subprogram boundaries (SUBPROCEDURE through END SUBPROCEDURE).

Rules for database access

The following rules apply to BREAK DEBUG:

– If any transaction is still open, DRIVE/WINDOWS resets it and issues message DRI0101.

– If any temporary SQL objects which have been defined in program mode (program cursor or temporary views) are present, they are deleted by DRIVE/WINDOWS. DRIVE/WINDOWS issues the message DRI0150 if it is unable to delete an SQL object.

– If dynamic, temporary views are present when SESAM V2.x is accessed, DRIVE/WINDOWS deletes them and issues message DRI0488.

– When accessing SESAM V2.x, DRIVE/WINDOWS issues a SET SESSION, SET CATALOG and SET SCHEMA statement. The operands for these statements are defined in the previous PARAMETER DYNAMIC AUTHORIZATION, PARAMETER DYNAMIC CATALOG and PARAMETER DYNAMIC SCHEMA statements respectively.

CALL
Call subprograms

This application is valid

– in TIAM and UTM mode
– in program mode

CALL is used within a program to call both internal and external subprograms.

The calling program is interrupted, and the subprogram is executed. Subsequently, control returns to the calling program, and processing continues with the statement that immediately follows the CALL.

An internal subprogram is a named sequence of DRIVE statements that may be called within a DRIVE program as often as desired. Other internal subprograms may only be called within an internal subprogram if they have been previously defined.

An external subprogram is an independent program which can be called by other programs as often as required. External subprograms can be written in DRIVE/WINDOWS (new style or old style) or in another programming language (e.g. COBOL or C). They may also take the form of UTM program units which are called with the transaction code.

Irrespective of the programming language used, external subprograms can be located locally or on a remote system. Depending on the distribution information specified in PARAMETER DISTRIBUTION, DRIVE/WINDOWS searches for a program in the local system or remote system if OPTION DISTRIBUTION is set to ON. CALL statements which call external subprograms in remote systems are referred to as remote CALL statements.

DRIVE programs in a remote system must not contain the following statements:

– CALL ... TAC (if the UTM subprogram is to be executed on the local system)
– COMMIT WORK WITH DISPLAY
– COMMIT WORK WITH SEND MESSAGE
– DISPLAY
– DO
– FILL
– ROLLBACK WORK WITH RESET
– SEND MESSAGE
– STOP WITH DISPLAY
– STOP WITH char-expression

External DRIVE subprograms can be available as intermediate code or as source code. DRIVE/WINDOWS searches for the most recently processed program with the specified name, irrespective of whether it is available as source code or intermediate code. If intermediate code is found, no syntax or semantic checks are performed (see DO statement).

If CALL ... is specified without a library, DRIVE/WINDOWS searches for an internal subprogram with the specified name in the current program (CALL *subprogram-name*) when the source is compiled.
If there is no internal subprogram with the specified name, then at runtime DRIVE/WINDOWS searches for a member in the library specified in the PARAMETER DYNAMIC LIBRARY statement (CALL *member-name*). If no member having this name exists, the program is aborted.

Within an EXECUTE statement, the CALL ... statement without a library specification calls an external DRIVE subprogram.

Recursive program calls are not permitted and cause the program to abort.

```
CALL { subprog-name |
       library(member-name) |
       member-name |
       [ COBOL | C | PASCAL ] { MODULE module-name | TAC tac-name }

     [ USING { [ RETURN ]  { expression | NULL }
               [ INIT expression [ NOCHECK ] ] [ INDICATOR ] }, ... ] }
```

---

subprog-name
: Name of an internal subprogram (max. 31 characters). The subprogram must be defined in the current calling program with SUBPROCEDURE *subprog-name*.

  You may not specify a USING clause in CALL *subprog-name*.

library
: Specifies the DRIVE library (max. 54 characters) from which the subprogram is read. *library* can also be the file link name of the DRIVE library (in accordance with BS2000 conventions).

  DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name.

  If the DRIVE library has been preset for the (local or remote) system using the PARAMETER DYNAMIC LIBRARY statement, you may omit the *library* specification.

member-name	Name of the member (max. 31 characters) which contains the subprogram. DRIVE/WINDOWS searches for the last member to have been processed, irrespective of whether this contains a source (S-member), an intermediate code (X-member) or an object code (R-member). *member-name* can be an old style program which was created using DRIVE V5.1. Please refer to the DRIVE Programming Language [2] manual for notes on calling old style programs. The system only checks whether the PARAMETER DISTRIBUTION statement was used to define distribution information if the compiler option OPTION DISTRIBUTION=ON was defined. A local or remote program is called in accordance with the distribution information. If no distribution information has been defined with the PARAMETER DISTRIBUTION statement, the system searches for the program locally. If you do not make a *library* specification, then the system uses the library which has been preset for the (local or remote) system using PARAMETER DYNAMIC LIBRARY.
COBOL	Default Language option for calling a subprogram written in COBOL or a UTM program unit.
C	Language option for calling a subprogram written in C or a UTM program unit.
module-name	Name of the subprogram (max. 8 characters) which is called (applies only to external subprograms written in other programming languages). The external subprogram be present in the library with the file link name USEROML. This library is automatically used if no other library has been specified with the PARAMETER DYNAMIC LIBRARY statement. The external subprogram must not contain any database statements (SESAM/UDS). Parameters are exchanged via a parameter transfer area (see the DRIVE Programming Language manual [2]).

tac-name

Name of the transaction code (max. 8 characters) of a user-specific UTM program unit. The UTM program unit may form part of the local or of a remote UTM application. The UTM statement can run under SINIX or BS2000.

tac-name must not have the prefix `dri`, `drt` or `drc`. These names are reserved for DRIVE/WINDOWS.

When you call a user-specific UTM program unit, you must specify the programming language in which it was written.

The system does not check to determine whether distribution information has been defined using the PARAMETER DISTRIBUTION statement unless you have specified the OPTION DISTRIBUTION=ON compiler option. A remote UTM program unit is called as specified in the distribution information. If you do not specify any distribution information with PARAMETER DISTRIBUTION then the system searches for the UTM program unit locally.

In TIAM mode, CALL TAC is ignored.

USING

Enables parameters to be passed from the calling program to the called program. The parameters must be compatible (see the DRIVE Programming Language manual [2]). If this is not the case, in debugging mode, the user is prompted for parameters when an external DRIVE program is called (CALL [(library)]member-name) (see the DRIVE Programming Language manual [2]).

If CALL MODULE ... USING or CALL TAC ... USING is specified, the total length of the data values passed must not exceed 31 Kb. There is no restriction when calling external DRIVE subprograms in local systems.

If the external subprogram is a DRIVE program, it must have been defined with PROCEDURE...USING...; otherwise, the program is aborted.

If the external subprogram is a UTM program unit, the passed parameters are placed in the message area. The UTM statement MGET requests the passed data from UTM for the program unit (see the DRIVE Programming Language [2] manual).

The rules described in the DRIVE Programming Language [2] manual apply to external subprograms written in other programming languages.

USING is not permitted when calling an internal subprogram (CALL *subprog-name*).

RETURN	Identifies the parameters to be returned to the calling program. In the case of external subprograms in DRIVE, the called program must have been defined with PROCEDURE...RETURN... . If no RETURN clause was specified with PROCEDURE, the program is aborted. In the case of external subprograms in other programming languages, the values stored by the subprogram in the parameter transfer area are passed to the RETURN parameters. If the external subprogram is a UTM program unit, the data values placed in the message area by that program unit are transferred to the RETURN parameters (see the section on distributed transaction processing in the DRIVE Programming Language [2] manual). A variable identified with RETURN may only be specified once in the USING clause.
expression	Specifies the parameters to be passed (send fields). Variables (including structured variables), vectors, matrices, aggregates, literals and arithmetic expressions can be passed to external subprograms in DRIVE/WINDOWS. Variables (including structured variables), vectors, matrices, literals and arithmetic expressions can be passed to external subprograms in the remote system in DRIVE/WINDOWS. Variables (including structured variables), vectors and matrices can be passed to external subprograms written in other programming languages (CALL MODULE) as well as to transaction codes (CALL TAC). *expression* must be a variable (see the metavariable *variable*). The INDICATOR clause must be specified if *expression* has the value NULL with CALL MODULE or CALL TAC. If this is not the case, the program is aborted. If RETURN or INIT is specified, *expression* must be a variable.
NULL	The NULL value is passed to the subprogram. NULL is not permitted for CALL MODULE or CALL TAC.
INIT	INIT is used to assign a value to *expression*. If *expression* is a vector or matrix, all the components receive the value *literal*.
expression1	*expression1* may only contain literals, NULL or functions whose arguments are literals (not CURRENT DATE/TIME/TIMESTAMP). This means that *expression1* must be able to be calculated at compilation time.

NOCHECK	Specifies that *expression1* is not to be checked for any existing CHECK clause which apply to *expression*.
INDICATOR	INDICATOR is used to create an indicator variable. The value of the indicator variable specifies whether the transfer parameter contains the null value or a defined value. The INDICATOR specification is only permitted with CALL MODULE and CALL TAC.

Example

The external DRIVE subprogram "empcorr", located in the preset library is called. the parameter &cmp1 is passed to the subprogram.

```
CALL empcorr USING &cmp1;
```

Relationship to other statements

- CALL statements that call a UTM program unit (CALL TAC) are not permitted in programs started with ENTER.
- CALL statements which call an old style program are not permitted in programs which are started with ENTER.
- Remote CALL statements within a dispatch block are executed simultaneously, not sequentially (see the DISPATCH statement).
- Remote CALL statements are not permitted in programs started with ENTER.
- If the subprogram was compiled with the compiler option OPTION OBJECT=ON, no *library* may be specified when calling the program.
- If a program was compiled with the compiler option OPTION DISTRIBUTION= ON, DRIVE/WINDOWS searches for a program in the local or remote system in accordance with the distribution information.

Access rules for databases

The following rules apply for subprograms called in the local system:

– If different database systems are assigned to the calling program and subprogram called (DBSYSTEM ≠ OFF), then the CALL statement is aborted. You can only assign different database systems if CALL calls intermediate code or object code which was generated using another database system in a previous DRIVE session.

– If the SESAM V2.x database system is assigned to the calling program and subprogram called (DBSYSTEM = SESAMSQL), the CALL statement is only executed if there is no transaction open for the calling program or if intermediate code has been generated for the subprogram called. It should be noted that DRIVE/WINDOWS always uses the last member to have been processed, irrespective of whether this contains a source (S-member), an intermediate code (X-member) or an object code (R-member).

– If a database system is assigned to the calling program (DBSYSTEM ≠ OFF) and not to the subprogram called (DBSYSTEM = OFF), the subprogram called accesses the same database system as the calling program.

– If a BS2000 database system is assigned to the called subprogram (DBSYSTEM = UDS / SESAM / SESAMSQL), the CALL statement is only executed if this database system matches the loaded variant.

– If the called subprogram is an old style program, the CALL statement is aborted if the UDS database system is assigned to the calling program (DBSYSTEM = UDS).

– If the called subprogram is an old style program, the CALL statement is only executed if there is no (new style) transaction open for the calling program.

CASE
Program conditional branches

This application is valid

– in TIAM and UTM mode
– in program mode

CASE indicates the start of a CASE block. The end of the block is indicated by END CASE. The first statement after CASE must be an OF statement. The sequence of statements following an OF statement through to the next OF statement or, if there is no further OF statement, through to END CASE is known as an OF branch.

Conditional branches are defined within a CASE block. This involves comparing values with patterns defined in the OF branches. If the comparison returns the truth value TRUE, DRIVE/WINDOWS branches to the subsequent statements in the OF branch.

Statements with CASE can be nested as often as required, i.e. CASE ... END CASE can also occur within the OF branches. CASE, IF and CYCLE can be nested, but must not overlap.

Within a program, the statement CASE [ALL] [expression1] must be followed by a semicolon.

```
CASE [ ALL ] [ expression1 ]
     { OF { expression2, ... | condition | REST } [ programming ... ] } ...
```

ALL — ALL executes **all** OF branches for which the corresponding *condition* is found to be TRUE.

expression1 — The condition *expression1=expression2* is formed from *expression1* and *expression2* and then evaluated. If the truth value TRUE is returned, the statements in the OF branch are executed.

If *expression1* is not specified, the OF statement must contain a *condition*.

The value of *expression1* is calculated. It remains constant until the END CASE statement is reached.

The equals sign is used as the comparison operator.

OF	Either a *condition* is evaluated or *expression2* and *expression1* are combined using an equals sign to form a *condition* which is then evaluated. If you did not specify ALL and the comparison returns the value TRUE, the OF branch is executed and the program branches to END CASE. If you specified ALL and the comparison returns the value TRUE, the OF branch is executed and the program branches to the next OF branch (if there is one). If none of the comparisons return the value TRUE, DRIVE/WINDOWS executes the statements following REST or, if REST was not specified, terminates the branch at END CASE.
expression2	If more than one expression has been specified with OF, the results of the individual comparisons of *expression1* with *expression2* are ORred and then checked for the truth value (TRUE or FALSE).
condition	If the comparison with *condition* returns the value TRUE, the OF branch is executed. *condition* must not contain a *column* (see the *column* metavariable in the SQL Directories [8-11]).
REST	OF REST is used to evaluate the results of previous OF branches. OF REST applies if all previous conditions of the OF statements resulted in FALSE. The statement may occur only once and must be the last OF branch.
programming	See the metavariable *programming*

Example

The variable &cross is defined as a vector with a repetition factor of 3. The program checks whether &cross(1), &cross(2) or &cross(3) contains a value other than ' ' (i.e. whether an entry has been made).
Depending on the component for which an entry has been made, one of the subprograms "terminate", "display" or "reminder" is called.
If no entry has been made, the message "Input please (DUE) " is outputein Meldungsfenster ausgegeben.

```
...
DECLARE VARIABLE &cross(3) CHAR (1);
...
CASE;
    OF &cross(1) <> ' ' CALL terminate;
    OF &cross(2) <> ' ' CALL display  USING &database, &project, &delay;
    OF &cross(3) <> ' ' CALL reminder   USING &database, &project;
OF REST SEND MESSAGE ‘Input please (DUE)‘ WAIT;
END CASE;
...
```

Defining error exits

If an error occurs during comparisons or calculations, a branch is made to END CASE and, if applicable, WHENEVER is evaluated. Execution errors in DRIVE statements within OF branches are treated as described for the individual statements.

CLEAR
Reset variable or DRIVE form

This application is valid

– in TIAM and UTM mode
– in program mode

CLEAR sets variables to their starting values and deletes DRIVE forms which have not yet been output (DECLARE FORM), with the exception of the page header and footer.

In particular, CLEAR can be used to reset the contents of input and output fields. You can reset input and output fields either individually or in groups. The statement also applies to screen variables (CLEAR *screenvariable*).

Variables are reset to the INIT value declared in DECLARE VARIABLE.

If you did not declare an INIT value when you declared the variables, the variable is set to the initial value for the relevant data type when the CLEAR statement is executed (see DRIVE Programming Language [2]). Variables with a time or date data type which have been declared as TEMPORARY are in this case assigned the current date (CURRENT DATE), the current time (CURRENT TIME) or the current timestamp (CURRENT TIMESTAMP).

The following applies to DRIVE forms: All FILL statements for the specified form which have not yet been output using DISPLAY are reset. This means that the statement sequence

```
DECLARE FORM name ...
...
FILL name ...
FILL name ...
CLEAR name
DISPLAY name ...
```

results in the output of a form which contains only the page header and footer declared in DECLARE FORM (TTITLE and BTITLE) but which is otherwise empty.

CLEAR has no effect on an implicit DISPLAY resulting from a screen overflow in a FILL statement.

```
CLEAR { variable | form-name }, ...
```

---

variable	Name of the variable to be reset. The name must refer to a variable which is valid in the current program unit (see DRIVE Programming Language [2]).
form-name	Name of the DRIVE form which is to be reset and which has been declared in DECLARE FORM.

CLOSE FILE
Close file

This application is valid

– in TIAM and UTM mode
– in program mode

CLOSE FILE closes an open file.

```
CLOSE FILE file
```

---

file — Logical name of a file (max. 31 characters).

The file must have been defined with this name in the program using the DECLARE FILE statement.

Special characteristics in UTM mode

– ISAM files with the attribute SHARED-UPDATE=YES are not closed until the UTM application terminates.
– Files opened using "INPUT" are not closed until the UTM application terminates.
– Files are closed on screen display (e.g. DISPLAY ..., SEND MESSAGE, = end of dialog step) and re-opened in the next dialog step if necessary. In such cases, DRIVE/WINDOWS monitors the current file position.

COMPILE
Compile program

This application is valid

– in TIAM and UTM mode
– in interactive mode

COMPILE is used to check a source program for syntax and semantic errors. Options for controlling compilation can be specified (e.g. that the intermediate code generated in an error-free compiler run is to be stored in the DRIVE library). The options specified with COMPILE overwrite those in the source program.

An interpreter listing is generated when the source program is compiled. The option OPTION LISTING=LIST/LIBRARY/BOTH controls the output of this listing which consists of

– a header (member name, library name, date and time of the check),
– the source listing. This contains all source statements and the resolution of all the COPY members, USE members and EUA forms which are present.
– an overview of the compiler options, the number of errors, and the size of objects.

If errors occur during compilation, error messages are additionally entered in the source listing.

A program can only be compiled if no transaction is open.

If COMPILE identifies errors in a source program located in EDT work file 0, no error list is generated. At the next EDT statement, the error messages are inserted into the source program contained in EDT work file 0 (see the EDT statement). The interpreter listing is written to EDT work file 9.

If COMPILE is specified without operands, the source program located in EDT work file 0 is checked. The OPTION entries LISTING=LIBRARY and CODE=ON are not evaluated in this case and are skipped without warning.

```
COMPILE [ library1(member-name1) | member-name1 ]

        [ INTO { library2(*) | library2(member-name2) | member-name2 } ]

        [ OPTION ]
```

library1	Specifies the DRIVE library (max. 54 characters) from which the source program to be compiled is read. *library1* may also be the file link name of the DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library1* first as a file link name, then as a library name. If the DRIVE library has been preset using the PARAMETER DYNAMIC LIBRARY statement, you can omit the *library1* specification.
member-name1	Specifies an S-type member (max. 31 characters) to be compiled. If there is no *library1* specification, the library which has been preset in PARAMETER DYNAMIC LIBRARY is used. An error message is issued if *member-name1* is not present in the specified DRIVE library. If you do not specify *member-name1* then the source which is present in EDT work file 0 is analyzed and compiled.
INTO	INTO identifies the members and DRIVE libraries to which the intermediate code, object code, or interpreter listing is written if CODE=ON, OBJECT=ON, or LISTING=LIBRARY has been specified with OPTION. With the option CODE=ON, a member of type X is created, containing the generated intermediate code. With the option OBJECT=ON, a member of type R is created, containing the object code. With the option LISTING=LIBRARY or =BOTH, a member of type P is created, containing the interpreter listing.

	The following applies if the INTO clause is not specified: Members of type X and P are overwritten in the library named in *library1* or in the current DRIVE library. Members of type R are stored in *library1* under a name that is constructed from the first 4 and last 3 characters of *member-name1* (4-3 rule); invalid characters are replaced by #.
library2	Specifies the DRIVE library (max. 54 characters) to which the generated intermediate code, object code, or interpreter listing is written. *library2* may also be the file link name of the DRIVE library (in accordance with BS2000 conventions). If *library2* is not specified, *library1* is used.
*	The member containing the intermediate code, object code or interpreter listing is assigned the name *member-name1*.
member-name2	Specifies the member containing the intermediate code, object code or interpreter listing. The name for members of type X or P must not exceed 31 characters; the maximum length for members of type R is 7 characters. If the name for an R-type member exceeds 7 characters, it is reduced by applying the 4-3 rule. Invalid characters are replaced by #. If you do not specify *library2*, *member-name2* is saved in *library1*. If you do not specify either *library1* or *library2* then the library specified in PARAMETER DYNAMIC LIBRARY is used
OPTION	Specifies the desired options (see the OPTION statement).

Examples

The source with the name "prog1" is read from the current DRIVE library and compiled. The intermediate code is stored under the same name in the current DRIVE libary.

```
COMPILE prog1 OPTION CODE=ON
```

The source with the name "prog1" is read from the current DRIVE library and compiled. The intermediate code is stored under the name "prog2" in the current DRIVE library.

```
COMPILE prog1 INTO prog2 OPTION CODE=ON
```

The source with the name "prog1" is read from the DRIVE library "lib1" and compiled. The intermediate code is stored under the name "prog2" in the DRIVE library “lib2”.

The specification "lib1" must not be omitted, because the DRIVE library was not defined with the PARAMETER DYNAMIC LIBRARY statement.

```
COMPILE lib1(prog1) INTO lib2(prog2) OPTION CODE=ON
```

The source with the name "prog1" is read from the current DRIVE library and compiled. The interpreter listing is stored under the name "list" in the DRIVE library "lib2".

```
COMPILE prog1 INTO lib2(list) OPTION LISTING=LIBRARY
```

The source with the name "prog1" is read from the current DRIVE library and compiled. The interpreter listing is stored under the name "list" in the DRIVE library "lib2". At the same time, the interpreter listing is output to SYSLIST in TIAM mode or to the central print file in UTM mode.

```
COMPILE prog1 INTO lib2(list) OPTION LISTING=BOTH
```

CONTINUE
Continue loop cycle or debugging run

This application is valid

– in TIAM and UTM mode
– in program and debugging mode with distinct functions (v.i)

CONTINUE has different functions depending on the mode:

– In program mode, CONTINUE CYCLE embedded in a statement sequence enclosed between CYCLE ... END CYCLE can be used to jump prematurely to END CYCLE.
– In debugging mode, program execution is continued (in debugging mode) after CONTINUE.

You may only specify the CONTINUE statement without the CYCLE operand in debugging mode.

```
CONTINUE [ CYCLE ]
```

CYCLE

CONTINUE CYCLE is only permitted in program mode and must be within a cycle (CYCLE ... END CYCLE).

The loop is repeated or terminated, depending on the condition defined for it.

COPY
Insert copy member

This application is valid

– in TIAM and UTM mode
– in interactive and program mode with distinct functions (v.i)

COPY is used to insert copy members into a program. These consist of a sequence of statements or parts of statements and must be stored in a DRIVE library (see the DRIVE Programming System [1] manual). Statement units can be concatenated as COPY members to form a complete statement. COPY members may be located in either the declaration section or the program body.

Copy members may not be nested, i.e. copy members may not include COPY statements.

In program mode, COPY members are copied to the source during compilation. The usual source restrictions apply (e.g. program structure).

In interactive mode, only the first statement in the specified copy member is output at the terminal. The statement may be modified and then executed.

```
COPY { library(member-name) | member-name }
```

---

library	Specifies the DRIVE library (max. 54 characters) from which the copy member is read. *library* may also be the file link name of the DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name. You can omit the *library* specification if the DRIVE library has been preset in the PARAMETER DYNAMIC LIBRARY statement.
member-name	Specifies the copy member (max. 31 characters). The copy member of type S is copied from the specified DRIVE library. *member-name* must be specified, otherwise an error message is issued. If you do not specify *library*, the library specified in PARAMETER DYNAMIC LIBRARY is used.

An error message is output if the COPY member is not present in the DRIVE library.

Example 1

The COPY member "MITVAR" from the "DRI.LIB" library is inserted into a program.

```
COPY "DRI.LIB"(MITVAR)
```

Example 2

A *select-expression* completes a cursor declaration as COPY member "select1".

```
DECLARE c1 CURSOR FOR COPY select1;;
```

– The first semicolon terminates the COPY statement
– The second semicolon terminates the DECLARE statement

If the copy member "select1" has the contents:

```
SELECT * FROM tab1 WHERE ...
```

the result is:

```
DECLARE c1 CURSOR FOR SELECT * FROM tab1 WHERE ...;
```

CYCLE
Program loop

This application is valid

– in TIAM and UTM mode
– in program mode

CYCLE marks the beginning of a CYCLE block whose end is defined with END CYCLE. A loop in a CYCLE block can be traversed any number of times or until a terminating condition is met.

A loop which accesses a cursor (CYCLE cursor-name) is executed until the end of the cursor table has been reached. This statement is a simple method of processing all the rows of a cursor table.

If CYCLE is specified without operands, it defines an endless loop which can be terminated only with BREAK CYCLE or by aborting the program.

Loops can be nested to any depth. The only restriction is the amount of memory required by DRIVE/WINDOWS for processing.

Loops, conditions (IF) and branches (CASE) may likewise be nested, but are not permitted to overlap.

A semicolon must follow the CYCLE statement in a program.

```
CYCLE [ cursor-name INTO variable, ... |

      WHILE condition |

      FOR variable1=value-expression1 [ BY value-expression2 ] TO value-expression3 ]
```

---

cursor-name — Name of a cursor. The cursor must have already been declared and closed.
CYCLE *cursor-name* is simply an output function. If no row is found, running the loop is no longer useful. The loop is terminated and the cursor is closed.

INTO	Causes a row to be transferred from the cursor table *cursor-name* to *variable* at the beginning of each cycle. The loop is traversed as long as there are entries in the cursor table. The cursor is opened at the start of the first cycle. It is then positioned at the next row in the cursor table and the variable is supplied with values from these columns. The loop is terminated when the last row in *cursor-name* is reached. The cursor is closed at "TABLE END" or following BREAK CYCLE. The CYCLE ... INTO statement implicitly executes all SQL calls. These must be formulated explicitly in a 3GL environment (OPEN, FETCH and CLOSE). If WHENEVER was used to define an error exit ≠ CONTINUE for "TABLE END", it is not executed if the event "TABLE END" occurs with CYCLE *cursor-name* INTO *variable*,... .
variable	Name of a variable.
WHILE	If WHILE is specified, the loop is traversed so long as *condition* returns the true value.
condition	Condition(s) to be applied to data values. At the beginning of each cycle, a check is made to see if *condition* is still met. The loop is traversed as long as *condition* returns the true value. *condition* may not contain *expression* or *.
FOR	At the beginning of each loop the value of *value-expression1* is assigned to *variable1*. Before the loop is traversed for the first time, a check is run to see if the new value of *variable1* is smaller than or equal to *value-expression3* (greater than or equal to if *value-expression2* has a negative value). If this is the case, the sequence of statements between CYCLE FOR and END CYCLE is processed. If not, the loop is ended with END CYCLE. Before any further loop, the value of *value-expression* is added to *variable1* and the comparison with *value-expression3* is carried out again. If errors occur while the values are calculated and compared, the loop is ended and a branch is made to END CYCLE. From this point, it may be possible to handle the error with WHENEVER. *-expr1* and the result of the increment before every loop must be compatible with *variable1*.

variable1

variable1 controls the FOR loop. *variable1* must not be structured, indexed, redefined or redefining. Only the data types NUMERIC, DECIMAL, INTEGER, SMALLINT, EXTENDED DECIMAL or XDEC are permitted.

An explicit DRIVE statement (e.g. SET or RETURN clause for an input/output) must not assign a value to *variable1* within the loop. *variable1* cannot be used again as the controlling variable within the current loop.
If the FOR loop is ended as it reaches *value-expr3* or BREAK CYCLE, *variable1* holds the most current data value. This value may be different from *value-expr3*.

value-expr1

Specifies the start value of the FOR loop. If the value of *value-expr1* at the start of the FOR loop is the null value, the loop is ended with END CYCLE.

BY value-expr2

Specifies the increment for the FOR loop. *value-expr2* may not have the value 0.
If *value-expr2* has the null value, the FOR loop is aborted with an error message.
At the beginning of the loop the current sign of *value-expr2* determines the direction of the FOR loop for the entire loop. The value of *value-expr2* is constant for this time.

TO value-expr3

Specifies the end value of the FOR loop. If the value of *value-expr3* at the start of the FOR loop is the null value, the loop is ended with END CYCLE.
The value of *value-expr3* remains constant for the entire loop.

If CYCLE is used to process a cursor, COMMIT WORK is only permitted if the cursor was saved and restored with STORE and RESTORE, respectively.

Defining error exits

Defining an error exit with WHENEVER is the only way to avoid a program abortion due to a semantic error identified during loop evaluation. It must be defined in the declaration section. The program is continued in accordance with the error exit defined with WHENEVER following END CYCLE (see the WHENEVER statement).

Example 1

All the rows in cursor table "cr" are transferred to the variable &var until the end of the cursor table has been reached (&DML_STATE='TABLE END'):

```
CYCLE cr INTO &var.*;
/* Processing */
...
END CYCLE;    /* CYCLE End of cursor */
```

The system variable &ERROR_STATE='OK' is set.

The statements shown implicitly contain the following sequence of statements:

```
OPEN cr;
CYCLE;
FETCH cr INTO &var.*;
IF &DML_STATE='TABLE END'
   THEN
   BREAK CYCLE;
END IF;
/* Processing */
...
END CYCLE;
CLOSE cr;
```

DRIVE/WINDOWS closes the cursor (CLOSE cr) and the STATUS returned by the CLOSE operation is passed to the system variable &ERROR_STATE.

Notes on the example:

– The loop must not contain a transaction variable.
– If a cursor is opened and no rows are found, the processing part of the loop is not executed.

Example 2

All rows in the "print-cursor" cursor table are to be transferred to the variable &printrow.* until the end of the table is reached. The output fields of the list form "personnel" are to be filled with this data.

```
CYCLE print-cursor INTO &printrow.*;
   FILL personnel TABLE NAMES &printrow.*;
END CYCLE;
```

Example 3

A loop is used to assign the value NULL to all the fields of the vector &language(5).

```
SET &index = 1;
CYCLE WHILE &index <= 5;
   SET &language(&index) = NULL;
   SET &index = &index + 1;
END CYCLE;
```

Example 4

Example 4 behaves in the same way as example 3.

```
CYCLE FOR &index=1 TO 5;
   SET &language(&index) = NULL;
END CYCLE;
```

Example 5

A loop is used to assign the value NULL to the last four fields of the vector &month(12).

```
CYCLE FOR &index=12 BY -1 TO 9;
   SET &month(&index) = NULL;
END CYCLE;
```

Example 6

The following loop is not executed, since the termination condition *value-expression3* > *&index* is already fulfilled before the first iteration, where 12 > 10.

```
CYCLE FOR &index=10 BY -1 TO 12;
   SET &month(&index) = NULL;
END CYCLE;
```

Example 7

The screen form "form1" continues to be output and the variable &article.* continues to be entered as a record in the table "v_article" as long as the system variable &KFKEY has the value 'K3' (i.e. until the K3 key is pressed).

```
PARAMETER KFKEY = 'K3';
...
CYCLE;
   DISPLAY form1;
   IF &KFKEY = 'K3'
      THEN BREAK CYCLE;
      ELSE INSERT INTO v_article VALUES (&article.*);
      ...
   END IF;
END CYCLE;
```

DEBUG
Start program and switch to debugging mode

This application is valid

– in TIAM mode
– in interactive mode

The DEBUG statement can be used to switch to debugging mode and to start a DRIVE program (with subprogram) or an external DRIVE subprogram under the control of the debugger, thus enabling a controlled analysis and execution of the program.

For this to be possible, the program must have been compiled and an interpreter listing must be available.
If there is no intermediate code corresponding to the source, DEBUG causes implicit compilation to be performed. However, no interpreter listing is implicitly created. Instead, it must be created using, for example, the statement OPTION LISTING=LIBRARY.

Debugging mode does not modify program processing. Nevertheless, it is possible to interrupt program execution and perform certain operations (see below). In this way, the debugging mode allows you to detect and locate program errors and determine their cause.

On starting the debugging mode, an initial breakpoint is automatically set after the PROCEDURE statement of the program started with DEBUG.

As soon as the prompt (*) appears, you can enter debugging statements at this initial breakpoint.

The following debug statements can be entered at breakpoints:

– AT
– BREAK
– BREAK DEBUG
– CONTINUE
– DISPLAY FORM
– DISPLAY LIST
– REMOVE
– SET
– TRACE

If subprograms are called with the CALL or DO statement, the USING clause may be omitted from the DEBUG statement. In this case, DRIVE/WINDOWS requests the required parameters by means of parameter prompting. You can supply the parameters with the SET statement. DRIVE/WINDOWS stops the called subprogram after the PROCEDURE statement. You may then enter debug statements.

In mixed mode operation, the following comments apply to the debugging mode:
If, in a new style program, an old style program is called with CALL, the old style program is executed. Processing then switches to new style debugging mode.
If, in a new style program, an old style program is called with DO, debugging mode is terminated in the new style program and the old style program is executed. Processing then reverts to new style.
If, in an old style program, a new style program is called with DO, processing switches to new style. However, the program is not executed in debugging mode.

The SET statement cannot be used to assign field attributes to a variable or global attributes to a screen variable in debugging mode.

The BREAK DEBUG statement causes the program to exit debugging mode and switch to interactive mode.

```
DEBUG { library(member-name) | member-name }

      [ USING { expression | NULL }, ... ]
```

---

library — Specifies the DRIVE library (max. 54 characters) from which the program is read to be analyzed and executed under the control of the debugger.

library may also be the file link name of the DRIVE library (in accordance with BS2000 conventions).

DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name.

You can omit the *library* specification if the DRIVE library has been preset in the PARAMETER DYNAMIC LIBRARY statement.

member-name — Name of the member (max. 31 characters) which contains the program.

DRIVE/WINDOWS searches for the last member to have been processed, irrespective of whether this contains a source (S-member) or an intermediate code (X-member).

If you do not specify *library*, the library which has been preset in PARAMETER DYNAMIC LIBRARY is used.

If you do not specify *member-name* then DRIVE/WINDOWS uses the source which is present in EDT work file 0.

USING	Enables parameters to be passed to the program to be started. This program must have been defined with PROCEDURE ... USING. The parameters must not contain any variables at compile time. If the parameters are not compatible, or if the USING clause is omitted from the DEBUG statement, DRIVE/WINDOWS displays the following message: DRI0561 PLEASE SUPPLY USING PARAMETERS. At this point, the user may either supply all parameters with the SET statement or exit debugging mode with the BREAK DEBUG statement. In debugging mode, you are also prompted for DRIVE subprograms called with DO or CALL.
expression	Identifies the parameters that are passed to the program to be started (send fields). The following may be passed as transfer parameters: literals, aggregates composed of literals, and arithmetic expressions containing no variables. Variable values may also be passed. *expression* must be calculable at compile time.
NULL	Passes the null value to the program to be started.

Error handling

If an error occurs in a statement when in debugging mode, a check is made to determine if WHENEVER has been declared for that error.

If WHENEVER was not declared, the program will stop before the statement containing the error.

If WHENEVER was declared, one of the following is initiated, depending on the WHENEVER operation defined:
If the declared WHENEVER operation is CALL or CONTINUE, the program is not interrupted. If the operation is BREAK, the program stops before the statement containing the error.

In debugging mode, if a transaction cannot be rolled back because a COMMIT WORK statement is missing, control is passed to the final breakpoint.

If an error occurs in a debug statement in debugging mode, DRIVE/WINDOWS first displays an error message: DRI0553 PLEASE ENTER DEBUG STATEMENT. The program stops at the breakpoint at which the invalid debug statement was entered. Debug statements which follow the one containing the error are not executed. Corrections made to the cause of the error do not take effect until the testpoint is reached again.

Relationship to other statements

– If neither *library* nor *member-name* is specified with DEBUG, the OPTION entries LISTING=LIBRARY and CODE=ON in the source are not executed and are skipped without warning.

– If an analysis phase is included in DEBUG, the following options specified in the source program are ignored:

  UREF=ON, CODE=ON.

  Other options are executed, e.g. if LISTING=LIBRARY is specified, the interpreter listing will be written to a library member.

i Tracing cannot be activated (TRACE) if the program is located in EDT work file 0.

Access rules for databases

– The DEBUG statement is only executed if no (new style) transaction is open for the interactive mode.

– If different database systems are assigned to interactive mode and the program called (DBSYSTEM ≠ OFF), then the DEBUG statement is aborted. You can only assign different database systems if DEBUG calls intermediate code or object code which was generated using another database system in a previous DRIVE session.

– If the SESAM V2.x database system is assigned to both interactive mode and the called program (DBSYSTEM = SESAMSQL), then the DEBUG statement is only executed if no transaction is open for interactive mode.

– If a database system is assigned to interactive mode (DBSYSTEM ≠ OFF) and not to the called program (DBSYSTEM = OFF), then the called program accesses the same database system as interactive mode.

– If a BS2000 database system is assigned to the called program (DBSYSTEM = UDS / SESAM / SESAMSQL), then the DEBUG statement is only executed if this database system matches the loaded variant.

– If the called program is an old style program, the DEBUG statement is aborted if the UDS database system is assigned interactive mode (DBSYSTEM = UDS).

Example

The "test" program is tested in debugging mode.

All the executable program statements are counted. The statements in lines 15, 17 and 20 to 55 of the interpreter listing for the main program, "test", are testpoints. The variable &var1 is first set to 1, output at the printer and then the screen. The program run is then continued.

At the statement in line 33 of the compiler listing for the"test2" subprogram which is located in the predefined library, the variable &subvar1 is set to 2 and output at a printer.

As of line 99 of the main program, "test", program tracing is activated and the debugging run continues.

```
DEBUG test;          /* DRIVE halts at the first executable statement*/
                     /* in the program body                          */
AT ALL COUNT;
AT 15 SET &var1 = 1
AT 17 DISPLAY LIST &var1
AT 20 - 55 DISPLAY FORM &var1
AT * CONTINUE
AT test2 33 SET &subvar1 = 2
AT test2 33 DISPLAY LIST &subvar1
AT 99 TRACE
CONTINUE             /* Only now does the debugging run continue   */
...
```

DECLARE CONSTANT
Define constant

This application is valid

– in TIAM and UTM mode
– in program mode

DECLARE CONSTANT is used to define constants.

Within a DRIVE program, the defined constant can syntactically represent a variable, but not a literal.

DECLARE CONSTANT must precede all executable statements in the declaration section of the program.

```
DECLARE CONSTANT { var-name
                 { literal |
                   MSGSTRING ( value-expr1[ [, value-expr2], name ] ) } }, ...
```

---

var-name	Name of a constant *var-name* must start with an ampersand (&) and may be up to 32 characters in length.
literal	Literal whose value is assigned to *var-name*.
MSGSTRING	The value of the following expression is generated at compile time and stored in the intermediate code. The message is taken from the message file (= current MIP file) which is specified by the parameter. If you specify three parameters (*value-expr1*, *value-expr2*, *name*) then *value-expr2* is ignored. If you specify two parameters then the second parameter must be *name* (*value-expr1*, *name*). If no message can be unambiguously identified, DRIVE/WINDOWS returns: MESSAGE NOT FOUND.
value-expr1	Second part of the message code (= message number). *value-expr1* must be a numeric expression without decimal places. If *value-expr1* contains the null value, the complete expression results in the null value.

value-expr2	*value-expr2* is ignored if a value is specified. *value-expr2* is supported for reasons of compatibility only.
name	First part of the message code (= identification of the system component that generates the message). *name* can have a maximum length of 3 characters.

DECLARE FILE
Define file

This application is valid

– in TIAM and UTM mode
– in program mode

DECLARE FILE defines files, i.e. you use the statement to declare the logical name of a file to the program. This logical name is assigned to the file when it is opened with the OPEN FILE statement.

This statement also defines a replacement character for representing null values. If no character is defined for representing null values, DRIVE/WINDOWS issues an error message when an attempt is made to write a null value to a file.

DECLARE FILE must precede all executable statements in the declaration section of the program.

```
DECLARE FILE { file [ NULL character ] }, ...
```

---

file	Logical name of a file (max. 31 characters).
NULL	Defines a character for representing null values. This character is written to the file or expected in the file in place of the null value a number of times corresponding to the length of the field containing the null value.
character	Specifies the null replacement character (one character). You should not specify the following items for *character*: – in alphanumeric data fields: a blank (␣) – in numeric data fields: the characters + - , or . since this could lead to unexpected results. The character must be specified as an alphanumeric literal (*char-literal*, see the *literal* metavariable) or as a hexadecimal character (*hex-literal*, see the *literal* metavariable).

DECLARE FORM
Define DRIVE screen form

This application is valid

– in TIAM and UTM mode
– in program mode

DECLARE FORM is used to define a DRIVE screen form for data input and output.

DECLARE FORM creates a memory area for a screen form and defines its layout. The memory area is filled in using FILL statements. Output formatting is concluded with DISPLAY *form-name*. At the same time, output is initiated.

DECLARE FORM must precede all executable statements in the declaration section of the program.

The program is aborted at runtime if the page header or footer contains an input or output field which does not fit into the space provided.

```
DECLARE FORM form-name

            [ PERMANENT | TEMPORARY ]
            [ NULL null-value ]
            [ { COLUMNS n | LINES n }, ... ]

            [ TTITLE [format] { [RETURN] expression[INIT expression1 [NOCHECK] ]
                                  [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |
                              NEWLINE n |
                              TABULATOR n |
                              BLANK n }, ... ]

            [ BTITLE [ format ] { [RETURN] expression[INIT expression1 [NOCHECK] ]
                                  [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |
                              NEWLINE n |
                              TABULATOR n |
                              BLANK n }, ... ]
```

---

form-name	Name of the DRIVE form (max. 31 characters).
PERMANENT	The contents of a form defined with PERMANENT are retained beyond the end of a subprogram and are available when the same subprogram is called again.
TEMPORARY	Default The contents of a form defined with TEMPORARY are dropped at the end of a subprogram.
NULL	Definition of a character which is to represent the null value. This overwrites a null value representation defined in PARAMETER.
null-value	Specifies the null value character (max. 1 character). The null value representation is specified for the character data type (CHARACTER, VARCHAR) or for numeric data types (NUMERIC, DECIMAL, INTEGER, SMALLINT, REAL and FLOAT). See also the metavariable *null-value*. The character null value representation is also valid for the data types DATE and TIME. The numeric null value representation is also valid for the data type INTERVAL.
COLUMNS n	Specifies the number of columns per screen. COLUMNS may only be specified once within DECLARE FORM. $0 < n \leq$ number of screen columns Default value: number of columns in the screen involved.
LINES n	Specifies the number of lines per screen. LINES may only be specified once within DECLARE FORM. $0 < n \leq$ number of screen lines minus 1 Default value: number of screen lines in the screen involved minus 1 The last line is reserved as a message line. Line count (TTITLE) + line count (BTITLE) < LINES

TTITLE	Defines a page header to be output with each new screen. The total number of lines for TTITLE, BTITLE and FILL must not exceed: the number of screen lines - 1. The last line is reserved for use as a message line. There must always be at least one FILL line. In the case of a screen overflow, TTITLE is output as the page header.
BTITLE	Defines a form footer output with each new screen. The number of lines that can be defined for a page footer is the same as the number of lines under TTITLE. In the case of a screen overflow, BTITLE is output as a page footer.
format	*format* specifies the format of the screen form. If TABLE is specified for *format*, NEWLINE is not permitted. If LINE is specified for *format*, TABULATOR and BLANK result in line feed with an empty line.
RETURN	Specifies that a variable becomes an entry field, which may be preset. A variable for which RETURN has been specified may be used as an entry field only once for each form.
expression	Defines output and/or entry fields for the screen form. *expression* may be one or more variables (including system variables) and/or one or more literals. When RETURN or INIT is specified, *expression* must be a variable not qualified with ".*".
INIT	An initial value is assigned to *expression*. The INIT clause is permitted only if *expression* is a variable. If *expression* is a vector or a matrix, all components are given the corresponding initial value *literal* or NULL.
INIT expression1	*expression1* may be only a *literal*, NULL or a function whose arguments are literals (except CURRENT DATE/TIME). *expression1* must be calculable at compile time.
NOCHECK	A CHECK clause (see the metavariable *check*) specified when declaring the *expression* variable is not evaluated for the assignment of the initial value. NOCHECK is not permitted for redefined variables or variables which redefine other variables.

ATTRIBUTE (attribute,...)

Screen forms are assigned field attributes.

Only the following may be specified as color attributes:

GREEN, RED, WHITE and YELLOW.

Default values for output fields are: VISIBLE, PROTECTED, NOUNDERLINE, NORMALINTENSITY.

Default values for entry fields are: VISIBLE, UNPROTECTED, NOUNDERLINE, HIGHINTENSITY.

INVISIBLE may be specified for entry fields only (with RETURN).

mask

Defines the representation options for masked input and output (= output editing).

mask may only be specified if *expression* is a simple variable or a simple component.

NEWLINE n

Defines the position of the screen fields. NEWLINE causes an advance of *n* lines. The current line is completed with NEWLINE 1. Any subsequent data is written in the next line.
Even if the current line has already been filled, i.e. it already contains as many characters as are defined in the COLUMNS specification of the corresponding DECLARE statement, NEWLINE 1 does not cause a blank line to be output. Instead it positions to the next line.

If *n* has the value "0", a conditional line feed is performed, i.e. if the current line is blank, NEWLINE 0 is ignored; if the current line is not blank, NEWLINE 0 has the same effect as NEWLINE 1.

TABULATOR n

Defines the position of the screen fields.

The output is continued from column *n*. If the value is less than the current column position, a line or page feed is performed. The resulting gap is filled with blanks.

If TABULATOR is specified without a value *n*, either this has no effect or only one line feed is performed. The following applies:
1 ≤ TABULATOR ≤ COLUMNS

BLANK n

Defines the position of the screen fields. BLANK causes *n* blanks to be inserted. This may result in a line feed or page feed.

n is an integer.

Example

The page header of the screen form "output-staff" consists of five lines and the footer of two lines.
The header has the following structure (the current date and time are output in the first line):

```
1995-12-20                                                  13:30:31

                                  Staff
                                  ------
```

The footer consists of a blank line and a line containing 80 equals signs (=).

```
DECLARE VARIABLE &date DATE;
DECLARE VARIABLE &time TIME;
...
DECLARE FORM output-staff
   TTITLE &date,' '(50),&time,NL 2,
          ' '(30),'Staff',NL 1,
          ' '(29),'-'(13),NL 2
   BTITLE NL 2,'='(80);
```

DECLARE LIST
Define list form

This application is valid

– in TIAM and UTM mode
– in program mode

DECLARE LIST is used to define a list form. This is done by creating a memory area for the list form and defining the layout for the printed list. The memory area is filled using FILL statements. Output formatting is concluded with DISPLAY *list-name*.

The system variable &PAGES records the number of pages printed thus far. &PAGES is incremented following each form feed. Initialization of the page count requires that an initializing value be explicitly supplied. This is not necessary at the start of a program.

In UTM applications, the list file must be generated. There are three options for generation:

– the `DRI.LIST.FILE` already exists,
– the file is assigned via the file link name `DRILIST`,
– the `DRI.LIST.FILE` is generated when the first DRIVE/WINDOWS DISPLAY statement is executed.

The actual output on the printer occurs at the next STOP or LIST (see the LIST and STOP statements).

DECLARE LIST must precede all executable statements in the declaration section of the program.

```
DECLARE LIST list-name

            [ PERMANENT | TEMPORARY ]
            [ NULL null-value ]
            [ { COLUMNS n | LINES n }, ... ]

            [ TTITLE [ format ] { expression [ mask ] |
                                  NEWLINE n |
                                  TABULATOR n |
                                  BLANK n }, ... ]

            [ BTITLE [ format ] { expression [ mask ] |
                                  NEWLINE n |
                                  TABULATOR n |
                                  BLANK n }, ... ]
```

list-name	Name of the DRIVE list form (max. 31 characters).
PERMANENT	The contents of a list defined with PERMANENT are retained beyond the end of a subprogram and are available when the same subprogram is called again.
TEMPORARY	Default The contents of a list defined with TEMPORARY are dropped at the end of a subprogram.
NULL	Definition of a character which is to represent the null value. This overwrites any null value representation defined in PARAMETER.
null-value	Specifies the null value character (max. 1 character). The null value representation is specified for the data type CHARACTER or the numeric data types NUMERIC, DECIMAL, INTEGER, SMALLINT, REAL and FLOAT. The character null value representation is also valid for the data types DATE and TIME. The numeric null value representation is also valid for the data type INTERVAL.
COLUMNS n	Specifies the number of columns per list line. COLUMNS may only be specified once within DECLARE FORM. *n* may have the value: $0 < n \leq 255$ Default value: 132

LINES n	Specifies the number of lines per list page. LINES may only be specified once within DECLARE LIST. *n* may have the value: $0 < n \leq 999$ Default value: 60 LINES may have the value: number of lines (TTITLE) + number of lines (BTITLE) <LINES
TTITLE	Defines a list header output with each new page.
BTITLE	Defines a list footer output with each new page.
format	Specifies the *format* of the list form. If TABLE is specified for *format*, NEWLINE is not permitted. If LINE is specified for *format*, TABULATOR and BLANK result in a line feed with a blank line.
expression	Defines output fields for the list form.
mask	Defines the representation options for masked output (= output editing). *mask* may only be specified if *expression* is a simple variable or a simple component.
NEWLINE n	Defines the position of the output fields within the list form. NEWLINE causes an advance of *n* lines. The current line is completed with NEWLINE 1. Any subsequent data is written in the next line. Even if the current line has already been filled, i.e. it already contains as many characters as defined in the COLUMNS specification of the corresponding DECLARE, NEWLINE 1 does not cause a blank line to be output. Instead it positions to the next line. If *n* has the value "0", a conditional line feed is performed, i.e. if the current line is blank, NEWLINE 0 is ignored; if the current line is not blank, NEWLINE 0 has the same effect as NEWLINE 1.
TABULATOR n	Defines the position of the output fields within the list form. The output is continued from column *n*. If the value is less than the current column position, a line or page feed is performed. The resulting gap is filled with blanks. If TABULATOR is specified without a value *n*, either this has no effect or only one line feed is performed. The following applies: 1 < TABULATOR < COLUMNS.

BLANK n

Defines the position of the output fields within the list form. BLANK causes *n* blanks to be inserted. This may result in a line feed or page feed.

n is an integer.

Example

The page header of the list form "staff-list" consists of six lines and the footer of two lines.
The header has the following structure (the current date, time and page number are output in the first line):

```
1995-12-20  14:30:22                                Page:          1


                              Staff
                             ———————
```

The footer consists of a blank line and a line containing 80 equals signs (=).

```
DECLARE VARIABLE &date DATE;
DECLARE VARIABLE &time TIME;
...
DECLARE LIST staff-list
   TTITLE &date,'  ',&time,TAB 60,'Page:',&PAGES,NL 3,
          TAB 31,'Staff',NL 1,
          TAB 30,'-'(13),NL 2
   BTITLE NL 2,'='(80);
```

DECLARE SCREEN
Start editing FHS form

This application is valid

– in TIAM and UTM mode
– in program mode

DECLARE SCREEN defines an FHS form which you have previously created with IFG (see IFG [28]). DECLARE SCREEN must be located ahead of all executable statements in the declaration section of the program.

When compiling DECLARE SCREEN, DRIVE/WINDOWS generates a structured variable from the addressing aid of the screen form, the screen variable. DRIVE/WINDOWS exchanges data with FHS via this screen variable.

The addressing aids for DRIVE programs must be stored for the forms addressed in the form library.

```
DECLARE SCREEN screenform   [ variable ]
                            [ PERMANENT | TEMPORARY ]
                            [ ERRORATTRIBUTE ( attribute, ... ) ]
```

---

screenform — FHS partial form (max. 7 characters).

variable — Name of the variable created by DECLARE SCREEN. The FHS form is copied to this variable (addressing aid). The DISPLAY statement passes this variable to FHS. It is also known as a screen variable. If you do not specify *variable*, the screen variable contains the FHS form name with the prefix "&". *variable* may be a maximum 32 characters in length (including "&").

PERMANENT — The contents of a variable defined with PERMANENT are retained beyond the end of a subprogram and are available when the same subprogram is called again.

TEMPORARY — Default

The contents of a variable defined without PERMANENT are lost once the subprogram terminates.

ERRORATTRIBUTE — Field attributes are assigned to data fields for error handling purposes. These attributes are evaluated during the automated error dialog (see the statements DISPLAY *screenform*, SCREEN-ERROR REPEAT). If you omit this specification, then the specification in PARAMETER DYNAMIC is used.

attribute — Field attribute (see the metavariable *attribute*).

Allocating resources

DRIVE/WINDOWS searches for the forms selected with DECLARE SCREEN in the format library with the file link name FORMOML. If no such file link name is known in a TIAM application, DRIVE/WINDOWS searches for the selected forms in the format library ($userid.)DRI.LIB. The statement PARAMETER STATIC FORMLIB=... may be used, however, to assign a different format library prior to the first processing statement.

Special characteristics in UTM mode

If you want to use FHS forms in UTM mode you must specify the form library in the UTM start procedure:

```
SET-FILE-LINK LINK-NAME=FORMOML,FILE-NAME=form-library    or
PARAMETER STATIC FORMLIB=form-library
```

You must also specify the following UTM start parameters in the UTM start procedure:

```
.FHS DE=NO          if no FHS-DE forms are used    or
.FHS DE=YES         if FHS-DE forms are used
```

and

```
.FHS MAPLIB=form-library
```

Example

The FHS partial form "form" is defined. The associated screen variable is given the name &imagevar.

```
DECLARE SCREEN form &imagevar
```

DECLARE TYPE
Define data type

This application is valid

– in TIAM and UTM mode
– in program mode

DECLARE TYPE defines user-specific data types.

DECLARE TYPE can precede the PROCEDURE statement in a program or can precede all executable statements in the declaration section of the program.
DECLARE TYPE must precede any OPTION statements in programs which use them.

```
DECLARE TYPE { [ level ] user-type data-type }, ...
```

---

level — Specifies a one-digit or two-digit level number.

This defines the hierarchical structure of a data type declaration. The level number need not be specified as long as the data type to be defined is unstructured. The level number is thus used only for data groups and repeating groups.

user-type — A data type defined by the user. It must not exceed 31 characters in length.

data-type — Since the syntax for *data-type* is extremely complex, the following only refers to clauses contained in *data-type*, but which are not clear from the syntax above.

The following must be observed for the individual data types which can be specified for *data-type*.

– Basic data type (for time periods)

  In the case of the data type INTERVAL, *date-time-field1* must be equal to *date-time-field2* in *date-time-unit*.

– Structure type (structured variable)

  structure-type must not contain any LIKE clauses.

– Base type (redefined variable)

base-type must not contain a REDEFINES clause.

In the CHECK clause, the check condition *condition* may not contain a variable. The check condition *condition* may contain the keyword VALUE instead of the variable which is defined in *user-type*.

Examples

The variable &apartment has the user-specific data type "address".

```
DECLARE TYPE     1 address,
                 2 street   CHARACTER (30),
                 2 zip      NUM       (5),
                 2 city     CHARACTER (30);
...
DECLARE VARIABLE &apartment   address;
```

The user-specific data type "db-type" is declared in a program and specified in the USING clause.
DECLARE TYPE follows the OPTION statement and precedes the PROCEDURE statement.

```
OPTION LISTING=LIST;
DECLARE TYPE 1 db-type,
               2 db-system   CHARACTER (3),
               2 function,   CHARACTER (10),
               2 data,
                 3 column    CHARACTER  (20),
                 3 type-name CHARACTER  (15);
PROCEDURE h-prog USING &db-parameter db-type;
...
```

DECLARE VARIABLE
Define variable

This application is valid

– in TIAM and UTM mode
– in program mode

DECLARE VARIABLE is used to define variables. Initial values, check conditions and attributes for displaying data values can be defined for the variables. You also have the option of redefining variables.

DECLARE VARIABLE must precede all executable statements in the declaration section of the program.

```
DECLARE VARIABLE { [ level ] var-name
                 [ PERMANENT | TEMPORARY ]
                 { data-type | LIKE { CURSOR cursor-name | TABLE table } } }, ...
```

---

level — Specifies a one-digit or two-digit level number.

This defines the hierarchical structure of a variable declaration. The level number need not be specified as long as the variable is unstructured. The level number is thus used only for data groups and repeating groups.

var-name — Name of variable.

var-name must be preceded by an ampersand (&) and must not exceed 32 characters in length.

It is possible to define simple variables, vectors, matrices, data groups and repeating groups.
The value range of a variable, excluding its indicator value range must not be greater than 32 Kb.

PERMANENT — Specifies the lifetime of the variable if the program in which the variable is defined is called with CALL.

– Variables are only initialized on the first CALL. These values are retained for subsequent CALLs.
– Variable values are retained after the subprogram has terminated.

TEMPORARY

Default

– Variables are re-initialized on every CALL.
– Variable values are lost after the subprogram has terminated.

data-type

Data type for the variable to be defined. The following data types are available for DRIVE variables:

– Alphanumeric data type
– Numeric data type
– Time data type
– INTERVAL data type
– User-defined data types
– Structured data types

The structured data types include vector, matrix, data group and repeating group. Data group and repeating group are also categorized as "groups" in the DRIVE manuals.

Groups consist of multiple group components, while vectors and matrices consist of a single, simple component and a repetition factor and simple variables consist of a single component only.

The syntax of *data-type* is extremely complex. The following only refers to the specification of clauses for *data-type* (which are not given in the syntax diagram above).

INIT clause.

INIT is used to initialize a variable with a literal or the null value. DRIVE assigns each variable an initial value, even if no INIT clause has been specified (see "Initializing variables with a data type", below).

LIKE clause.

LIKE is used to copy the structure of one variable to another variable (= *variable*) component by component.

CHECK clause

CHECK is used to define a condition that is checked during program execution.

MASK clause

MASK is used to define the attributes for output formatting data types.

	REDEFINES clause REDEFINES is used to specify multiple declarations for a storage area of a variable.
LIKE	LIKE is used to copy the structure of a cursor or table to a variable component by component. The variable must be a data group or a repeating group. The level numbers are adapted during copying. The specifications for the components are of the same data type as the columns in the specified table or cursor. If there are data type differences between the database system and DRIVE/WINDOWS then the data type used in the database system is unambiguously converted into a DRIVE-compatible data type (see Programming Language [2] manual). The components receive the same names as the columns in the table or cursor. If ambiguities or empty strings are present (e.g. in an expression in select-list, see the SQL directories[4-6]), the name "FILLER" is used.
cursor-name	Name of the cursor, which must already be declared (see SQL directories [4-6], DECLARE... CURSOR... statement).
table	Name of a base table or of a persistent or temporary view which must already be declared (see SQL directories [4-6]).

Initializing variables with a data type

If no INIT clause is specified for a variable, the variable is initialized according to its type:

Data type	Default
CHARACTER	Blank (␣)
CHARACTER VARYING, VARCHAR	Empty string
DECIMAL, INTEGER, NUMERIC, SMALLINT, REAL, FLOAT, DOUBLE PRECISION, XDEC, EXTENDED DECIMAL	0 (not the null value!)
DATE	Compilation date if PERMANENT is specified, otherwise execution date
TIME, TIME(3),	Compilation time if PERMANENT is specified, otherwise execution time
TIMESTAMP(3)	Timestamp for compilation time if PERMANENT is specified, otherwise timestamp for execution time
INTERVAL	0 (not the null value!)

Example 1

Definition of simple variables:
The alphanumeric variable "name" has a length of 20 characters and the numeric (packed) variable "number" has a length of 10 characters. &number has 2 decimal places.

```
DECLARE VARIABLE &name CHAR (20);
DECLARE VARIABLE &number NUM (10,2);
```

Example 2

Definition of a vector (= one-dimensional variable):
The vector &language(3) defines the alphanumeric field "language" three times with a length of 10 characters, e.g. for the languages English, French and Spanish.

```
DECLARE VARIABLE &language(3) CHAR (10);
```

Example 3

Definition of a matrix (= two-dimensional variable):
The matrix "&monthly-sales / branch" (12,5) defines the numeric field "monthly-sales / branch" 60 times. The "monthly-sales / branch" field has a length of 12 characters, 2 of which are decimal places and contains the sales figures for one of 5 branches in a single month.

```
DECLARE VARIABLE "&monthly-sales / branch"(12,5) NUM(12,2);
```

The special characters (blanks and slash) in the variable name mean that the name must be enclosed in double quotes ("). The sales figure for August for branch 3 are, for instance, in the variable "&monthly-sales / branch" (08,3).

Example 4

Definition of a data group:

```
DECLARE VARIABLE 1 &staff-member,
                   2 staff-no          CHAR (06),
                   2 last-name         CHAR (20),
                   2 first-name        CHAR (20),
                   2 address,
                     3 country         CHAR (03),
                     3 street          CHAR (26),
                     3 zip             CHAR (10),
                     3 city            CHAR (20),
                   2 salary            NUM  (7,2),
                   2 dept-manager      CHAR (06),
                   2 dept-emp          INT,
                   2 proj-emp          INT;
```

Example 5

Definition of a repeating group:
The repetition factor is to be set to 5.

```
DECLARE VARIABLE 1 &w(5),
                   2 i   INT,
                   2 c   CHAR (3);
```

Example 6

In addition to the data type and length the following items are declared for the components "adept-emp-no", "esalary", "dept-emp-no" and "salary" in the data group "rows":

The initial value 0000 is assigned to the variable &adept-emp-no. If a non-numeric value is assigned to the variable during program execution, the message "Please enter a number" is issued.

If a value less than 20000 is assigned to the variable &esalary during program execution, the message "Salary is too low" is issued.

The variable &dept-emp-no redefines the variable &adept-emp-no.

Values in the variable &salary are displayed without leading zeros, with at least one digit before the decimal point, the decimal point itself, two digits after the decimal point and the string "␣GBP".

```
DECLARE VARIABLE 1 &rows,
                  2 adept-emp-no CHAR (4) INIT '0000'
                    CHECK &adept-emp-no IS NOT NUMERIC
                    MESSAGE 'Please enter a number',
                  2 entry-row,
                    3 elast-name CHAR (20),
                    3 efirst-name CHAR (20),
                    3 esalary NUM (7,2)
                      CHECK &esalary < 20000
                      MESSAGE 'Salary is too low',
                  2 print-row,
                    3 dept-emp-no CHAR (4) REDEFINES &adept-emp-no,
                    3 last-name CHAR (20),
                    3 first-name CHAR (20),
                    3 salary NUM (7,2) MASK 'ZZZZ9P99'' GBP''';
```

DRIVE system variables

DRIVE/WINDOWS provides system variables which can be used in DRIVE programs. You will find a list of these system variables in the DRIVE/WINDOWS Programming Language manual [2].

The scope of the system variables is restricted to the program involved. It is not possible to use system variables to pass information to other, external DRIVE programs, except by means of the USING clause.

DELETE FILE RECORD
Delete record in ISAM file

This application is valid

– in TIAM and UTM mode
– in program mode

DELETE FILE RECORD deletes the data record with the specified ISAM key in an open ISAM file.

```
DELETE FILE RECORD file KEY char-expression
```

---

file	Logical name of a file in which a record is to be deleted. The file must be declared with this name in the program using the DECLARE FILE statement.
char-expression	ISAM key of the record to be deleted.

DISPATCH
Call subprograms concurrently in distributed system

This application is valid

– in UTM mode, in UTM mode without function(v.i)
– in program mode

DISPATCH identifies the start of a dispatch block. The end of this block is defined with END DISPATCH. If a dispatch block has been defined, all CALL statements that call subprograms on a remote system (remote CALL statements) are executed concurrently at END DISPATCH. All other permissible statements in the dispatch block are executed sequentially.

The calling program waits at END DISPATCH until all called subprograms have been executed and then continues with the next statement.

The following statements are not permitted in dispatch blocks:

– BREAK PROCEDURE
– COMMIT WORK
– DISPATCH
– DO
– STOP

Identical RETURN parameters are not permitted in the remote CALL statements of a dispatch block.

If an error occurs while a remote CALL statement is being executed, the calling program is aborted. The WHENEVER statement is not supported.

If variable values have been altered in a DISPATCH block before an incorrect remote CALL statement, the variable values are not reset.

The DISPATCH and END DISPATCH statements are ignored in TIAM mode and the CALL statements are executed sequentially.

The statements BREAK CYCLE, BREAK SUBPROCEDURE, and CONTINUE CYCLE cannot be used to exit DISPATCH blocks.

Dispatch blocks, loops (CYCLE), conditions (IF), and branches must not overlap.

DISPATCH

Example

```
DISPATCH;
  CALL local_proc1          USING RETURN &a;
  CALL remote_proc1         USING RETURN &a;
  SET &b = &a;
  CALL TAC remote_tac       USING RETURN &b;
END DISPATCH;
```

(For an example see the DRIVE Programming Language [2] manual, section Starting concurrent distributed processing via dispatch)

Rules for distributed transaction processing

Remote CALL statements in the same dispatch block must be located in the compilation unit of that dispatch block. This means that if an external subprogram is called locally with CALL within a dispatch block, the external subprogram must not contain any remote CALL statements.

DISPLAY FORM
Define and display compact screen form

This application is valid

- in TIAM mode
- in UTM mode but not in asynchronous UTM applications and not in the receiving environment of distributed transactions
- in program and debugging mode

Used in the program body of a program, DISPLAY FORM defines an ad hoc compact screen form, fills it with its content and outputs it on screen.

The DISPLAY FORM statement allows input to and output from the screen form. It implicitly contains the DECLARE FORM, FILL and DISPLAY statements. This means that you can use just one statement in place of three. The compact screen form is a special type of DRIVE form.

Once the compact screen form has been output, it is not possible to output the contents of the form again since the form cannot be addressed by name.

```
DISPLAY FORM [ format ] { [ RETURN ] expression[ INIT expression1[ NOCHECK ] ]
                          [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |
                        NEWLINE n |
                        TABULATOR n |
                        BLANK n }, ...

            [ { COLUMNS n | LINES n }, ... ]

            [ TTITLE [ format ] { [RETURN] expression[INIT expression1 [NOCHECK] ]
                                  [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |
                                NEWLINE n |
                                TABULATOR n |
                                BLANK n }, ... ]

            [ BTITLE [ format ] { [RETURN] expression [INIT expression1 [NOCHECK] ]
                                  [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |
                                NEWLINE n |
                                TABULATOR n |
                                BLANK n }, ... ]
```

---

format	Specifies the format of the screen form (see the metavariable *format*). If TABLE is specified for *format*, NEWLINE must not be specified. If LINE is specified for *format*, TABULATOR and BLANK result in a line feed with a blank line.
RETURN	Specifies that a variable becomes an entry field, which may be preset. A variable for which RETURN has been specified may only be used as an entry field once for each form. If an input variable (with RETURN) does not fit on one screen page, the program is aborted when the statement is executed. If an output variable (without RETURN) does not fit on one screen page, it is truncated for the output. The last three characters of the output variable are represented by ">>>". If an entry or output field in TTITLE or BTITLE does not fit into the space allocated to it, the procedure is aborted when the statement is executed. RETURN is ignored in debugging mode.
expression	Defines output and/or entry fields for the screen form (see the metavariable *expression*). *expression* may not exceed 31 Kbytes in length. When RETURN or INIT is specified, *expression* must be a variable that may not be qualified with ".*".
INIT	Assigns an initial value to *expression*. The INIT clause is permitted only if *expression* is a variable. If *expression* is a vector or a matrix, all components are given the corresponding initial value *literal* or NULL.
expression1	*expression1* may only be *literal*, NULL or a function which possesses literal arguments (but not CURRENT DATE/TIME/TIMESTAMP). *expression1* must be calculable at compilation time.
NOCHECK	A CHECK clause (see the metavariable *check*) specified when declaring the *expression* variable is not evaluated for the assignment of the initial value. NOCHECK is not permitted for redefined variables or variables which redefine other variables.
ATTRIBUTE	Assigns field attributes to screen forms. ATTRIBUTE is ignored in debugging mode.

attribute	Field attribute (see the metavariable *attribute*). Only the following color attributes are permitted: GREEN, RED, WHITE and YELLOW. Output fields are preset with the following values: VISIBLE, PROTECTED, NOUNDERLINE, NORMALINTENSITY. Input fields are preset with the following values: VISIBLE, UNPROTECTED, NOUNDERLINE, HIGHINTENSITY. INVISIBLE can only be specified for input fields (with RETURN).
mask	Defines the representation options for masked input and output (= output editing). *mask* may only be specified if *expression* is a simple variable or a simple component.
NEWLINE n	Defines the position of the screen fields. NEWLINE causes an advance of *n* lines. The current line is completed with NEWLINE 1. Any subsequent data is written in the next line. Even if the current line has already been filled, i.e. it already contains as many characters as defined in the COLUMNS specification of the corresponding DECLARE, NEWLINE 1 does not cause a blank line to be output. Instead it positions to the next line. If *n* has the value "0", a conditional line feed is performed, i.e. if the current line is blank, NEWLINE 0 is ignored; if the current line is not blank, NEWLINE 0 has the same effect as NEWLINE 1.
TABULATOR n	Defines the position of the screen fields. The output is continued from column *n*. If the value is less than the current column position, a line or page feed is performed. The resulting gap is filled with blanks. If TABULATOR is specified without a value *n*, either this has no effect or only one line feed is performed. The following applies: 1 < TABULATOR < COLUMNS
BLANK n	Defines the position of the screen fields. BLANK causes *n* blanks to be inserted. This may result in a line feed or page feed. *n* is an integer.

COLUMNS n	Defines the number of columns per screen. You can only specify COLUMNS once within the DISPLAY FORM statement. *n* can have the following value: 0 < *n* ≤ number of screen columns. Default: number of columns in current screen.
LINES n	Defines the number of lines per screen. You can only specify LINES once within the DISPLAY FORM statement. *n* may have the following value: 0 < *n* ≤ number of screen lines - 1 Default: number of lines in current screen - 1 The last line is reserved as a message line. LINES may have the following value: number of lines (TTITLE) + number of lines (BTITLE) < LINES.
TTITLE	Defines a page header to be output with each new screen. The total number of lines for TTITLE, BTITLE and FILL must not exceed: the number of screen lines - 1. The last line is reserved for use as a message line. There must always be at least one FILL line. In the case of a screen overflow, TTITLE is output as the page header.
BTITLE	Defines a page footer to be output with each new screen. The number of lines that can be defined for a page footer is the same as the number of lines defined under TTITLE. In the case of screen overflow, BTITLE is output as a page footer.

Masking CHAR expressions in the NUM function only has the effect that the CHAR expression is checked on entry with regard to permissibility in the mask, i.e. the mask has no effect on the output.

Relations to other statements

– Screen output is determined by assigning structured variables (see DECLARE SCREEN statement). You may, for example, use SET to perform this variable assignment (see SET statement).

Example

The compact screen form is empty up to line 15. The 15th line contains the text "Enter serial number:" indented by five characters.

```
DECLARE VARIABLE &number NUM(3);
...
DISPLAY FORM NL 15, TAB 5,'Enter serial number: ',RETURN &number;
```

DISPLAY form-name
Display DRIVE form

This application is valid

– in TIAM mode
– in UTM mode but not in asynchronous UTM applications and not in the receiving environment of distributed transactions
– in program mode

DISPLAY *form-name* is used to conclude DRIVE form editing and display the form on the screen. The memory area with the DRIVE screen form is not released following display. Thus, a form can be displayed again and again if no new FILL statement is issued for that memory area. When a FILL statement is specified for that memory area, form memory is cleared and new contents generated.

DISPLAY *form-name* is not permitted in asynchronous UTM applications and in the receiving environment of distributed transactions.

```
DISPLAY form-name
```

---

form-name — Name of the DRIVE form (max. 31 characters).

The specified form must be defined in the declaration section of the program using DECLARE FORM.
form-name may be specified only once.

DISPLAY LIST
Define and output compact list form

This application is valid

– in TIAM and UTM mode
– in program and debugging mode

Used in the program body of a program, DISPLAY LIST defines an ad hoc compact list form, fills it with its content and outputs it at the printer.

The DISPLAY LIST statement implicitly contains the DECLARE LIST, FILL and DISPLAY statements. This means that you can use just one statement in place of three. The compact list form is a special type of DRIVE form.

Once the compact list form has been output, it is not possible to output the contents of the form again since the form cannot be addressed by name

If you use this statement under UTM you must generate the central print file. There are three ways of generating this file:

– the `DRI.LIST.FILE` file may already be present,
– the file may be assigned using the file link name `DRILIST`,
– the file `DRI.LIST.FILE` may be generated by DRIVE/WINDOWS on the first DISPLAY statement.

Actual output to the printer is performed on the next STOP (see STOP statement) or LIST (see LIST statement).

```
DISPLAY LIST [ format ] { expression [ mask ] |
                        NEWLINE n |
                        TABULATOR n |
                        BLANK n }, ...

            [ { COLUMNS n | LINES n }, ... ]

            [ TTITLE [ format ] { expression [ mask ] |
                                  NEWLINE n |
                                  TABULATOR n |
                                  BLANK n }, ... ]

            [ BTITLE [ format ] { expression [ mask ] |
                                  NEWLINE n |
                                  TABULATOR n |
                                  BLANK n }, ... ]
```

format — Specifies the format of the list form (see the metavariable *format*).

If TABLE is specified for *format*, NEWLINE must not be specified.

If LINE is specified for *format*, TABULATOR and BLANK result in a line feed with a blank line.

expression — Defines output fields for the list form.

mask — Defines the representation options for masked output (= output editing).
mask may only be specified if *expression* is a simple variable or a simple component.

NEWLINE n — Defines the position of the output fields within the list form.

NEWLINE causes an advance of *n* lines. The current line is completed with NEWLINE 1. Any subsequent data is written in the next line.

Even if the current line has already been filled, i.e. it already contains as many characters as defined in the COLUMNS specification of the corresponding DECLARE, NEWLINE 1 does not cause a blank line to be output. Instead it positions to the next line.

If *n* has the value "0", a conditional line feed is performed, i.e. the current line is blank, NEWLINE 0 is ignored; if the current line is not blank, NEWLINE 0 has the same effect as NEWLINE 1.

TABULATOR n	Defines the position of the output fields within the list form. The output is continued from column *n*. If the value is less than the current column position, a line or page feed is performed. The resulting gap is filled with blanks. If TABULATOR is specified without a value *n*, either this has no effect or only one line feed is performed. The following applies: 1 ≤ TABULATOR ≤ COLUMNS
BLANK n	Defines the position of the output fields within the list form. BLANK causes *n* blanks to be inserted. This may result in a line feed or page feed. *n* is an integer.
COLUMNS n	Defines the number of columns per list line. You may only specify COLUMNS once within the DISPLAY LIST statement. *n* may have the following value: 0 < *n*≤ 255 Default: 132
LINES n	Defines the number of lines per list page. You may only specify LINES once within the DISPLAY LIST statement. *n* may have the following value: 0 < *n* ≤ 999 Default: 60 Lines may have the following value: number of lines (TTITLE) + number of lines (BTITLE) < LINES
TTITLE	Defines a list header to be output with each new page.
BTITLE	Defines a list footer to be output with each new page.

Example

The compact list form has the following structure: the second line contains the current date, the current time and the current page number. The eleventh line contains the data contents of the variables &surname, &firstname, &salary.

```
1995-12-20  15:03:42                                        Page:              1


                              Staff
                             ------




   Winterberg                Abigail                   3500.00 US$
```

```
DISPLAY LIST
   NL 1,&date,'  ',&time,TAB 60,'Page: ',&PAGES,NL 3,
   TAB 31,'Staff',NL 1,
   TAB 30,'-'(13),NL 5,
   TAB 5,&surname,TAB 30,&firstname,TAB 55,&salary,' US$';
```

DISPLAY list-name
Output list form

This application is valid

– in TIAM and UTM mode
– in program mode

The DISPLAY *list-name* statement is used to complete editing a DRIVE list form and output it to a printer.

The memory area containing the DRIVE list form is not released following output. Thus, a form can be output again and again if no new FILL statement is issued for that memory area. When a FILL statement is specified for that memory area, form memory is cleared and a new form is generated.

If you use this statement under UTM you must generate the central print file. There are three ways of generating this file:

– the DRI.LIST.FILE file may already be present,
– the file may be assigned using the file link name DRILIST,
– the file DRI.LIST.FILE may be generated by DRIVE/WINDOWS on the first DISPLAY statement.

The actual output on the printer occurs at the next STOP or LIST (see the LIST and STOP statements).

```
DISPLAY list-name
```

---

list-name	Name of the DRIVE list form (max. 31 characters). The specified list form must be defined in the declaration section of the program using DECLARE LIST.

DISPLAY screenform
Output FHS form

This application is valid

– in TIAM mode only in the case of FHS without dialog extension (DE)
– in UTM mode but not in asynchronous UTM applications and not in the receiving environment of distributed transactions
– in program mode

DISPLAY *screenform* outputs an FHS form which you have previously created with IFG (see IFG [28]). You can output multiple partial forms simultaneously. If simultaneously outputting multiple partial forms, you may not mix FHS forms for dialog extension (FHS-DE) with simple FHS forms.

When a screen form is output, the screen variable, together with any available input values, is passed to FHS and then output on the screen.

The screen variable is the structured variable which is created using DECLARE SCREEN in order to receive the addressing aid of an FHS form.

The contents of the screen variable are not deleted after output. Forms may not overlap on screen.

```
DISPLAY screenform, ...
        [ SCREENERROR { REPEAT | CONTINUE } ]
        [ CURSOR { POSITION ( line1, column1 ) | TO field1 }
        [ MESSAGE key [ POSITION ( line2, column2 ) | TO field2 ]
```

---

screenform

Name of FHS partial form (max. 7 characters).

The specified form must be defined in the declaration section of the program using DECLARE SCREEN.

If you want to output multiple FHS partial forms simultaneously, you may either only output forms with dialog extension (DE) or only forms without dialog extension.

In the case of automatic error dialog, the last line must not be used since DRIVE/WINDOWS uses this line to output messages.

In the case of a user-controlled error dialog, the last line may be used, e.g. for the output of user-defined error messages.

SCREENERROR — Specifies the behavior of DRIVE/WINDOWS in the case of invalid field input.

REPEAT — Default value

The partial form is repeatedly output until the input is correct or aborted by means of BREAK (= automated error dialog). Incorrect input values are all fields which do not satisfy the CHECK clause (see the DECLARE VARIABLE statement.

The incorrect input values are indicated with the ERRORATTRIBUTE operand defined in PARAMETER DYNAMIC or DECLARE SCREEN (see the PARAMETER DYNAMIC and DECLARE SCREEN statements).

The data of the individual partial forms is not transferred from FHS-FORM format to the screen variable until input to all the screen fields is correct.

CONTINUE — The program is continued after DISPLAY (= user-controlled error dialog). Invalid input causes the program to abort if no suitable WHENEVER statement has been specified, and the &ERROR_STATE system variable receives the following values:

&ERROR='FORMAT_ERROR'
&FORMAT_NAME=name of the first invalid partial form
&VAR_NAME=name of the first invalid entry field

If the program is aborted by illegal use of a K/F key, the &ERROR_STATE system variable receives the following values:

&ERROR='FORMAT_ERROR'
&FORMAT_NAME= "%K/F_ERROR"
&VAR_NAME=INIT value of DECLARE VARIABLE

Invalid input values can be marked with SET ERRORATTRIBUTE and the invalid forms redisplayed. Invalid input values are all fields which do not satisfy the CHECK clause.

All valid fields are passed to the structured screen variable. The other components are retained.

CONTINUE is used only if the CHECK clauses are not satisfied.

CURSOR — The cursor is set to a specified position on the screen.

You may not specify CURSOR unless the partial form to be output is an FHS-DE form and the global attribute “Cursor position” was set for *screenform* in IFG (see IFG [28]).

POSITION	Specifies the absolute position (line/column) of the cursor.
line1	Line (1 ≤ *line2* ≤ number of screen lines). *line2* must be a whole number.
column1	Column (1 ≤ *column2* ≤ number of screen columns). *column2* must be a whole number.
TO	The cursor is set at the first character of field *field1*. In the case of lists, the cursor is set to the first column and first line of the list area.
field1	Field in the partial form which is to be output. *field1* must be a component of the screen variable for *screenform*. The first 8 characters of the field names of all screen forms output using DISPLAY screenform must differ to make the unambiguous assignment of screen variable components possible.
MESSAGE	This outputs the FHS-DE message with the message key *key* which you created with IFG. Depending on the IFG specification, output either takes place in a message box or in a message area in the partial form *screenform*. You may not specify MESSAGE unless the partial form to be output is an FHS-DE form and the global attribute “Message identifier” was set for *screenform* in IFG (see IFG [28]).
key	Message key of the FHS-DE message. You can specify *key* either as a variable (see the metavariable *variable*) or as an alphanumeric literal (see *char-literal* in the metavariable *literal*). *key* must be specified in the form AAAAnnn, where A is a letter (A-Z) and n is a digit (0-9). AAAA may not have the value IDHS.
POSITION	Specifies the absolute position of the message box. The message box is positioned with an additional offset (+2,+2) to *line3*, *column3*. POSITION is only evaluated if the IFG specification stipulates that the message is to be output in a message box. If a message is intended for output in a message box and you have not specified either POSITION or TO then the message box is output in the middle of the screen. If a message box which has been positioned using MESSAGE POSITION covers a cursor which has been set with CURSOR POSITION then MESSAGE POSITION is ignored.
line2	Line (1 ≤ *line3* ≤ number of screen lines). *line3* must be a whole number.

column2
: Column ($1 \leq column3 \leq$ number of screen columns). *column3* must be a whole number.

TO
: Specifies that the message box is to positioned with the default offset (+2,+2) to *field2*.

  TO is only evaluated if the IFG specification stipulates that the message is to be output in a message box.

  If a message is intended for output in a message box and you have not specified either TO or POSITION then the message box is output in the middle of the screen.

field2
: Field in the partial form which is to be output. *field2* must be a component of the screen variable for *screenform*.

  The first 8 characters of the field names of all screen forms output using DISPLAY screenform must differ to make the unambiguous assignment of screen variable components possible.

Relationships to other statements

Screen output is determined by assigning structured variables (see DECLARE SCREEN statement). You may, for example, use SET to perform this variable assignment (see SET statement).

Example

The IFG/FHS partial form "form" is output. If input is incorrect, the form is output again.

```
DISPLAY form SCREENERROR REPEAT
```

DO
Start interactive program

This application is valid

– in TIAM mode
– in UTM mode but not in asynchronous UTM applications and not in the receiving environment of distributed transactions
– in interactive mode
– in program mode only within an interactive program

DO has two functions depending on the mode:

– If you specify DO in interactive mode, an implicit COMPILE is performed and the program is then started.

  If intermediate code has already been generated for the program and stored in the current DRIVE library, DO accesses it directly and simply starts the program. Syntax and semantic checking are omitted in this case.

  DRIVE searches first for intermediate code under the specified name and only then searches for a source program.

– If DO is specified within an interactive program, the program is aborted, and the successor program called. This DO has the same effect as an END PROCEDURE followed by DO in interactive mode. DO may not be specified so long as a transaction is still open or screen output has not yet been concluded.

If the runtime system is used, a program can only be called with the DO statement if it is available as intermediate code.

```
DO [ library(member-name) | member-name ]

   [ USING { expression | NULL }, ... ]
```

---

library — Specifies the DRIVE library (max. 54 characters) from which the DRIVE program is read.

library may also be the file link name of the DRIVE library (in accordance with BS2000 conventions).

DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name.

	If the DRIVE library has been preset using the statement PARAMETER DYNAMIC LIBRARY then you do not need to specify *library*.
member-name	Name of the member (max. 31 characters) which contains the DRIVE program as a source or intermediate code. DRIVE/WINDOWS searches for the last member to have been processed, irrespective of whether this contains a source (S-member) or an intermediate code (X-member). *member-name* can be an old style program which was created using DRIVE V5.1. Please refer to the DRIVE Programming Language [2] manual for notes on calling old style programs. If you do not specify *library*, the library which has been preset in PARAMETER DYNAMIC LIBRARY is used. An error message is issued if *member-name* is not present in the specified library. You must specify *member-name* in UTM and program mode. If you do not specify *member-name* then DRIVE/WINDOWS uses the source which is present in EDT work file 0 (in interactive mode only).
USING	Allows parameters to be passed to the interactive or successor program to be started. These programs must have been defined with PROCEDURE...USING..., and the parameters must be compatible (see the "DRIVE Programming Language" manual [2]). In debugging mode, the user is prompted for parameters if this is not the case (see the "DRIVE Programming Language" manual [2]).
expression	Specifies the parameters to be passed (send fields). In interactive mode, only literals, aggregates whose components are literals and arithmetic expressions without variables are permitted as transfer parameters. Values of variables can also be passed in interactive programs.
NULL	Transfers the null value to the interactive or successor program to be started.

Displaying syntax and runtime errors

If a program is started with DO then an error list is generated if runtime errors occur. In TIAM mode this list is written to SYSLST and in UTM mode it is written to the central print file.

No error list is generated if an error occurs when a member from EDT work file 0 is executed. Instead the error messages are inserted in the member from EDT work file 0 when the next EDT statement is processed (see EDT statement).

If errors occur in external subprograms started with CALL then an error list is generated even if the calling main program is located in EDT work file 0.

Relationship to other statements

– DO statements have no effect within transaction-driven programs which are started as asynchronous UTM conversations. Instead you may use ENTER statements in such cases.

– If you do not specify either *library* or *member-name* in the DO statement then the source OPTION specifications LISTING=LIBRARY and CODE=ON are not executed and are skipped without any comment.

– If an analysis phase is performed with DO, the compiler option CODE=ON specified in the source program is ignored.

  Other options are executed, e.g. if LISTING=LIBRARY is specified, the compiler list is written to a library member.

– If the program is compiled with compiler option OPTION OBJECT=ON then you may not specify *library* when the program is called.

Access rules for databases

– The DO statement is only executed if no (new style) transactions are open in interactive mode.

– If different database systems are assigned to interactive mode and the subprogram called (DBSYSTEM ≠ OFF), then the DO statement is aborted. You can only assign different database systems if DO calls intermediate code or object code which was generated using another database system in a previous DRIVE session.

– If a database system is assigned to interactive mode (DBSYSTEM ≠ OFF) and not to the called subprogram (DBSYSTEM = OFF) then the called subprogram accesses the same database system as interactive mode.

– If a BS2000 database system is assigned to the called program (DBSYSTEM = UDS / SESAM / SESAMSQL), then the DO statement is only executed if this database system matches the loaded variant.

– If the called subprogram is an old style program then the DO statement is aborted if the UDS database system is assigned to interactive mode (DBSYSTEM = UDS).

If an interactive program contains the DO statement, you should observe the rules relating to END PROCEDURE in programs which are called using DO (see section “Access rules for databases” on page 107)

Rules for distributed transaction processing

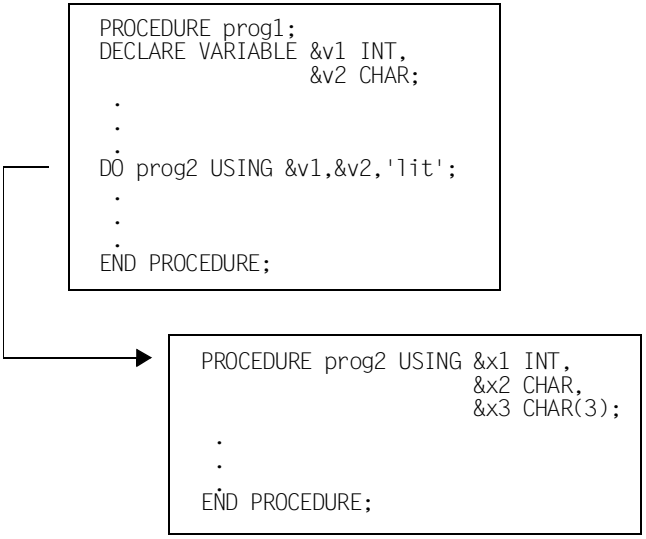
– DO in an interactive program is only permitted in the submitting partner environment.

– DO may not occur in DISPATCH blocks.

– DO cannot be executed until all receiving partner conversations have been terminated.

Example

The program "prog1" passes the parameters &v1, &v2 and the literal "lit" to the program "prog2". For each of these parameters, a comparable parameter (&x1, &x2, &x3) must have been defined in "prog2".



&x1 contains the value of &v1.
&x2 contains the value of &v2.
&x3 contains the value 'lit'.

EDT
Call editor

This application is valid

– in TIAM mode
– in interactive mode

EDT calls the standard BS2000 editor EDT.

If you specify EDT together with operands, a source is loaded in F mode into the EDT work file 0. This source may have a maximum of 999 999 lines each consisting of up to 256 characters.

If you specify EDT without operands, processing branches to the EDT work file 0. This file is empty if no file has yet been loaded into EDT during the current DRIVE session. If a file has already been loaded then it is this file that is loaded into EDT work file 0.

```
EDT [ library(member-name) | member-name ]

    [ SOURCE | LIST | COPYSOURCE | USERLABEL ]
```

---

library	Specifies the DRIVE library (max. 54 characters) from which the member with the specified *member-name* is read. *library* may also be the file link name of the DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name. If the DRIVE library is preset using the PARAMETER DYNAMIC LIBRARY statement, you may omit the *library* specification.
member-name	Name of the member which is to be read (max. 31 characters). If you do not specify *library,* the library specified in PARAMETER DYNAMIC LIBRARY is used. If you do not specify *member-name* then DRIVE/WINDOWS uses the source which is present in EDT work file 0 (in interactive mode only).
SOURCE	Default The member which is to be read is a source (S-member) in the DRIVE library.

LIST	The member which is to be read is an interpreter listing (P-member) in the DRIVE library. LIST must be specified when reading the interpreter listing into the editor, because otherwise a source program with the name of the interpreter listing would be searched for. The interpreter listing should not be modified, because otherwise program statements cannot be traced, or error messages referring to line numbers in interpreter listings will not be correct.
COPYSOURCE	The member which is to be read is a COPY member (S-member) in the DRIVE library. You must specify COPYSOURCE when you read a COPY member into the editor as the system otherwise searches for a source with the same name as the COPY member.
USERLABEL	The member which is to be read is a user label (S-member) in the DRIVE library. USERLABEL must be specified when reading the user label into the editor, because otherwise a source program with the name of the user label would be searched for.

BS2000 standard editor EDT:

Rules

The end of a line is a delimiter, i.e. keywords, names and operators may not extend beyond one EDT line.

Exception:

alphanumeric, hexadecimal literals, comments and all strings enclosed in quotes (").

Restrictions on the use of EDT functions

– You may not enter the following EDT statements:

  @EDIT
  @EXEC
  @LOAD
  @RUN
  @SYSTEM

– EDT marks 1-4 can be used as required. In contrast, EDT marks 5-9 may not be used. They are reserved for internal DRIVE use.

– The EDT work file 0 and EDT work files 1-8 can be used as required. The EDT work file 9 is reserved for DRIVE/WINDOWS. This is where the interpreter listing is written following DO and COMPILE if no *library(member-name)* is specified.

Prompting to protect against accidental overwriting

The contents of EDT work file 0 that have not yet been stored with SAVE are protected by DRIVE/WINDOWS against accidental overwriting. For example, if a new member is to be read into EDT work file 0 with EDT, DRIVE/WINDOWS displays the message
`DRI0046 OVERWRITE' member-name'? REPLY: (Y=YES, N=NO).`

If the response is "Y", the member in EDT work file 0 is overwritten with the new member.

If the response is "N", the old member is retained, and no branch is made to EDT. The source program can be saved with the SAVE statement or viewed with the EDT statement (without operands).

Indication of errors in DRIVE programs (max. 4000 errors)

– DO and COMPILE applied to programs contained in EDT work file 0:

  If DRIVE/WINDOWS finds syntax errors during analysis, the error messages are inserted into the program contained in EDT work file 0 at the next EDT statement. In addition, EDT work file 9 contains the complete interpreter listing.
  EDT work file 9 should not be used by the user, since the contents of work file 9 are overwritten without warning.

  EDT lines should not exceed 230 characters in length. Longer lines cannot be completely shown in the interpreter listing.

– If EDT is activated after analyzing a DRIVE program that contains errors, DRIVE/WINDOWS will position EDT work file 0 to the line containing the first error and additionally mark all program lines containing errors with EDT mark 5. It is thus possible to move to each successive invalid program line by entering EDT +(5) and pressing the F3 key.

– DRIVE/WINDOWS inserts a line after each program line containing an error and indicates the respective error positions by asterisks (*) in the line. The line is marked with EDT mark 13.
  Additional lines containing error messages for the programming errors in question are inserted after each line showing the positions of the errors. These lines are also marked with EDT mark 13.

– Up to 48 error messages are shown per EDT line.

– If you switch back to DRIVE interactive mode via the EDT statement HALT or RETURN, all lines with EDT mark 13 that have not been modified are deleted (i.e. all lines containing error positions and messages which were not overwritten with the same or new contents). All EDT marks are cleared.

  On the other hand, if you exit EDT with the K1 key, the lines showing the positions of errors and the lines containing error messages are retained. EDT marks 5 and 13 remain set. The errors are redisplayed when EDT is called again.

  If the EDT statement DELETE MARK is entered, the marks set by DRIVE/WINDOWS are also deleted.

– Behavior in the case of an error in a copy member.

  The error message is inserted into the source program following the copy member involved. Instead of the line containing asterisks, a message appears. This message indicates that the error has occurred in a copy member and shows in which line of the copy member the error can be found.

  The interpreter listing in EDT work file 9 contains the expanded copy member and the exact location of the error.

Relationship to other DRIVE statements

A DRIVE program in EDT work file 0 can be stored with SAVE. Entering SAVE causes the current contents of EDT work file 0 to be stored in the current DRIVE library.

END
Mark end of logical program unit

This application is valid

– in TIAM and UTM mode
– in program mode

END marks the end of a logical program unit. A program unit may be: a branch (CASE), a loop (CYCLE), a condition (IF), concurrent remote processing (DISPATCH), report generation (REPORT), internal subprograms (SUBPROCEDURE) or complete programs (PROCEDURE).

```
END { CASE | CYCLE | DISPATCH | IF | PROCEDURE | REPORT | SUBPROCEDURE }
```

CASE	End of a branch.
CYCLE	End of a loop.
DISPATCH	End of a DISPATCH block. All CALL statements in the dispatch block that call subprograms on a remote system (remote CALL statements) are executed. The calling program is interrupted until all remote requests have been completed.
IF	End of a condition.
PROCEDURE	End of a program. END PROCEDURE must be the last statement in the program, and may occur only once in the program. The program is terminated normally. DRIVE/WINDOWS displays the message `DRI0088 'program-name' TERMINATED NORMALLY.` If the program was called with CALL, control returns to the calling program, and processing continues with the statement following the CALL. The RETURN parameter values are passed to the calling program. If you use END PROCEDURE to end a program which was called with CALL, all open transactions and all files opened with the OPEN FILE statement remain open.

	If the program was called with DO, DRIVE/WINDOWS switches to interactive mode. All the necessary system resources are freed. If a transaction is open, the program is aborted with an error message and the transaction is reset. Any files opened with the OPEN FILE statement are closed. It is not possible to restart the program.
REPORT	End of reporting.
SUBPROCEDURE	End of an internal subprogram. END SUBPROCEDURE is always the last statement in a subprogram. The calling program is continued immediately following the CALL to the internal subprogram. Open loops (CYCLE without END CYCLE), open conditions (IF without END IF), open DISPATCH blocks (DISPATCh without END DISPATCH), open branches (CASE without END CASE), are not permitted within an internal subprogram.

During error analysis, an incorrect END can cause any number of subsequent errors as other possibly correct END statements are then reported as incorrect or missing (e.g. if an IF structure within a loop is terminated with END IF before END CYCLE).

Relationship to other DRIVE statements (applies only to main programs)

Output areas for forms set up with FILL FORM/LIST must be closed with DISPLAY FORM/LIST before END PROCEDURE is reached.

Relationship to other DRIVE statements (applies only to subprograms called with CALL)

Depending on whether output areas for forms were declared as PERMANENT or TEMPORARY, they must be closed with DISPLAY FORM/LIST before END PROCEDURE is reached.

Rules for distributed transaction processing

– The following applies to programs which were started using DO in the submitting partner environment: All receiving partner programs which have been started in the interactive program must have terminated when END PROCEDURE is issued.

– When END PROCEDURE is issued for a DRIVE program in the receiving partner environment which was called directly by the submitting partner activity, processing returns to the submitting partner activity. RETURN parameters are passed to the submitting partner activity.

Access rules for databases

The following applies to END PROCEDURE in subprograms which were called with CALL in the local system:

– If a database system is assigned to the calling program (DBSYSTEM ≠ OFF) and not to the subprogram called (DBSYSTEM = OFF), the calling program accesses the same database system as the subprogram called after the subprogram called has executed.

The following applies to END PROCEDURE in programs which were called with DO or DEBUG:

– If any transaction is still open, DRIVE/WINDOWS resets it and issues message DRI0101.

– If any temporary SQL objects which have been defined in program mode (program cursor or temporary views) are present, they are deleted by DRIVE/WINDOWS. DRIVE/WINDOWS issues the message DRI0150 if it is unable to delete an SQL object.

– If dynamic, temporary views are present when SESAM V2.x is accessed, DRIVE/WINDOWS deletes them and issues message DRI0488.

– When accessing SESAM V2.x, DRIVE/WINDOWS issues a SET SESSION, SET CATALOG and SET SCHEMA statement. The operands for these statements are defined in the previous PARAMETER DYNAMIC AUTHORIZATION, PARAMETER DYNAMIC CATALOG and PARAMETER DYNAMIC SCHEMA statements respectively.

ENTER
Start program as asynchronous UTM conversation

This application is valid

– in UTM mode

– in interactive and program mode

In UTM mode, ENTER initiates the asynchronous execution of a program or a user-specific program unit in local or remote UTM applications. ENTER statements which call programs in remote systems are known as remote ENTER statements.

If a DRIVE program is called with ENTER, an implicit COMPILE is performed by DRIVE/WINDOWS before starting the program.

If intermediate code has already been generated for this program and stored in the current DRIVE library, ENTER accesses it directly and simply starts the program. Syntax and semantic checking are omitted in this case. DRIVE/WINDOWS searches for the last program to have been processed, irrespective of whether this is present as a source (S-member) or an intermediate code (X-member).

If ENTER is a statement within a program started with ENTER, any subsequent program is started as an asynchronous UTM conversation. The calling program continues to execute normally.

The following applies to calling DRIVE programs:

– Options specified for the initiating asynchronous program (e.g. via PARAMETER) are passed to successor programs in the case of local execution, but not for remote execution.

– If errors occur within a program which was started with ENTER, the program is aborted. An error list and the message `DRI0087 ERROR program-name ABORTED` are written to the central print file (see DRIVE Programming System [1] manual).

In addition, a user ID can be specified with PERMIT, and transaction conditions can be defined with SET TRANSACTION.

```
ENTER { library(member-name) |
          member-name |
        [ COBOL | C ] TAC tacname }

      [ USING { expression | NULL [ INDICATOR ] }, ... ]
      [ PERMIT ]
      [ SET TRANSACTION ]
```

library
: Name of the DRIVE library (max. 54 characters) in which the DRIVE program is read.

  library may also be the file link name of the DRIVE library (in accordance with BS2000 conventions).

  DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name.

  If the DRIVE library has been predefined with the PARAMETER DYNAMIC LIBRARY statement, *library* need not be specified.

member-name
: Name of the member (max. 31 characters) which contains the DRIVE program as a source or intermediate code.

  DRIVE/WINDOWS searches for the last member to have been processed, irrespective of whether this contains a source (S-member) or an intermediate code (X-member).

  In interactive mode, the system checks whether the PARAMETER DISTRIBUTION statement was used to define distribution information. In program mode, the system only checks for distribution information if the compiler option OPTION DISTRIBUTION=ON is defined.

  A local or remote program is called in accordance with the distribution information. If no distribution information has been defined with the PARAMETER DISTRIBUTION statement, the system searches for the program locally.

  If you do not make a *library* specification, then the system uses the library which has been preset for the (local or remote) system using PARAMETER DYNAMIC LIBRARY.

COBOL
: Default.

  Language option for calling a subprogram written in COBOL.

C	Language option for calling a program written in C.
TAC	If ENTER TAC is specified, *permit* and *set transaction* (see below) are not permitted. In interactive mode, a check is made to determine whether distribution information was specified with the PARAMETER DISTRIBUTION statement. In program mode, the distribution information is only checked if the compiler option OPTION DISTRIBUTION=ON was defined. Depending on the distribution information, either a local or a remote program is called. If no distribution information was defined with the PARAMETER DISTRIBUTION statement, the program is searched for locally. The number of variables passed must be limited so that the total length of all data values and descriptions does not exceed 31 Kbytes.
tacname	Name of the transaction code (max. 8 characters) of a UTM user program unit. The UTM program unit may be part of the local or a remote UTM application. The UTM application can run under SINIX or BS2000. The prefixes `dri`, `drt`, `drc` and `sql` are not permitted in *tacname*. These names are reserved for DRIVE/WINDOWS. If a UTM user program unit is called, the programming language in which it is written must be specified.
USING	Enables parameters to be passed from the calling program to the called program. The parameters must be compatible (see the "DRIVE Programming Language" manual [2]). If the called program is a DRIVE program, it must have been defined with PROCEDURE...USING... . If the called program is a UTM program unit, the passed parameters are placed in the message area. The UTM statement MGET requests the passed data from UTM for the program unit. The USING clause must not be used when calling a UTM program unit (ENTER TAC) in interactive mode.

expression	Specifies the parameters to be passed (send fields). In program mode, it is possible to pass variables, vectors, matrixes, aggregates, literals or arithmetic expressions to external subprograms in DRIVE (ENTER *library(member-name)* ..., ENTER *member-name* ...). Simple variables, vectors, matrixes, literals, or arithmetic expressions may be passed to external DRIVE subprograms on the remote system. In interactive mode, literals, aggregates composed of literals, and arithmetic expressions which do not contain any variables may be passed to external subprograms in DRIVE. Simple variables, vectors, matrixes, or components of structured variables may be passed to UTM program units (ENTER TAC). *expression* must be of type *variable*. The total length of all data values and descriptions passed must not exceed 31 Kbytes. If the value of *expression* is NULL in an ENTER TAC, the INDICATOR clause must be specified; otherwise, the program is aborted.
NULL	The NULL value is passed to the program which is to be started.
INDICATOR	INDICATOR is used to create an indicator variable. The value of the indicator variable specifies whether the transfer parameter contains the null value or a defined value. The INDICATOR entry is only permitted for ENTER TAC.
PERMIT	PERMIT statement. See the description of the PERMIT statement in the SQL directories [4], [5] and [6]. The PERMIT statement is evaluated when DRIVE programs are called in the local system and ignored when they are called in the remote system. The PERMIT statement is not permitted when calling UTM program units.
SET TRANSACTION	SET TRANSACTION statement. See the SET TRANSACTION statement in the SQL directories [4], [5] and [6]. *set transaction* is evaluated for calls to DRIVE programs in the local system. It is ignored when calling DRIVE programs on the remote system. No *set transaction* entry is allowed when calling UTM program units.

Runtime

The maximum number of asynchronous UTM conversations which can run in parallel is determined by the UTM generation (see DRIVE Programming System [1] manual).

It is therefore possible that a program which is started with ENTER will not be executed immediately.

The order in which programs which are started as asynchronous UTM conversations are processed is not always identical to the sequence of ENTER statements read by the system.

Relationship to other statements

– DO statements have no effect within transaction-driven programs which are started as asynchronous UTM conversations. Instead you may use ENTER statements in such cases.

– A program started as an asynchronous UTM conversation may not contain any CALL statements which call a UTM program unit (CALL TAC).

– A program started as an asynchronous UTM conversation may not contain any CALL statements which call an old style program.

– A program started as an asynchronous UTM conversation may not contain any remote CALL statements.

– When a program is called in the local system, all the specifications for parameterizing the program which is started as an asynchronous UTM conversation are taken over from the PARAMETER statement of the calling process.

– If the program is compiled with compiler option OPTION OBJECT=ON then you may not specify *library* when the program is called.

EXECUTE
Generate and execute statement dynamically

This application is valid

– in TIAM and UTM mode
– in program mode

At runtime, EXECUTE generates exactly one statement, analyzes it and executes it immediately.

The dynamically generated statement string must be a syntactically correct DRIVE or SQL statement. This statement is then referred to as a "dynamic" DRIVE or SQL statement.

In contrast to the normal rule that DRIVE statements must be terminated with a semicolon, statements in an EXECUTE statements need not be terminated with a semicolon.
This means, for instance, that either of the following are permissible:
EXECUTE 'OPEN c1;'; or EXECUTE 'OPEN c1';.
If two statements are specified, separated by a semicolon, only the first is executed dynamically. The second is ignored.

Any declarative parts to be used in a statement must previously have been defined during the execution of the program, either statically in the declaration section or dynamically in a preceding EXECUTE. Declarative parts defined only dynamically using EXECUTE are subject to the following restriction: dynamically declared parts cannot be accessed in non-declarative statements in static programs.

Any cursor and views declared dynamically in a program remain in effect until the end of the program or until a dynamic DROP is explicitly specified. Permanent cursors remain valid until program mode is terminated.

When dynamic SQL statements are executed during access to a SESAM V2.x database, any previous SET SESSION, SET CATALOG and SET SCHEMA statements are evaluated. If there are no statements of this type, DRIVE/WINDOWS consults the corresponding PARAMETER DYNAMIC statements (AUTHORIZATION, CATALOG, SCHEMA operands).

EXECUTE { char-expression }

---

char-expression

The result of *char-expression* must be a syntactically correct DRIVE statement (see the *char-expression* metavariable).

The following statements can be executed dynamically:

SQL statements	DRIVE statements
ALTER TABLE	CALL
CLOSE cursor	DECLARE FORM
COMMIT WORK	DECLARE LIST
CREATE SCHEMA	DISPLAY
CREATE TABLE	DO
CREATE VIEW	ENTER
CREATE TEMPORARY VIEW	FILL
DECLARE ... CURSOR ...	LIST
DELETE	SEND MESSAGE
DROP CURSOR	SET
DROP CURSORS	UNSAVE
DROP SCHEMA	
DROP TABLE	
DROP VIEW	
DROP VIEWS	
FETCH	
GRANT	
INSERT	
OPEN	
PERMIT	
PRAGMA	
RESTORE	
REVOKE	
ROLLBACK WORK	
SELECT	
SET CATALOG	
SET SCHEMA	
SET SESSION	
SET TRANSACTION	
STORE	
UPDATE	

If *char-expression* is a literal, the complete statement (including the terminating semicolon) must be enclosed in single quotes (').

If *char-expression* contains literals, these may not exceed 256 characters in length.

You may not specify any CALL, DO or ENTER statements if you use the DRIVE compiler DRIVE/WINDOWS-Comp.

An error message is issued if an error occurs.
In UTM mode, the process is aborted.
In TIAM mode, the program is aborted.

If a "CALL ..." statement is included in an EXECUTE statement without specifying a library then an external DRIVE subprogram is called.

Example 1

The contents of a cursor table with the name "c1" are only determined at runtime and can be structured as being variable.
The *query-expression* in the cursor declaration is located in the variable &S. Different values can be assigned to this variable during runtime. By means of dynamic statement execution (EXECUTE), the cursor declaration is generated, analyzed and executed immediately at runtime. The same applies to opening the cursor and assigning values to variables.

The statements following EXECUTE must be enclosed in single quotes.

```
...
/* Variable declaration */
DECLARE VARIABLE &s CHAR (250);
DECLARE VARIABLE &c CHAR (20) INIT 'CLOSE c1';
DECLARE VARIABLE 1 &e,
                   2 e1 CHAR (20) INIT 'FETCH c1 INTO',
                   2 e2 CHAR (20) INIT '&f1,&f2,&f3';

/* Execution section */
...
SET &s='SELECT * FROM t1 WHERE a=0';
EXECUTE 'DECLARE c1 CURSOR FOR ' || &s;
EXECUTE 'OPEN c1';
EXECUTE CHARACTER (&e);
EXECUTE &c;
```

Example 2

At runtime, two concatenated variables (CONCAT (&query-command,&query-text)) and a simple variable (&error-message) are executed dynamically. Values were previously assigned to the variables.

```
DECLARE VARIABLE
   &query-command CHAR(15) INIT 'DISPLAY FORM',
   &query-text CHAR(74) INIT
      'NL 15,TAB 5,"Enter the sequence number: "RETURN &number;',
   &error-message CHAR(74) INIT
      'SEND MESSAGE "Record does not exist. Press DUE";';
...
EXECUTE CONCAT (&query-command,&query-text);
...
EXECUTE &error-message;
```

The following variable declarations form the basis for examples 3 through 6.

```
DECLARE VARIABLE
    1 &e,
       2 table-n   CHAR (19) INIT 'T1',
       2 set-s     CHAR (04) INIT 'SET',
       2 set-c     CHAR (200),
       2 w,
         3 w1      CHAR (10) INIT 'WHERE',
         3 where-c CHAR (200);
DECLARE VARIABLE &opc  CHAR (30) INIT 'DELETE FROM UPDATE';
DECLARE VARIABLE $v    CHAR (17) INIT 'V1';
DISPLAY FORM LINE
        RETURN &table-n,
        RETURN &set-c INIT 'F1=100',
        RETURN &where-c INIT 'F2=2';
```

Example 3

Cursor "c1" is opened dynamically.

The statement must be enclosed in apostrophes.

```
...
EXECUTE 'OPEN c1';
...
```

Example 4

The statement "UPDATE T1 SET F1=100 WHERE F2=2;" is executed dynamically using a concatenation (CONCAT) and a substring (SUBSTRING). The substring (SUBSTRING) is defined at position 13 and having a length of 6.

```
...
EXECUTE CONCAT (SUBSTRING (&opc,13,6) CHAR(&e));
...
```

Example 5

The statement "DELETE FROM T1 WHERE F2=2;" is executed dynamically using the concatenation symbol "||" and a substring (SUBSTRING).

```
...
EXECUTE SUBSTRING (&opc,1,12) || &table-n || CHARACTER (&w);
...
```

Example 6

The statement "DECLARE c1 CURSOR FOR SELECT * FROM T1V1 WHERE F2=2;" is executed dynamically.

```
...
EXECUTE 'DECLARE c1 CURSOR FOR SELECT * FROM' || UPDSTRING (&table-n,&v,3) ||
CHARACTER (&w);
...
```

EXIT
Terminate DRIVE run

This application is valid

– in TIAM and UTM mode
– in interactive mode
– in program mode, only as screen input in a program

EXIT is used to terminate the DRIVE run. Any open transaction is rolled back without an error message. The contents of EDT work file 0 are not saved.

EXIT has the same effect as a K/F key assigned ACTION=EXIT.

In TIAM mode, all the requested files are closed and all views and DRIVE-specific memory areas are released.

In UTM mode, all the lists for the conversation are printed and then deleted from the central print file unless they have been explicitly printed using LIST * ... DELETE. A transaction code (TAC) or KDCOFF must then be entered.

```
EXIT
```

FILL form-name
Create and fill DRIVE screen form

This application is valid

– in TIAM mode
– in UTM mode but not in asynchronous UTM mode and not in the receiving partner environment in DTP
– in program mode

FILL *form-name* is used to define the contents and layout of screen forms. The statement fills the memory area defined for the DRIVE form with DECLARE FORM.

If the statement causes a screen overflow, a DISPLAY is executed implicitly.

Entry fields may only be used on one screen page; otherwise, DRIVE/WINDOWS indicates an error.

Entry fields are displayed at high intensity, output fields at low intensity.

When a group (= data group or repeating group) is to be output, the names of the simplest components (= lowest level) are always output. This requires that the *format* specification provides for the output of names (NAMES).
If LINE is specified for *format*, the structure names are also output.

Data values that extend beyond the end of a line are continued at the beginning of the following line.

```
FILL form-name [ format ]

        { [ RETURN ] expression [ INIT expression1 [ NOCHECK ] ]
            [ ATTRIBUTE ( attribute, ... ) ] [ mask ] |
          NEWLINE n |
          NEWPAGE n |
          TABULATOR n |
          BLANK n }, ...
```

---

form-name — Name of the DRIVE screen form (max. 31 characters). It must be defined in the declaration section of the program with DECLARE FORM.

format

Determines the screen format of the FILL area (see metavariable *format*).

If *format*=TABLE:

– You may not specify NEWPAGE and NEWLINE.

– The program is aborted if data values extend beyond the end of the line. If TABLE is not specified then these values are continued at the start of the next line.

If *format*=NAMES:

– You may not specify RETURN.

– If literals are output, they are output as names.

If *format*=LINE:

– The specifications TABULATOR and BLANK are followed by a line feed with a blank line.

RETURN

Specifies that a variable becomes an entry field, which may be preset. A variable for which RETURN has been specified may only be used as an entry field once for each form.

If an input variable (with RETURN) does not fit on a screen page, the procedure is aborted at execution time.

If an output variable (without RETURN) does not fit on a screen page, it is truncated when output. The last three characters of the output variable are represented by ">>>".

expression

Defines output and/or entry fields for the screen form.

expression may be one or more variables (including system variables) and/or one or more literals.

If RETURN, INIT or *mask* is specified, *expression* must be a variable that may not be qualified with ".*".

INIT

Assigns an initial value to *expression*. The INIT clause is permitted only if *expression* is a variable.

If *expression* is a vector or a matrix, all components receive the corresponding initial value *literal* or NULL.

expression1

expression1 may only be *literal*, NULL or a function whose arguments are literals (not CURRENT DATE/TIME/TIMESTAMP). *expression1* must be calculable at compilation time.

NOCHECK	A CHECK clause (see the metavariable *check*) specified when declaring the *expression* variable is not evaluated for the assignment of the initial value. NOCHECK is not permitted for redefined variables or variables which redefine other variables.
ATTRIBUTE	Assigns field attributes to screen forms.
attribute	Field attribute (see metavariable *attribute*)
mask	Defines the representation options for masked input and output (= output editing). *mask* may only be specified if *expression* is a simple variable or a simple component.
NEWLINE n	NEWLINE causes an advance of *n* lines. The current line is completed with NEWLINE 1. Any subsequent data is written in the next line. Even if the current line has already been filled, i.e. it already contains as many characters as defined in the COLUMNS specification of the corresponding DECLARE, NEWLINE 1 does not cause a blank line to be output. Instead, it positions to the next line. If *n* has the value "0", a conditional line feed is performed, i.e. if the current line is blank, NEWLINE 0 is ignored, if the current line is not blank, NEWLINE 0 has the same effect as NEWLINE 1.
NEWPAGE n	NEWPAGE causes an advance of *n* pages. The current page is completed with NEWPAGE 1. Any subsequent data is written on the next page. Even if the current page has already been filled, i.e. it already contains as many lines as defined in the LINES specification of the corresponding DECLARE, NEWPAGE 1 does not cause a blank page to be output. Instead, it positions to the next page. If *n* has the value "0", a conditional page feed is performed, i.e. if the current page is blank, NEWPAGE 0 is ignored, if the current page is not blank, NEWPAGE 0 has the same effect as NEWPAGE 1.
TABULATOR n	The output is continued from column *n*. If the value is less than the current column position, a line or page feed is performed. The resulting gap is filled with blanks. IF TABULATOR is specified without a value *n*, either this has no effect or one line feed is performed. The following applies: 1 ≤ TABULATOR ≤ COLUMNS

BLANK n

BLANK causes *n* blanks to be inserted. This may result in a line feed or page feed.

n is a whole number.

Example

For a definition of the screen form "output-staff" refer to page 63.

The FILL area of this screen form is filled as follows:
The names of the simplest components in the group item &staffrec are output first. The data contents of the group item &staffrec then continue to be output until the end of the cursor table "stcursor" is reached.

```
1995-12-20                                                15:28:32

                              Staff
                             _______

surname              first name            salary
Winterberg           Abigail              3500.00
Martin               George               2700.00
.
.
.

============================================================

DECLARE stcursor CURSOR ...
DECLARE VARIABLE &staffrec,
                  2 surname     CHAR (20),
                  2 first name  CHAR (20),
                  2 salary      NUM (7,2);
...
FILL output-staff TABLE NAMES &staffrec;
CYCLE macursor INTO &staffrec.*;
   FILL output-staff TABLE VALUES &staffrec;
END CYCLE;
```

FILL list-name
Create and fill in list form

This application is valid

– in TIAM mode
– in UTM mode but not in asynchronous UTM applications
– in program mode

FILL *list-name* is used to define the contents and layout of lists for data output. If the statement results in a page overflow, a DISPLAY *list-name* is implicitly executed.

In UTM applications, the central print file must be generated. There are three options for generation:

– the DRI.LIST.FILE file already exists
– the file is assigned via the file link name DRILIST
– the DRI.LIST.FILE is generated when the first DRIVE/WINDOWS DISPLAY statement is executed

The actual output on the SPOOL printer occurs at the next STOP or LIST statement (see the LIST and STOP statements).

```
FILL list-name [ format ]

     { expression [ mask ] |
       NEWLINE n |
       NEWPAGE n |
       TABULATOR n |
       BLANK n }, ...
```

---

list-name	Name of the list form (max. 31 characters). It must be defined in the declaration section of the program using DECLARE LIST.
format	Specifies the format of the list form. If TABLE is specified for *format*, NEWPAGE and NEWLINE must not be specified. If LINE is specified for *format*, TABULATOR and BLANK cause a line feed with a blank line.

expression	Defines output fields for the list form. *expression* may be one or more variables (including system variables) and/or one or more literals. If an output variable does not fit on a screen page, it is truncated when output. The last three characters of the output variable are represented by ">>>".
mask	Defines the representation options for masked output (= output editing). *mask* may only be specified if *expression* is a simple variable or a simple component.
NEWLINE n	NEWLINE causes an advance of *n* lines. The current line is completed with NEWLINE 1. Any subsequent data is written in the next line. Even if the current line has already been filled, i.e. it already contains as many characters as defined in the COLUMNS specification of the corresponding DECLARE, NEWLINE 1 does not cause a blank line to be output. Instead, it positions to the next line. If *n* has the value "0", a conditional line feed is performed, i.e. if the current line is blank, NEWLINE 0 is ignored, if the current line is not blank, NEWLINE 0 has the same effect as NEWLINE 1.
NEWPAGE n	NEWPAGE causes an advance of *n* pages. The current page is completed with NEWPAGE 1. Any subsequent data is written on the next page. Even if the current page has already been filled, i.e. it already contains as many lines as defined in the LINES specification of the corresponding DECLARE, NEWPAGE 1 does not cause a blank page to be output. Instead, it positions to the next page. If *n* has the value "0", a conditional page feed is performed, i.e. if the current page is blank, NEWPAGE 0 is ignored, if the current page is not blank, NEWPAGE 0 has the same effect as NEWPAGE 1.
TABULATOR n	The output is continued from column *n*. If the value is less than the current column position, a line feed is performed. The resulting gap is filled with blanks. The following applies: 1 < TABULATOR < COLUMNS
BLANK n	BLANK causes *n* blanks to be inserted. This may result in a line feed or page feed. *n* is an integer.

GET FILE POSITION
Read file position

This application is valid

– in TIAM and UTM mode
– in program mode

GET FILE POSITION reads the current file position in an open file and transfers the value to a variable.

In ISAM files you use the LOCATE FILE statement for positioning via the ISAM key.

```
GET FILE POSITION file TO variable
```

---

file	Logical name of the file from which the file position is to be read. This must be the name under which the file has been declared in the program using the DECLARE FILE statement.
variable	Name of the variable to which the file position is passed (see the *variable* metavariable). The variable *variable* must be large enough to store the file position (260 bytes).

GET MODIFIED INDEX
Record modified list line

This statement is valid

– in UTM mode
– in program mode

GET MODIFIED INDEX records modified lines from list areas in the sequence in which they are modified by the user.

```
GET { FIRST | NEXT } MODIFIED INDEX INTO variable FROM screenform
```

FIRST	Records the first line to be modified in the list area.
NEXT	Records the next line to be modified in the list area. You may not specify NEXT unless you have already recorded a line from the same list area using FIRST. If a list area contains no further modified lines, NEXT returns the NULL value.
variable	Name of the variable in which DRIVE/WINDOWS stores the number of the modified list line *variable* must be a numeric variable.
screenform	FHS-DE partial form with a list area. The partial form must be output using DISPLAY SCREEN.

You cannot use GET MODIFIED INDEX to process two list areas in parallel.

GET SCREEN CURSOR
Read cursor position

This statement is valid

– in UTM mode
– in program mode

GET SCREEN CURSOR reads the position of the cursor in a screen form. The screen form may be a partial form or a dialog box.

If the cursor is positioned in a named field, DRIVE/WINDOWS identifies the field name. If the field is unnamed, DRIVE/WINDOWS ascertains the absolute position of the cursor in the partial form or dialog box.

```
GET SCREEN CURSOR INTO variable1, variable2, variable3 FROM screenform
```

---

variable1
: Name of the variable in which DRIVE/WINDOWS stores the name of an FHS-DE partial form or an FHS-DE dialog box if the cursor is positioned in a named field.

  variable1 contains the NULL value if the cursor is positioned in an unnamed field.

  variable1 must be an alphanumeric variable. *variable1* must be at least 8 characters long.

variable2
: Name of the variable in which DRIVE/WINDOWS stores the line (absolute position) if the cursor is positioned in an unnamed field.

  variable2 contains the NULL value if the cursor is positioned in a named field.

  variable2 must be a numeric variable.

variable3
: Name of the variable in which DRIVE/WINDOWS stores the column (absolute position) if the cursor is positioned in an unnamed field.

  variable3 contains the NULL value if the cursor is positioned in a named field.

  variable3 must be a numeric variable.

screenform	FHS-DE form (partial form or dialog box) containing the cursor whose position you want to read. *screenform* must be output on the screen.

IF
Program condition

This application is valid

– in TIAM and UTM mode
– in program mode

IF identifies the beginning of a condition. The end of a condition is defined by END IF. Depending on the truth value of a condition, a branch is made to alternative statements within an IF block. If the condition returns the TRUE value, the system executes the THEN branch, otherwise the ELSE branch is executed.

If conditions appear within an internal subprogram, they must also be terminated with END IF (see the SUBPROCEDURE statement).

Conditions may be nested to any depth. The nesting depth is restricted only by the amount of memory space needed by DRIVE/WINDOWS for processing.

Conditions, loops (CYCLE), branches (CASE ... OF), and concurrent remote processing (DISPATCH) must not overlap.

Conditions, loops (CYCLE) and branches (CASE ... OF) may not overlap.

In a program, the IF statement may not be followed by a semi-colon. The semi-colon follows the statement in the THEN branch.

```
IF condition THEN { programming } ...
             [ ELSE { programming } ... ]
```

---

condition	Condition(s) applied to data contents of attributes and/or variables. *condition* must not contain a *projection*.
THEN	The program branches to THEN if *condition* returns the truth value TRUE.
ELSE	The program branches to ELSE if *condition* returns the truth value FALSE or UNKNOWN. If ELSE is not specified and *condition* returns the truth value FALSE or UNKNOWN, the next statement following END IF is executed.
programming	Statements for the body of a program.

Example

If the contents of the variable &response is "y", the row &article.* is inserted in the table "v-article", otherwise the subprogram "terminate" is called.

```
IF &response = 'y'
   THEN INSERT INTO v-article VALUES (&article.*);
   ELSE CALL terminate;
END IF;
```

Defining error exits

Defining an error exit with WHENEVER is the only way to avoid program abortion due to semantic errors identified during the evaluation of conditions. The error exit must be defined in the declaration section. The program is continued after END IF as defined in the error exit in WHENEVER (see the WHENEVER statement).

LIST
Output list

This application is valid

– in UTM mode
– in interactive and program mode

The LIST statement can be used to issue UTM print jobs both in interactive mode and in asynchronous mode. Printing itself takes place asynchronously. In order to generate print jobs, at least one asynchronous task, the transaction code DRILIST, and the list file must be generated. LIST accesses the user-specific data in the list file and outputs it.

LIST accesses the print-edited contents of the list file and outputs it to a printer (DEVICE) or a terminal (LTERM), or copies it into a file (FILE) or a P-member of the DRIVE library (LIB).

You can use the DELETE operand to determine whether the contents of the central print file should be deleted following printer output.

If DRIVE/WINDOWS is terminated by a STOP statement in interactive mode, the contents of the central print file are automatically printed and deleted from the print file. Automatic printing of error lists after STOP is suppressed if the parameter PARAMETER DIAGNOSIS INTTRACE LIST ='NO_ERR_LIST' has been set.

If the conversation is aborted with errors the contents of the central print file are not deleted. You must use the LIST statement to output these contents during the next UTM conversation.

UTM also generates a start file, which is used to store the user-specific DRIVE start parameters. This file cannot be output with LIST *. It is deleted when a STOP statement is entered. If errors occur, error messages are also stored in this file.

```
LIST * [ WHERE STATUS { ENTER | DIALOG | conversation } ]

       [ INTO { FILE { filename | variable1 } |
                LTERM { ltermname | variable2 } |
                DEVICE { devname | variable3 } } ]
                LIB { library(member–name) | member–name }

       [ spoolparameter ] [ DELETE ]
```

---

*	Outputs all lists.
WHERE STATUS	All outstanding print jobs for the specified conversation (interactive, asynchronous mode) are output. If you do not specify WHERE STATUS then only the print jobs for the current conversation are output.
ENTER	All the print jobs for asynchronous mode are output.
DIALOG	All the print jobs for interactive mode are output.
conversation	Variable name for the current conversation. *conversation* must be an alphanumeric variable and may be assigned only one of the values ENTER or DIALOG. *conversation* can only be specified in program mode.
INTO	INTO specifies the output medium If you do not specify INTO, then the spool parameters specified in *spoolparameter* are evaluated. Output is performed at the central printer.
FILE	Output is sent to a file.
filename	Name extension for the file to which the output is sent (max. 20 characters). *filename* must comply with BS2000 filename conventions. DRIVE/WINDOWS adds the prefix DRI.LIST.*utmname*. The full name is: DRI.LST.*utmname*.*filename*.
variable1	Variable name for the name extension described in *filename*. *variable1* must be an alphanumeric variable of type CHAR(L) where $1 \leq L \leq 21$. *variable1* may only be specified in program mode.
LTERM	Print output is sent to a UTM data display terminal entered in the KDCFILE or to a UTM workstation printer or to the output medium which is specified in LTERM in KDCDEF. This specification is not possible in TIAM mode.
ltermname	Name of a UTM data display terminal or a UTM workstation printer (max. 8 characters). *ltermname* must be specified as a literal.
variable2	Variable name for a UTM data display terminal. *variable2* must be an alphanumeric variable of type CHAR(L) where $1 \leq L \leq 8$. *variable2* may only be specified in program mode.

DEVICE	Print output is sent to a printer. All the print jobs relating to the conversation are printed.
devname	Name of a printer which must be generated in PDN (max. 8 characters). *devname* must be specified as a literal.
variable3	Variable name for a printer. *variable3* must be an alphanumeric variable of type CHAR(L) where 1≤L≤8. *variable3* may only be specified in program mode.
LIB	Output is sent to a P-member of the DRIVE library.
library	Name of the DRIVE library (max. 54 characters) to which the file is sent. *library* may also be the file link name of the DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name. If the DRIVE library has been predefined with the PARAMETER DYNAMIC LIBRARY statement, *library* need not be specified.
member-name	Name of the member (max. 31 characters) to which the list is to be output. If you do not specify *library*, the library specified in PARAMETER DYNAMIC LIBRARY is used.
spoolparameter	Designates printer management options. These options must comply with the syntax of the BS2000 PRINTFILE command (see BS2000 Commands [35]). The *spoolparameter* is passed unchecked to the printer management. *spoolparameter* is ignored if INTO FILE, INTO LTERM or INTO LIB is specified. The string may be a maximum of 256 characters in length and can be specified as a literal or as the contents of a DRIVE variable of type CHARACTER or VARCHAR.

If no print management options are specified, the SPOOLDOPT setting in the user profile is evaluated. If no specifications are available, the operand LAYOUT-CONTROL=PARAMETERS(CONTROL-CHARACTERS=PHYSICAL) is used for the BS2000 PRINT-FILE command (see BS2000 Commands [35]).

If more than one character set is used within a report, you must specify the character sets using the operand LAYOUT-CONTROL=PARAMETERS(CONTROL-CHARACTERS=PHYSICAL, CHARACTER-SETS=...) (see BS2000 Commands [35], PRINT-FILE command).

DELETE — When the print job has been terminated, the contents of the central print file are deleted.

LOCATE FILE
Locating a position in an ISAM file

This application is valid

– in TIAM and UTM mode
– in program mode

LOCATE FILE positions you at a data record in an open ISAM file.

Specify POSITION=KEY to move to a data record with the specified ISAM key. DRIVE/WINDOWS issues an error message if it is unable to locate a record with the specified key.

Specify POSITION >=KEY to move to a data record with the specified or next highest ISAM key.

```
LOCATE FILE file TO char–expression

                  [ [ WITH ] POSITION { = | >= } KEY ]
```

---

file	Logical name of a file in which a position is to be located. The file must be declared under this name in the program using the DECLARE FILE statement.
char-expression	Defines the key value (ISAM key) of the data record.
POSITION=KEY	Moves the position to the record with the ISAM key *char-expression*.
POSITION>=KEY	Default Moves the position to the record with the ISAM key *char-expression* or to the first record with an ISAM key greater than *char-expression*.

OPEN FILE
Open a file

This application is valid

– in TIAM and UTM mode
– in program mode

OPEN FILE opens a file and assigns it a logical name. The logical name must have been defined with the DECLARE FILE statement. This name is used to access the file in DRIVE programs.

DRIVE/WINDOWS allows you to open the same file under different logical names provided that this is permitted by the BS2000 operating system.

In this statement, you specify the file type and the OPEN mode to be used when opening the file.

When a file is opened with INPUT, OUTPUT, UPDATE, INOUT and OUTIN, the system positions at the beginning of the file and when you open a file with EXTEND, the system positions at the end of the file.

If a file is opened for read and write access with UPDATE, INOUT or OUTIN, read and write accesses must not follow each other directly. The SET FILE POSITION statement must occur between the READ FILE and WRITE FILE statements unless DRIVE/WINDOWS reaches the end of the file during read access.

DRIVE/WINDOWS issues an error message if a file which is to be opened for read access does not exist or does not correspond to the data type specifications.

The BS2000 operating system checks the file access permissions. If access is not permitted, DRIVE/WINDOWS intercepts and outputs the operating system error message.

Do not overwrite the contents of text files. Since the actual stored length of variables is unknown, new data may be longer or shorter than the data which is overwritten. In both cases the structure of the text file is modified and the text file can no longer be read via the DRIVE variable (which describes it).

```
OPEN FILE file IN bs-file
               [ SAM | ISAM | BIN | CHARACTER character ]
               [ INPUT | OUTPUT | EXTEND | UPDATE | INOUT | OUTIN ]
```

file	Logical name of a file (max. 31 characters). The file must have been defined with this name in the program using the DECLARE FILE statement. No file with this logical name may already be open.
IN	Specifies the file to be accessed under this logical name.
bs-file	Name of a file at operating system level (max. 54 characters) which is to be opened. *bs-file* may be the file link name of the file. DRIVE/WINDOWS first interprets *bs-file* as a file link name and then as a file name. *bs-file* must be a file link name if an ISAM file is to be processed as a SHARED-UPDATE file. This name must comply with the BS2000 naming conventions. *bs-file* must be of the type *char-expression*.
SAM	Default The file is a SAM file.
ISAM	The file is an ISAM file.
BIN	The file is a binary file (see the DRIVE Programming Language manual [2]).
CHARACTER	The file is a text file (see the DRIVE Programming Language manual [2]).

character	Specifies the field delimiter (one character). Default: blank Do not specify any of the following for *character*: – the end-of-line character – the end-of-file character – the blank (␣) – the characters + - , or . These could lead to unpredictable results. The delimiter *character* must be specified as an alphanumeric literal (*char-literal*, see the *literal* metavariable) or as a hexadecimal character (*hex-literal*, see the *literal* metavariable).
INPUT	Open for reading. The file must already exist.
OUTPUT	Default Open for writing.If a file with the name *bs-name* already exists, the old contents are deleted. Otherwise the file is created.
EXTEND	Open for writing. If a file with the name *bs-name* already exists, the old contents are retained and the new data is appended at the end of the file. If the file does not exist, it is created. You may not specify EXTEND for ISAM files.
UPDATE	Open for reading and writing. If a file with the name *bs-name* already exists, the old contents are retained and can be overwritten.
INOUT	Open for reading and writing. The file must already exist. The old contents are retained and can be overwritten.
OUTIN	Open for reading and writing. If a file with the name *bs-name* already exists, the old contents are deleted. If the file does not exist, it is created.

Special file characteristics

– You may process files with K and NK block format. (See the DMS Introductory Guide [37])

– If you create a file without specifying the file type or opening mode, the file is assigned the values SAM and OUTPUT. All other file attributes match the default settings for the BS2000 operating system.

– When you create a file it is assigned the following attributes depending on the specified data type:

  SAM: record format: (V,N)

  ISAM: record format: (V,N)
  key position: 5
  key length: 8

– If a file already exists, the specifications from the file catalog or file link entry apply. An existing file retains its catalog attributes even after its contents have been deleted.

– In DRIVE/WINDOWS you can only process files which possess block control information in the data block (block format DATA). Even if you access files without block control information (block format NO), the file is assigned the block format DATA (see the DMS Introductory Guide [37]).

– If you use the file link name to open a file, then you can specify the SET-FILE-LINK command to modify the following attributes in the BS2000 system:
access method, record format, record length, block format and block length.

  You must enter the SET-FILE-LINK command:

  – in TIAM mode, either before the start of the DRIVE program or with the SYSTEM statement in the DRIVE program. The SYSTEM statement must precede the OPEN-FILE statement.
  – in UTM mode, before the start of the DRIVE application.

– If ISAM files are to be processed by more than one program simultaneously (SHARED-UPDATE=YES) then you must use file link names to address these. In BS2000, enter the SET-FILE-LINK command with the operand LINK-NAME= and structure SUPPORT=...(SHARED-UPDATE=YES).

  You must enter the SET-FILE-LINK command:

  – in TIAM mode, either before the start of the DRIVE program or with the SYSTEM statement in the DRIVE program. The SYSTEM statement must precede the OPEN-FILE statement.
  – in UTM mode, before the start of the DRIVE application.

– The ISAM keys form part of the record data which is written by the DRIVE program or supplied to the DRIVE program when read.
  You must specify the position and length of the key using the corresponding SET-FILE-LINK command
  (SET-FILE-LINK ... KEY-LENGTH=..., KEY-POSITION=...).

  You must enter the SET-FILE-LINK command:

  – in TIAM mode, either before the start of the DRIVE program or with the SYSTEM statement in the DRIVE program. The SYSTEM statement must precede the OPEN-FILE statement.
  – in UTM mode, before the start of the DRIVE application.

– The following applies to UTM mode:
  Files which are opened using INPUT as well as ISAM files which are declared as SHARED-UPDATE files in the SET-FILE-LINK statement are only opened once during the UTM conversation (on the first OPEN FILE statement) and are then closed by DRIVE/WINDOWS when the conversation terminates. After the first OPEN statement, all further CLOSE FILE and OPEN-FILE statements are ignored. It is also not possible to re-open an open file in a different OPEN mode.

You are not allowed to process ISAM files with key duplication (DUPKEY=YES).

Relationship to other statements

You may enter the file name or file link name *bs-file* in uppercase or lowercase characters. The name is automatically converted to uppercase in BS2000. No specifications made in the PARAMETER DYNAMIC LETTERS or OPTION LETTERS statements are valid for the BS2000 operating system.

OPTION
Control compilation of a program

This application is valid

– in TIAM and UTM mode
– in interactive and program mode

OPTION controls the compilation run for a DRIVE program. It is used to set compiler options for compiling DRIVE programs to form intermediate code.

You may specify OPTION in a source (ahead of the PROCEDURE and DECLARE TYPE statements) or in the COMPILE statement.

OPTION specifications in the source override DRIVE/WINDOWS defaults. However, OPTION specifications in the source are overridden by OPTION specifications in the COMPILE statement.

Some of the OPTION statement operands are not supported by certain operating systems (BS2000, SINIX and MS-Windows) (see the DRIVE directories for SINIX [12] and MS-Windows[9]). For reasons of compatibility, DRIVE/WINDOWS ignores such operands and they remain without function.

```
OPTION { AUTHORIZATION=authorization |
         CATALOG=ses-db-name |
         CODE={ OFF | ON } |
         DBSYSTEM={ OFF | SESAM | SESAMSQL | UDS } |
         DCSYSTEM={ TIAM | UTM | BOTH } |
         DECIMALSIGN={ . | , } |
         DISTRIBUTION={ OFF | ON } |
         LETTERS={ CAPITAL | BOTH | UNCHANGED } |
         LISTING={ OFF | LIBRARY | LIST | BOTH } |
         LISTTYPE={ OFF | USER | EXPERT } |
         MONINFO={ OFF | ON } |
         NULLVALUE={ OFF | ON } |
         OBJECT={ OFF | ON } |
         PERMIT={ OFF | ON } |
         SCHEMA=schema-name |
         SCREENCHECK { ON | OFF }
         TASKTYPE={ DIALOG | ENTER | BOTH } |
         VERSIONMIX={ OFF | ON } |
         XREF={ OFF | ON } } ...
```

AUTHORIZATION	Specifies the authorization key *authorization* for a SESAM database (max. 18 characters) (see SQL directory for SESAM V2 [5]).
	Default: the current setting from PARAMETER DYNAMIC AUTHORIZATION
CATALOG	Specifies the default for a SESAM V2.x database *sesdbname* (max. 18 characters) (see SQL directory for SESAM V2 [5]).
	Default: the current setting from PARAMETER DYNAMIC CATALOG or blanks
CODE	Specifies whether the intermediate code which is generated is to be stored.
=OFF	Default
	The intermediate code is not stored.
=ON	The intermediate code is stored in the DRIVE library (member type X).
	The member with the intermediate code is given the same name as the member which contains the source program.
DBSYSTEM	Specifies the database system to be accessed by the SQL statements in a DRIVE program.
	Default: the loaded variant if the DRIVE program which is to be compiled contains SQL statements (with the exception of COMMIT WORK and ROLLBACK WORK).
=OFF	The DRIVE program to be compiled contains no SQL statements which access a database other than COMMIT WORK and ROLLBACK WORK.
	Default if the DRIVE program which is to be compiled contains no SQL statements apart from COMMIT WORK and ROLLBACK WORK.
	If no database system has been specified for a program (OPTION DBSYSTEM=OFF), the database defined for the calling program or, (in the case of a return) the called program is determined at runtime (see the DRIVE Programming Language manual [2]).
=SESAM	SQL statements access a SESAM V1.x database.
=SESAMSQL	SQL statements access a SESAM V2.x database.
=UDS	SQL statements access a UDS database.

DCSYSTEM	Specifies the target communication system in which the program is to be executed.
	Default: the current value at compilation time
=TIAM	For TIAM mode
=UTM	For UTM mode
=BOTH	For program use in either mode.
DECIMALSIGN	Specifies the decimal sign in the source program.
	Default: period (.)
	This option is also used in the analysis of dynamic statements (see the EXECUTE statement).
DISTRIBUTION	Specifies whether the distribution information is to be evaluated when programs are called with CALL or ENTER.
=OFF	Default
	The distribution information is not evaluated.
=ON	The distribution information is evaluated at execution time.
LETTERS	Specifies how letters are to be managed in the compiled program. This specification affects names and literals.
	This option is also used in the analysis of dynamic statements (see the EXECUTE statement).
=CAPITAL	Default
	Only uppercase characters are processed by the compiled program.
	All lowercase characters in names and literals are converted to uppercase characters. Umlauts are not converted.
=BOTH	Both uppercase and lowercase characters are processed by the compiled program.
	Lowercase characters in names are converted to uppercase. Lowercase characters in literals are not converted to uppercase.
	Keywords and metavariables are always converted (this only applies when creating an OLTP application for BS2000 and during remote access).

=UNCHANGED	Lowercase characters are not converted to uppercase for names or literals. You should not change the specification LETTERS=UNCHANGED since software products such as LMS, EDT, SESAM, UDS which work with DRIVE/WINDOWS handle lowercase letters in different ways.
LISTING	Specifies whether an interpreter listing is to be created.
=OFF	Default No interpreter listing is generated.
=LIBRARY	An interpreter listing is stored in the current DRIVE library (as a member of type P). The member with the interpreter listing and the member that contains the source program have the same name.
=LIST	In TIAM mode an interpreter listing is output to SYSLST. In UTM mode the interpreter listing is written to the central print file.
=BOTH	The interpreter listing is output to the DRIVE library. In addition the interpreter listing is output to SYSLST in TIAM mode and to the central print file in UTM mode.
LISTTYPE	Specifies whether a compiler listing is to be generated. LISTTYPE may only be specified if the DRIVE compiler DRIVE/WINDOWS-Comp is being used (see the DRIVE Compiler manual [16]).
=OFF	Default No compiler listing is generated.
=USER	A compiler listing with the generated assembler code is output to SYSLST.
=EXPERT	A compiler listing with the generated assembler code and a cross-reference list are output to SYSLST.

MONINFO	Specifies whether installation information (MONINFO) is to be generated. MONINFO may only be specified if the DRIVE compiler DRIVE/WINDOWS-Comp is being used (see the DRIVE Compiler manual [16]).
=OFF	Default No installation information is generated.
=ON	Installation information is output to SYSLST.
NULLVALUE	Specifies whether the null value is to be recognized for variables in the program. NULLVALUE may only be specified if the DRIVE compiler DRIVE/WINDOWS-Comp is being used (see the DRIVE Compiler manual [16]).
=OFF	Default The null value is not recognized. The program must not contain any assignments with NULL.
=ON	The null value is recognized.
OBJECT	Specifies whether the compiler should be started and object code (member type R) generated. OBJECT may only be specified if the DRIVE compiler DRIVE/WINDOWS-Comp is being used (see the DRIVE Compiler manual [16]).
=OFF	Default No compiler run. No object code is generated.
=ON	The DRIVE/WINDOWS-Comp compiler is started. Object code is generated.

PERMIT — Specifies whether the generated object should be accompanied by a screen for the input of a user ID (see the PERMIT statement in the SQL directories [4] and [6]).

PERMIT may only be specified if the DRIVE compiler DRIVE/WINDOWS-Comp is being used (see the DRIVE Compiler manual [16]).

=OFF — Default

A password screen is not to be displayed.

=ON — A password screen is to be displayed.

SCHEMA schema-name

Name of a SESAM V1.x database (max. 18 characters), a SESAM V2.x schema (max. 31 characters) or a UDS schema (max. 30 characters) if no name is specified in the SQL statements within a program (see SQL directories [4], [5], [6]).

Default for SESAM V1.x and UDS: none; for SESAM V2.x: the current setting from PARAMETER DYNAMIC SCHEMA or blanks.

SCREENCHECK — Specifies whether DRIVE/WINDOWS evaluates the CHECK clauses which are generated by IFG in the addressing aids (see IFG [28]).

=OFF — The CHECK clauses are not evaluated.

=ON — Default

The CHECK clauses are evaluated.

TASKTYPE — Specifies how a compiled program executes.

Default: the current value at compile time.

=DIALOG — The compiled program can only run as an interactive program (see DO statement).

=ENTER — The compiled program can only be started as an asynchronous UTM conversation (see ENTER statement).

=BOTH — The compiled program can run both as an interactive program and an asynchronous UTM conversation.

VERSIONMIX	Specifies whether the compiled program can run in mixed operation. VERSIONMIX may only be specified if the DRIVE compiler DRIVE/WINDOWS-Comp is being used (see the DRIVE Compiler manual [16]).
=OFF	Default The compiled program can only run in new style DRIVE/WINDOWS operation.
=ON	The compiled program can run in DRIVE/WINDOWS mixed operation. DO statements require that called programs receive a TAC definition in UTM generation (see DRIVE Compiler [16]).
XREF	Specifies whether cross references are to be output for the compiled program. This is only done if the option LISTING=OFF was not specified.
=OFF	Default No cross-references are output.
=ON	Cross-references are output for the compiled program.

Rules

– If format entries are made at runtime, the OPTION specifications LETTERS and DECIMALSIGN have no effect.

  Exception: format entries converted in a dynamic SQL statement are subordinate to the OPTION specifications.

Relationship to other statements

– If you do not specify a library member in DO or COMPILE then the following compiler options are ignored in the source: LISTING=LIBRARY, CODE=ON and OBJECT=ON.

– If DO is entered, the following source options are ignored: CODE=ON and OBJECT=ON.

– If you specify the option TASKTYPE=DIALOG in a program and start this program using the ENTER statement, the asynchronous UTM conversation will of course continue to run.

Summary of default values

Option	Default
AUTHORIZATION	Current setting from PARAMETER DYNAMIC AUTHORIZATION
CATALOG	Current setting from PARAMETER DYNAMIC CATALOG or blanks
CODE	OFF
DBSYSTEM	Loaded variant or UDS
DCSYSTEM	Current value at compilation time
DECIMALSIGN	. (point)
DISTRIBUTION	OFF
LETTERS	CAPITAL
LISTING	OFF
LISTTYPE	OFF
MONINFO	OFF
NULLVALUE	OFF
OBJECT	OFF
PERMIT	OFF
SCHEMA	Current setting from PARAMETER DYNAMIC SCHEMA or blanks
SCREENCHECK	ON
TASKTYPE	Current value at compilation time
VERSIONMIX	OFF
XREF	OFF

Examples

The following options are to be entered in the source "JOBS":

– With SQL statements where no schema name has been specified explicitly, the name "TEST" is to be used.

– Cross-references should be output to SYSLST.

The OPTION specifications must **precede** the PROCEDURE statement in the source code:

```
OPTION SCHEMA=TEST XREF=ON LISTING=LIST;
PROCEDURE order;
...
```

COMPILE starts the compiler run for the source "JOBS". The following options are to be respected:

– The generated intermediate code is to be stored in the DRIVE library under the name "JOBC" (member type X).

– No cross-references are to be printed. Since an OPTION specification overrides a corresponding specification in the source, here XREF=OFF

– An interpreter listing is to be generated and stored in the current DRIVE library (member type P). The interpreter listing has the same name as the intermediate code ("JOBC").

```
COMPILE JOBS INTO JOBC
        OPTION LISTING=LIBRARY CODE=ON XREF=OFF
```

PARAMETER
Select PARAMETER statement

This application is valid

– in TIAM and UTM mode
– in interactive mode

PARAMETER outputs a menu from which it is possible to branch to the menu of one of the following PARAMETER statements:

– PARAMETER DIAGNOSIS
– PARAMETER DYNAMIC
– PARAMETER KFKEY

– PARAMETER LOCK
– PARAMETER STATIC

It is also possible to branch directly into the menu screens of the individual statements. This is done by entering PARAMETER with the appropriate operands.

```
PARAMETER [ DIAGNOSIS |
            DYNAMIC |
            KFKEY |
            LOCK { DIALOG | PROCEDURE } |
            STATIC ]
```

DIAGNOSIS	The menu screen with the current parameter values (except for the INTTRACE operand) is displayed. The parameter values can be corrected by the user. Invalid entries are redisplayed at the terminal and can also be corrected by the user.
DYNAMIC	The menu screen with current parameter values that can be corrected by the user is displayed. Invalid entries are redisplayed at the terminal and can be likewise corrected by the user.
KFKEY	The menu screen with the current key assignments is displayed. In TIAM applications, key assignments can be corrected at any time. Invalid entries are redisplayed at the terminal and can also be corrected by the user. Key assignments cannot be corrected in UTM applications.
LOCK	The menu screen with the statements to be locked in interactive or program mode is displayed. The statements, which are arranged in alphabetical order, may be locked by the user. Invalid entries are redisplayed at the terminal and can be corrected by the user.
DIALOG	The statements for interactive mode are displayed.
PROCEDURE	The statements for program mode are displayed.
STATIC	The menu screen with the current parameter values that can only be corrected by the user in the first call is displayed. Subsequent calls can only retrieve information on these parameter values.

PARAMETER DIAGNOSIS
Activate tracing

This application is valid

– in TIAM and UTM mode
– in interactive mode
– in program mode only in the body of an interactive program

PARAMETER DIAGNOSIS controls tracing and specifies special features for program execution.

The trace is logged in the internal DRIVE diagnostic file `DRI.INTTRACE.FILE`. The file `DRI.INTTRACE.FILE` is an ISAM file. (See DRIVE Programming System manual [1]).

This statement may be specified in menu-driven mode without the use of operands or with the use of at least one operand.

Without operands, PARAMETER DIAGNOSIS can only be used in interactive mode.

If you specify multiple operands within a statement, these become effective simultaneously. If an operand is specified in both interactive and program mode, the program mode specification applies.

If errors occur during execution, none of the statement is executed and the errored statement is marked and output. An error message is also displayed in the message line.

A PARAMETER DIAGNOSIS statement within a DRIVE program (compilation unit) has no effect on the compiler run for that program.

```
PARAMETER DIAGNOSIS [ACCOUNT { ON | OFF }
                     DBTRACE={ ON | OFF } |
                     DMSTRACE={ ON | OFF } |
                     INTTRACE={ ON | OFF } |
                     MEMTRACE={ ON | OFF } | ...
                     { 'FILEON' | 'FILEOFF' } ]
```

ACCOUNT	Specifies whether or not performance is measured. The measured data is written to the file DRI.ACCOUNT.DAT. This file is an ISAM file. Default: OFF
DBTRACE	Specifies whether contents of the database interface are to be traced. DBTRACE may be specified only if the current DRIVE environment connects to SESAM or UDS. Default: OFF
DMSTRACE	Specifies whether the contents of the DMS interface are to be traced. DMSTRACE may be specified only if the current DRIVE environment offers access to DMS. Default: OFF
INTTRACE	INTTRACE is used for internal test purposes. You may only specify INTTRACE=ON when requested to do so by the Siemens Nixdorf Informationssysteme AG service department. Default: OFF
MEMTRACE	The memory management tables are traced. Default: OFF
'FILEON'	If the statements are read from SYSDTA then, if an error occurs, data continues to be read from SYSDTA and not – as is otherwise the case when errors occur – from the screen.
'FILEOFF'	The setting FILEON is deactivated.

Evaluation time

The following table contains an overview of the times at which the operands of the PARAMETER DIAGNOSIS statement are evaluated.

Operand	Compilation time	Runtime
ACCOUNT	yes	yes
DBTRACE	yes	yes
DMSTRACE	-	-
INTTRACE	-	-
MEMTRACE	yes	yes

PARAMETER DISTRIBUTION
Define access in a distributed system

This application is valid

– in UTM mode, but in the case of distributed transactions only, when all receiving conversations have been terminated
– in interactive and program mode

PARAMETER DISTRIBUTION can be used to define access for ENTER and CALL statements in the remote system. The distribution information that is supplied with PARAMETER DISTRIBUTION is evaluated at each ENTER statement in interactive mode. In all other program calls, the distribution information is only evaluated if the statement OPTION DISTRIBUTION=ON has been specified.

The statement must be specified with at least one operand (STATUS).

The PARAMETER DISTRIBUTION statement is evaluated at program execution time.

If PARAMETER DISTRIBUTION is specified in both interactive and program mode, the program mode specification applies.

```
PARAMETER DISTRIBUTION

        [ [ LIBRARY=library ] ELEMENT=member-name [ TYPE={ CODE | OBJECT } ] |
          TAC=tac-name ]

        [ APPLICATION=application ]

        STATUS={ OFF | ADD | REMOVE }
```

---

library — Specifies the name of the remote system DRIVE library (max. 54 characters) from which the member is read.

If the DRIVE library in the remote system has been set with the PARAMETER DYNAMIC LIBRARY statement, the *library* specification can be omitted.

You must specify *library* if a DRIVE library specification is entered in the CALL or ENTER statement.

You may not specify *library* if no DRIVE library specification is entered in a CALL or ENTER statement.

In a remote BS2000 system:

library can be the link name of the DRIVE library (in accordance with the BS2000 conventions).

DRIVE/WINDOWS first attempts to interpret *library* as a link name and then as a library name.

In a remote SINIX system:

Absolute or relative path name of a directory representing the DRIVE library.

A relative path name refers to the directory in which DRIVE/WINDOWS was started in the remote system.

The *library* and *member-name* specifications are combined with *class-name* specified for the remote system in the PARAMETER DYNAMIC CLASS statement or with the preset value for the remote system to form the file path name *library\class-name\member-name* (see the PARAMETER DYNAMIC statement).

member-name

Name of the remote system library member (max. 31 characters) to be read.

In a remote BS2000 system:

Name of the library member containing source code(= S member), intermediate code (= X member) or object code (= R member).

If *library* is not specified, the library set for the remote system with PARAMETER DYNAMIC LIBRARY is used.

In a remote SINIX system:

Name of the file containing source code, intermediate code or object code.

If *library* is not specified, the directory set for the remote system using PARAMETER DYNAMIC LIBRARY is used. If no directory has been set with PARAMETER DYNAMIC LIBRARY, the directory specified with the environment variable DRIVE_PROJECTLIB is used.

The file must be located in the directory *library/class-name* (see the PARAMETER DYNAMIC CLASS statement).

TYPE

Specifies the member type.

CODE	Default The member *member-name* is source code or intermediate code. DRIVE/WINDOWS searches for the most recently processed member in a remote BS2000 system and for the most recently processed file in a remote SINIX system. It is of no significance whether the member file contains source code or intermediate code.
OBJECT	The member *member-name* contains object code. TYPE=OBJECT may only be specified if the member *member-name* was compiled with the DRIVE/WINDOWS-Comp compiler (see DRIVE-Compiler for BS2000 [16] or SINIX [41]).
tacname	Identifies the UTM program unit that is called in the remote system. The UTM application to which the program unit belongs must have already been started at execution time. *tacname* must not be a DRIVE interpreter TAC or a DRIVE compiler TAC.
application	Name of the UTM application (max. 8 characters) in the remote system. If you do not specify *application*, the remote application specified during the generation of the KDCDEF control application LPAP is used.
STATUS	Determines how distribution information is to be handled.
OFF	Distribution information is deleted entirely. If STATUS=OFF, no other operands are required.
ADD	A new entry is included in the distribution information. If members or UTM program units of the same name already exist, they are overwritten.
REMOVE	The specified member or UTM program unit is deleted from the distribution information.

PARAMETER DYNAMIC
Specify dynamic parameter

This application is valid

– in TIAM and UTM mode
– in interactive mode
– in program mode only in the body of an interactive program

PARAMETER DYNAMIC defines parameter values that may be changed as often as desired during a DRIVE session. The parameter values determine certain attributes of the DRIVE session relating to both interactive mode and the environment in which DRIVE programs run.

This statement may be specified in menu-driven mode without the use of operands or with the use of at least one operand.

If you do not specify any operands you may only use PARAMETER DYNAMIC in interactive mode.

If you specify multiple operands within a statement, these become effective simultaneously. If an operand is specified in both interactive and program mode, the program mode specification applies.

If you enter PARAMETER DYNAMIC via the menu and errors occur during the execution of PARAMETER DYNAMIC this may be because specifications which can no longer be cancelled have already taken effect (e.g. LIBRARY). The statement to which the errors relate is marked and output. An error message is also output in the message line.

A PARAMETER DYNAMIC statement within a DRIVE program (compilation unit) has no effect on the compiler run for that program.

```
PARAMETER DYNAMIC [ AUTORIZATION=authorization |
                    CATALOG=ses-db-name |
                    DBSYSTEM={ SESAM | SESAMSQL | UDS } |
                    DECIMALSIGN={ , | . } |
                    ERRORATTRIBUTE=( attribute, ... ) |
                    FORMAT={ LINE | TABLE | SEQUENCE } |
                    LETTERS={ CAPITAL | BOTH | UNCHANGED } |
                    LIBRARY=library |
                    LOG={ OFF | IN | OUT | INOUT } |
                    LOGFILE=filename |
                    LOGPASSWORD=password |
                    NORMSQL={ ON | OFF } |
                    NULL { FORM | LIST } null-value |
                    SCHEMA=schema-name |
                    TEST={ STANDARD | ALL } |
                    USERMSGFILE=name ] ...
```

AUTHORIZATION defines the current authorization key *authorization* (max. 18 characters) for SESAM V2.x databases (see SQL directory for SESAM V2 [5]).

AUTHORIZATION=*authorization* is valid only for SQL statements which are entered in interactive mode and has no effect on SQL statements in programs.

If you specify AUTHORIZATION=*authorization* then you must work with PARAMETER DYNAMIC DBSYSTEM=SESAMSQL, i.e. you must load the SESAM V2.x database version.

Default: none

You may not specify AUTHORIZATION=*authorization* if a transaction is open.

CATALOG Name of a SESAM database *sesdb-name* (max. 18 characters) which is accessed if no name is specified in dynamic SQL statements (see SQL directory for SESAM V2 [5]).

If you specify CATALOG=*sesdb-name* then you must work with PARAMETER DYNAMIC DBSYSTEM=SESAMSQL, i.e. you must load the SESAM V2.x database version.

	CATALOG=*sesdb-name* is valid for SQL statements which are entered in Expertenmodus interactive mode and may affect dynamic SQL statements in programs (see OPTION statement and the SQL directory for SESAM V2 [5]). Default: none
DBSYSTEM	Specifies the database system which is accessed by SQL statements which are entered in interactive mode. This specification also applies to SQL statements in programs which are called with one of the statements CALL, DEBUG or DO (see the statements CALL, DEBUG and DO). You can only enter PARAMETER DBSYSTEM in interactive mode. If the specified database system does not correspond to the loaded variant, the PARAMETER DYNAMIC DBSYSTEM statement is rejected. The default is the loaded variant.
=SESAM	SQL statements access a SESAM V1.x database.
=SESAMSQL	SQL statements access a SESAM V2.x database. You must specify DBSYSTEM=SESAMSQL if you have set AUTHORIZATION=*authorization* or CATALOG=*sesdb-name*.
=UDS	SQL statements access a UDS database.
DECIMALSIGN	Defines the decimal sign for entries in interactive mode and for numeric literals in DRIVE forms. Commas (,) and periods (.) can be used as decimal signs. Default: period (.)
ERRORATTRIBUTE	Error attributes are assigned to identify incorrect screen fields
attribute	You can assign the following global field attributes: – UNPROTECTED – HIGHINTENSITY, NORMALINTENSITY – VISIBLE, SIGN, INVISIBLE – UNDERLINE, NOUNDERLINE – GREEN, RED, WHITE, YELLOW Default: HIGHINTENSITY, UNDERLINE If an incorrect entry is input in an FHS form then the specified field attribute is only displayed if SCREENERROR REPEAT has been specified in the DISPLAY screenform statement.

FORMAT — Specifies the output format for field names and the associated values.

=LINE — Default

The output format contains the field names and associated values in one line.

Example

```
CLIENTNUMBER : C03452
NAME: MILLER
TOWN: BALTIMORE
ROAD: MAIN 10
```

=TABLE — Field names and values are output in the form of a table. The field names are output in the first screen line, and the associated values are output in the following lines. The width of a given column in the table is determined by the maximum of the length of the field name and its values. Thus, neither the field name nor any of the associated values is truncated. The columns in the table are separated from one another by a space.

Example

```
CLIENTNUMBER NAME    TOWN    ROAD
K03452       MILLER  NEWARK  MAIN 10
C05734       HOWARD  BOSTON  WESTERN 5
C18982       STEEL   SEATTLE EASTERN 10
```

TABLE may only be used if the table will fit completely on the screen (line length ≤ 80 characters). If the line length exceeds 80 characters, an error message is output.

=SEQUENCE — Field names and values are output in sequential order.

Example

```
CLIENTNUMBER: C03452  NAME: MILLER  TOWN: NEWARK  ROAD:
MAIN 10  CLIENTNUMBER: C05734 ...
```

LETTERS — Determines how lowercase letters which are input in interactive mode are handled in both the EDT editor and in DRIVE forms. This specification affects only names and literals.

=CAPITAL — Default

All lowercase letters in names and literals are converted to uppercase.

=BOTH	Lowercase letters in names are converted to uppercase. Lowercase letters in literals are not converted.
=UNCHANGED	Lowercase letters in names and literals are not converted to uppercase.
	You should not change the specification LETTERS=UNCHANGED since software products such as LMS, EDT, SESAM, UDS which work with DRIVE/WINDOWS handle lowercase letters in different ways.
LIBRARY=library	*library* specifies the DRIVE library in which programs are managed.
	The specified library is defined for a system and is not transferred as a parameter to remote systems.
	There is no default name.
	library can also be the file link name of the DRIVE library (in accordance with BS2000 conventions).
	DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name.
	DRIVE/WINDOWS issues an error message if the specified library is not present.
LOG	Control dialog logging. If the dialog logging component SYSPRG.DRIVE.011.DRILOG has not yet been loaded, DRIVE/WINDOWS initiates the batch task SYSENT.DRIVE.011.DRILOG (see the "DRIVE Programming System" [1] manual).
=OFF	Default
	The dialog is not logged.
=IN	All input is logged.
=OUT	All output is logged.
=INOUT	All input and output is logged.
LOGFILE=filename	Name extension for the dialog logfile (max. 20 characters). The full filename is DRILOG.*filename*. The dialog logfile is a SAM file.
	Default: DRILOG.yymmdd.

LOGPASSWORD=password

Defines the password for the dialog logfile (max. 4 characters).

Default: blanks

You cannot specify LOGPASSWORD if you do not also specify LOGFILE.

NORMSQL — Specifies whether the statements are interpreted according to new-style or old-style conventions. NORMSQL may only be specified outside a transaction in interactive mode and when the current DRIVE environment permits mixed operation.

=ON — Default

DRIVE statements that follow are interpreted in accordance with the new style conventions.

=OFF — DRIVE statements that follow are interpreted in accordance with old-style conventions.

NULL — A character to be used to represent the null value is specified. Users can only enter null values if a character has been declared for representing the null value.

If you do not specify a null value representation in DECLARE FORM or DECLARE LIST then the representation specified in PARAMETER also applies to programs.

FORM — Specifies the null value representation for screen input/output.

Default: the special character @

LIST — Defines the null value representation for printer output.

Default: the period (.)

null-value — *null-value* specifies how null values are represented for the alphanumeric data types CHARACTER and for the numeric data types NUMERIC, DECIMAL, INTEGER and SMALLINT (see the metavariable *null-value*).

The alphanumeric null value representation also applies to date-time data types.

The numeric null value representation also applies to the data type INTERVAL.

SCHEMA=schema-name

Name of a SESAM V1.x database (max. 18 characters), a SESAM- V2.x schema (max. 31 characters) or a UDS schema (max. 30 characters) which is accessed if no name is specified in dynamic SQL statements (see SQL directories [4], [5], [6]).

SCHEMA=*schema-name* is valid for SQL statements which are entered in interactive mode and has no effect on SQL statements in programs that relate to SESAM V1.x or UDS. In the case of SESAM V2.x, SCHEMA=*schema-name* may affect dynamic SQL statements in programs (see OPTION statement and SQL directory for SESAM V2 [5]).

Exception: *schema-name* must be specified explicitly in the PERMIT statement.

In the case of SESAM V1.x and UDS you may enter SCHEMA=*schema-name* in interactive mode only.

There is no default name.

TEST — Specifies how DRIVE/WINDOWS is to react when a program is aborted.

=STANDARD — Default

Processing branches to interactive mode if a program is aborted.

In TIAM mode it is possible to branch to the EDT editor for error analysis if the program was located in the EDT work file 0.

=ALL — When a program is aborted, DRIVE/WINDOWS terminates and initiates a print output internally.

USERMSGFILE=name

First part of the message code (= identification). *name* can have a maximum length of 3 characters.

There is no default name.

Scope of application for operands

Operand	TIAM mode	UTM mode	UTM start procedure
AUTHORIZATION	yes	yes	yes
CATALOG	yes	yes	yes
DBSYSTEM	yes	yes	yes
DECIMALSIGN	yes	yes	no
ERRORATTRIBUTE	yes	yes	yes
FORMAT	yes	yes	no
LETTERS	yes	yes	no
LIBRARY	yes	yes	yes
LOG	yes	yes	no
LOGFILE	yes	yes	no
LOGPASSWORD	yes	yes	no
NORMSQL	yes	yes	no
NULL	yes	yes	no
SCHEMA	yes	yes	no
TEST	yes	yes	no
USERMSGFILE	yes	yes	yes

Time of evaluation

The following table contains an overview of the times at which the operands of the PARAMETER DYNAMIC statement are evaluated.

Operand	Compilation time	Runtime
AUTHORIZATION	yes, if not otherwise specified in OPTION	yes
CATALOG	yes, if not otherwise specified in OPTION	yes
DBSYSTEM	-	-
DECIMALSIGN	no	yes
ERRORATTRIBUTE	no	yes
FORMAT	no	yes
LETTERS	no	yes
LIBRARY	yes	yes
LOG	no	no
LOGFILE	no	no
LOGPASSWORD	no	no
NORMSQL	-	-
NULL	no	yes
SCHEMA	yes, if not otherwise specified in OPTION	yes
TEST	yes	no
USERMSGFILE	yes	yes

PARAMETER KFKEY
Assign K or F key

This application is valid

– in TIAM and UTM mode
– in interactive mode
– in program mode only in the body of an interactive program

PARAMETER KFKEY allocates K (short message) or F (function) keys.

This statement may be specified in menu-driven mode without the use of operands or with the use of at least one operand.

Without operands, PARAMETER FKEY can only be used in interactive mode.

Multiple operands specified within one statement take effect simultaneously. If an operand is specified in both interactive and program modes, the last specification applies.

If errors occur during execution, the entire statement is not executed and the invalid statement is displayed marked. In addition, an error message is output in the message line.

In TIAM applications, the key assignments can be corrected at any time. Invalid specifications are redisplayed at the terminal and can be revised.

A PARAMETER FKEY statement within a DRIVE program (compilation unit) has no effect on the compiler run for that program.

```
PARAMETER { KFKEY [=literal [ ACTION={ BREAK | EXIT | DELETE } ]
          [ UTMRC=literal ] ] } ...
```

KFKEY=literal

K or F keys are specified and this value is written to the system variable &KFKEY.

literal may assume the values K1 and K3 through K14, or F1 through F20. *literal* may also be specified in hexadecimal form.

The assignment specified in the system variable &KFKEY is available for processing in interactive programs until the next screen output is acknowledged with the DUE key. Following this acknowledgment, the system variable &KFKEY is again filled with blanks.

ACTION	The keys are assigned functions. No function may be assigned to the K2 key in TIAM mode. This key is used in TIAM applications to switch from DRIVE/WINDOWS to BS2000 system mode. In UTM mode, functions can only be assigned to the keys as part of the UTM start procedure. This assignment must be unambiguous, i.e. keys and UTM return codes may each only be input once. It is the user's responsibility to ensure that the key assignments are handled correctly and that the KDCDEF key assignment corresponds to the key assignment performed using PARAMETER in UTM mode. If, in program mode, a key is pressed for the runtime control of a DRIVE program although no function has been assigned to this key, the system variable &KFKEY is filled with blanks.
=BREAK	The key assigned with KFKEY is loaded with the BREAK function (see the BREAK statement). The default assignment for BREAK is the K1 key. This assignment can be changed by the user.
=EXIT	The key assigned with KFKEY is loaded with the EXIT function. When this key is pressed, the DRIVE run is aborted, and all active transactions are rolled back (see the EXIT statement).
=DELETE	The assignment for the key is deleted.
UTMRC=literal	The keys are loaded with UTM return codes. *literal* may assume a value from 20Z to 39Z. UTMRC=*literal* can only be used in the UTM start procedure where, in contrast, its psecification is obligatory.

Relationship to other statements

– In UTM mode, programs which were compiled with the compiler option OPTION OBJECT=ON may not contain the PARAMETER KFKEY statement. Instead, you may specify the KDCDEF control statement SFUNC together with a DRIVE parameter in the UTM start procedure (see DRIVE Programming System manual [1]).

– In TIAM mode, programs which were compiled with the compiler option OPTION OBJECT=ON may only contain the PARAMETER KFKEY statement if they are called with DO.

PARAMETER LOCK
Lock statement

This application is valid

– in TIAM and UTM mode
– in interactive mode
– in program mode only in the body of an interactive program

PARAMETER LOCK locks statements for the duration of a DRIVE run. The lock cannot be removed during the same run.

This statement may be specified in menu-driven mode without the use of operands or with the use of at least one operand.

Without operands, PARAMETER LOCK can only be used in interactive mode.

If you specify multiple operands within a statement, these become effective simultaneously. If an operand is specified in both interactive and program mode, the program mode specification applies.

If errors occur during execution, none of the statement is executed and the errored statement is marked and output. An error message also appears in the message line.

The statement PARAMETER LOCK PROCEDURE is evaluated when a program is run.

```
PARAMETER LOCK { DIALOG | PROCEDURE }

               { ALL | { statement={ ON | OFF } } ... }
```

DIALOG	The statements are locked for interactive mode.
PROCEDURE	The statements are locked for program mode.
ALL	All statements are locked for both interactive and program mode (except EXIT).
statement	The following statements can be locked: ACQUIRE ALTER TABLE BREAK [CYCLE | PROCEDURE | SUBPROCEDURE] CALL CASE

```
CLEAR
CLOSE { CURSOR | REPORT}
COMMIT
CONTINUE
CREATE { SCHEMA | TABLE | TEMPORARY VIEW | VIEW }
CYCLE [ CURSOR | FOR | WHILE ]
DEBUG
DECLARE { CONSTANT | CURSOR | FILE | FORM | LIST |
   REPORT | SCREEN | TYPE | VARIABLE}
DELETE
DETAIL
DISPATCH
DISPLAY [ FORM | LIST ]
DO
DROP { CURSOR(S) | SCHEMA | TABLE | TEMPORARY VIEW(S) | VIEW }
EDT
ENTER
EXECUTE
FETCH
FILL { FORM | LIST | REPORT }
GLOBAL
GRANT
GROUP
IF
INSERT
LIST
OPEN { CURSOR | REPORT }
PAGE
PARAMETER
PERMIT
PRINT
PROCEDURE
REPEAT
REPORT
RESTORE
REVOKE
ROLLBACK
SAVE
SELECT
SEND  MESSAGE
SET [CATALOG |  SCHEMA | SESSION | TRANSACTION ]
SHOW
SOURCE
STANDARD
```

STOP
STORE
SUBPROCEDURE
SYSTEM
UNSAVE
UPDATE
WHENEVER

=ON	This statement is locked for the user.
=OFF	Default The statement is permitted for the user.

PARAMETER STATIC
Specify static parameter

This application is valid

– n TIAM and UTM mode
– in interactive mode
– in program mode only in the body of an interactive program

PARAMETER STATIC is used to specify parameter values that remain in effect for the entire DRIVE run. The parameter values determine certain attributes of the DRIVE session relating to both interactive mode and to the environment in which the DRIVE programs run.

This statement may be specified in menu-driven mode without the use of operands or with the use of at least one operand.

Without operands, PARAMETER STATIC can only be used in interactive mode.

If you specify multiple operands within a statement, these become effective simultaneously. If you wish to specify operands in both interactive and program mode then you may only define operands which have not yet been entered.

If errors occur during execution, specifications that can no longer be reset may already have taken effect. Only operands that have not been previously supplied may be specified.

A PARAMETER STATIC statement within a DRIVE program (compilation unit) has no effect on the compiler run for that program.

```
PARAMETER STATIC [ FIRSTPAGE={ ON | OFF } |
                   FORMLIB=flib-name |
                   LASTPAGE={ ON | OFF } |
                   OLDSTYLE={ SESAM | SESAMSQL | LEASY | DMS } |
                   USER=username ] ...
```

FIRSTPAGE — When OFF is specified, output of the first page generated with LIST *, containing the list header, is suppressed.

Default: ON

FIRSTPAGE may only be specified in the UTM start procedure.

FORMLIB=flib-name — Name of the format library in which the FHS forms are stored.

Default: none

LASTPAGE	When OFF is specified, output of the last page generated with LIST *, containing the list footer, is suppressed. Default: ON LASTPAGE may only be specified in the UTM start procedure.
OLDSTYLE	Specifies the data management system which a DRIVE old style application accesses You can only specify OLDSTYLE in TIAM mode. Default: the loaded (new style) variant.
=SESAM	The DRIVE old style application accesses a SESAM V1.x database.
=SESAMSQL	The DRIVE old style application accesses a SESAM V2.x database.
=LEASY	The DRIVE old style application accesses LEASY files.
=DMS	The DRIVE old style application accesses DMS files.
USER=user-name	You may only specify USER in TIAM mode. *user-name* (max. 8 characters) is used to specify a user name. *user-name* may not contain the special characters / \ * ? [ ] or blanks (␣). *user-name* must be specified as a literal. You may assign *user-name* at any time provided that no SQL statement has been entered. After this, the TSN is entered for *user-name*. Default: blanks

Scope of application for operands

Operand	TIAM mode	UTM mode	UTM start procedure
FIRSTPAGE	no	yes	yes
FORMLIB	yes	no	no
LASTPAGE	yes	no	yes
OLDSTYLE	yes	no	no
USER	yes	no	no

Time of evaluation

The following table contains an overview of the times at which the operands of the PARAMETER STATIC statement are evaluated.

Operand	Compilation time	Runtime
FIRSTPAGE	no	no
FORMLIB	yes	yes
LASTPAGE	no	no
OLDSTYLE	no	yes
USER	yes	yes

PROCEDURE
Start program

This application is valid

– in TIAM and UTM mode
– in program mode

PROCEDURE is used to mark the start of a program. One program can receive parameters from another by means of PROCEDURE. For this, a variable must be defined in the USING clause for each parameter to be passed. If a program in which USING is followed by the RETURN statement is called in interactive mode, RETURN is ignored.

The parameters of the calling program are transferred individually to the variables of the successor program, after a variable has been defined in the successor program for each parameter. The number and format of the variables in the calling program and in the called program must be compatible. When parameters are passed, the length of all the parameters (definitions and values) must not exceed 31 Kbytes if the program was called with DO or ENTER.

The USING clause is not permitted in programs which are to be compiled with the DRIVE compiler DRIVE/WINDOWS-Comp if the program is intended as the main program in TIAM mode or as first TAC in UTM mode.

The declarative statements must come first in a program. Next, all the subprocedures must be defined. Only then does the program body follow.

If DRIVE/WINDOWS identifies syntax or semantic errors in a program, all elements defined up to that point in the program are reset.

The end of a program is defined with END PROCEDURE.

```
PROCEDURE prog-name

    [ USING { [ RETURN ] [ level ] var-name datatype }, ... ]
```

---

prog-name — Name of the program (max. 31 characters). This name need not be identical with the member name under which the source program is stored.

USING	USING is used to pass parameters to a program. USING must be specified if the USING clause is to be used when a program is called (with DO, CALL, ENTER or DEBUG).
RETURN	Marks the parameters that are to be returned by the called program to the calling program. The called program must be called with CALL or DEBUG. The corresponding parameter must also be identified with RETURN in the USING clause for CALL.
level	Level number. The level number must be 1.
var-name	The name of a simple variable. *var-name* must start with "&" and can be up to 32 characters in length. Simple variables, vectors, matrices, data groups and repeating groups can be defined. The value area of a variable, excluding the indicator variable area, must not exceed 32 Kb.
data-type	Data type of the variable. *data-type* must not contain an INIT or REDEFINES clause (see the DECLARE VARIABLE statement).

Example

The alphanumeric variable &cmp1, with 4 characters is passed to the program "staff1".

```
PROCEDURE staff1 USING &cmp1 CHAR(4);
```

READ FILE
Read a file

This application is valid

– in TIAM and UTM mode
– in program mode

READ FILE reads a data record at the current file position of a file which has been opened for read access and transfers this record to a variable or list of variables.

If the data record is longer than the variable or list of variables, DRIVE/WINDOWS truncates the record to the available length. The system variable &ERROR receives the entry "TOO LONG".

If the data record is shorter than the variable or list of variables, then:

– Numeric fields which cannot be filled in their entirety are assigned the null value. The system variable &ERROR contains the entry "TOO SHORT".
– Alphanumeric fields which cannot be filled are assigned the null value. The system variable &ERROR contains the entry "TOO SHORT".
– Fixed-length alphanumeric fields (CHARACTER) which cannot be filled in their entirety are assigned partial values and padded with blanks. The system variable &ERROR contains the entry "TOO SHORT".
– Variable-length alphanumeric fields (VARCHAR) which cannot be filled in their entirety are assigned partial values. The system variable &ERROR contains the entry "TOO SHORT".

After the record has been read, the new file position points to the next record.

In addition, DRIVE/WINDOWS enters the physical record length in the system variable &PHYS_REC_LENGTH and the DRIVE record length in the system variable &DRIVE_REC_LENGTH.

If the READ FILE statement does not find any characters in the file during an attempt to read the file, it has reached the end of the file. When the end of the file is reached, the system variable &ERROR contains the entry "OK END".

If a program accesses files which have been opened with the open mode UPDATE, INOUT or OUTIN, the READ FILE statement must not immediately follow the WRITE FILE statement. At least one file positioning statement (SET FILE POSITION) must occur between the READ FILE and WRITE FILE statements.

```
READ FILE file INTO variable, ...
```

file	Logical name of the file from which data is to be read. This must be the name used to declare the file in DECLARE FILE statement in the program.
variable	Name of the variable to which the read data record is transferred (see metavariable *variable*).

REMOVE
Cancel testpoint and operation

This application is valid

– in TIAM mode
– in debugging mode

REMOVE is used to cancel testpoints and operations at the testpoints set with AT. Counters declared with the AT ... COUNT statement are also removed.

If you do not specify an operation, then the specified testpoints are cancelled in their entirety.

```
REMOVE

   { [ library(member-name) | member-name ] { line  ... | line1 - line2 | ALL } |
     * }

   [ COUNT | DISPLAY | SET ]
```

---

library	Name of the DRIVE library (max. 54 characters) from which the program is read (applies only to external programs in DRIVE). *library* can also be the file link name of the DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name. If the DRIVE library has been preset with PARAMETER DYNAMIC LIBRARY then you may omit the *library* specification.
member-name	Name of the member (max. 31 characters) which contains the program. DRIVE/WINDOWS searches for the last member to have been processed, irrespective of whether this contains a source (S-member) or an intermediate code (X-member). If you do not specify *library* then the library specified in PARAMETER DYNAMIC LIBRARY is used.
*	The last testpoint entered is deleted. (see also the AT statement).

line	The specified *line* refers to a line number in the interpreter listing. All testpoints in this line are deleted. Several line numbers may be specified.
line1-line2	This specification refers to line numbers in the interpreter listing. All testpoints in the defined range are deleted. *line1* must be smaller than *line2*.
ALL	All testpoints set in the program are deleted.
COUNT	Only operations of the COUNT type are deleted.
DISPLAY	Only operations of the DISPLAY type are deleted.
SET	Only operations of the SET type are deleted.

REMOVE BOX
Remove dialog box

This specification is valid

– in UTM mode but not in asynchronous UTM mode and not in the receiving partner environment in DTP
– in program mode

REMOVE BOX removes one, several or all the dialog boxes which have been output. You cannot remove more dialog boxes than are output as otherwise DRIVE/WINDOWS aborts the program.

REMOVE BOX takes effect on the next screen output. REMOVE BOX behaves in different ways depending on the statement used to perform the next screen output:

DISPLAY screenform	All the dialog boxes which have been output are removed before the FHS form is output.
ADD BOX	The required number of output dialog boxes is removed before the new dialog box is output.
REPLACE BOX	The required number of output dialog boxes is removed before the replacement dialog box is output. (= total from the REMOVE BOX and REPLACE BOX statements).
SEND MESSAGE	If the message is output in a message box then the required number of output dialog boxes is removed before the message box is output. If the message is output in the message area of a partial form then all of the output dialog boxes are removed before the partial form is output.
DISPLAY form-name	All the dialog boxes which have been output are removed before the dynamic form is output.
DRIVE messages	All the dialog boxes which have been output are removed before DRIVE messages are output, for example when a program terminates or aborts.

If a further REMOVE BOX statement occurs before the next screen output, then all the dialog boxes specified in these statements are removed.

If you specify REMOVE BOX without operands then the last dialog box to have been output is removed.

```
REMOVE [ n | ALL ] BOX
```

n	The *n* topmost dialog boxes are removed. *n* must be a positive integer. *n* can be specified as a variable (see metavariable *variable*).
ALL	All the dialog boxes which have been output are removed.

REPEAT
Repeat statement

This application is valid

– in TIAM and UTM mode
– in interactive mode

REPEAT is used to redisplay the last statement entered. The user can then decide whether to execute or alter the statement.

In UTM mode, the statement buffer is reset to the last synchronization point after the ROLLBACK statement.

REPEAT only affects the following SQL statements:

– CREATE
– DECLARE
– DELETE { POSITIONED | SEARCHED }
– FETCH
– INSERT INTO
– SELECT
– UPDATE { POSITIONED | SEARCHED }

REPEAT

REPLACE BOX
Replace dialog box

This statement is valid

– in UTM mode but not in asynchronous UTM mode and not in the receiving partner environment in DTP
– in program mode

REPLACE BOX replaces one, several or all output dialog boxes with a new dialog box which you have already created in IFG (see IFG [28]). REPLACE BOX has the same effect as specifying the ADD BOX and REMOVE BOX statements together.

You cannot replace more dialog boxes than are output as otherwise DRIVE/WINDOWS aborts the program.

Any screen forms (partial forms or dialog boxes) which are not replaced continue to be displayed but are overlaid by the new dialog box and are locked, i.e. the user cannot make any input to these screen forms.

The last dialog box to be output is the current dialog box. Users can only input to the current dialog box.

If you specify REPLACE BOX without an operand then the last dialog box to be output is replaced.

```
REPLACE { n | ALL } BOX BY dialog-box
        [ POSITION ( line1 , column1 ) ] [ TO field1 ]
        [ CURSOR { POSITION ( line2 , column2 ) | TO field2 }
        [ MESSAGE key [ POSITION ( line3 , column3 ) | TO field3 ]
```

---

n	The *n* topmost dialog boxes are removed. *n* must be a positive integer. *n* can be specified as a variable (see metavariable *variable*).
ALL	All the dialog boxes which have been output are removed.
dialog-box	Name of the FHS-DE form (max. 7 characters). The form must have been created in IFG and have the property "Display in a box". The form must be defined in the declaration section of the program using the DECLARE SCREEN statement.

POSITION	Specifies the position of the dialog box. The position is specified using a starting or reference point The starting point is the first character (top left) of the dialog box. The reference point is the first character of *field1* if *field1* is specified or, otherwise, the top left-hand corner of the dialog box/FHS-DE partial form which is located below the current dialog box. If you do not specify POSITION or if you enter the value (0,0), DRIVE/WINDOWS attempts to position the dialog box with the default offset (+2,+2) to the reference point. If this is not possible, the dialog box is moved so that it fits on the screen. If you specify the POSITION but there is not enough space for the dialog box at the defined position, UTM aborts the operation with PEND ER.
line1	Line spacing between the reference point and the starting point of the dialog box. *line1* must be a whole number.
column1	Column spacing between the reference point and the starting point of the dialog box. *column1* must be a whole number.
field1	Field in the last FHS-DE form to be output (partial form and dialog box). *field1* must be a simple component of the associated screen variable. If *variable* is not a component of the screen variable of the last screen form to have been output, UTM aborts the conversation with PEND ER. The first 8 characters of the field names in the last screen form to have been output must differ in order to make the unambiguous assignment of screen variable components possible.
CURSOR	The cursor is set to a specific position in the dialog box. You may not specify CURSOR unless the global attribute “cursor position” was set for *dialogbox* in IFG (see IFG [28]).
POSITION	Specifies the absolute position (line/column) of the cursor.
line2	Line (1 ≤ *line2* ≤ number of screen lines). *line2* must be a whole number.
column2	Column (1 ≤ *column2* ≤ number of screen lines). *column2* must be a whole number.
TO	The cursor is set to the first character of field *field2*. In the case of lists, the cursor is set to the first column and first line of the list area.

field2	Field in the dialog box which is to be output. *field2* must be a component of the screen variable for *dialogbox*. The first 8 characters of the field names in the current dialog box must differ to make the unambiguous assignment of screen variable components possible.
MESSAGE	This outputs the FHS-DE message with the message key *key* which you created with IFG along with the dialog box. Depending on the IFG specification, output either takes place in a message box or in a message area in the dialog box. You may not specify MESSAGE unless the global attribute "Message Identifier" was set for *dialogbox* in IFG (see IFG [28]).
key	Message key of the FHS-DE message. You can specify *key* either as a variable (see the metavariable *variable*) or as an alphanumeric literal (see *char-literal* in the metavariable *literal*). *key* must be specified in the form AAAAnnn, where A is a letter (A-Z) and n is a digit (0-9). AAAA may not have the value IDHS.
POSITION	Specifies the absolute position of the message box. The message box is positioned with an additional offset (+2,+2) to *line3*, *column3*. POSITION is only evaluated if the IFG specification stipulates that the message is to be output in a message box. If a message is intended for output in a message box and you have not specified either POSITION or TO then the message box is output in the middle of the screen. If a message box which has been positioned using MESSAGE POSITION covers a cursor which has been set with CURSOR POSITION then MESSAGE POSITION is ignored.
line3	Line ($1 \leq line3 \leq$ screen lines). *line3* must be a whole number.
column3	Column ($1 \leq line3 \leq$ screen columns). *column3* must be a whole number.
TO	Specifies that the message box is to positioned with the default offset (+2,+2) to *field3*. TO is only evaluated if the IFG specification stipulates that the message is to be output in a message box. If a message is intended for output in a message box and you have not specified either TO or POSITION then the message box is output in the middle of the screen.

field3

Field in the dialog box which is to be output. *field3* must be a component of the screen variable for *dialogbox*.

The first 8 characters of the field names in the current dialog box must differ to make the unambiguous assignment of SCREEN variable components possible.

SAVE
Save EDT work file 0

This application is valid

– in TIAM mode
– in interactive mode

SAVE is used to store the contents of EDT work file 0 as a member of type S in a DRIVE library (e.g. source programs, copy members, user labels).

If a member of type S is already present in the DRIVE library under this name, the message `DRI0046 OVERWRITE 'member-name'? REPLY: (Y=YES, N=NO)` is output.

```
SAVE { library(member-name) | member-name }
```

---

library — Specifies the DRIVE library (max. 54 characters) in which the contents of EDT work file 0 are to be saved.

library may also be the file link name of the DRIVE library (in accordance with BS2000 conventions).

DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name.

If the DRIVE library has been predefined with the PARAMETER DYNAMIC LIBRARY statement, *library* need not be specified.

member-name — Specifies the member (max. 31 characters) in which the contents of EDT work file 0 are to be saved. This name need not be identical to the program name specified with PROCEDURE.

The member is stored in the specified DRIVE library. If no *library* is specified, the library that was specified in PARAMETER DYNAMIC LIBRARY is used.

SEND MESSAGE
Display message

This application is valid

– in TIAM mode
– in UTM mode but not in asynchronous UTM applications and not in the receiving environment of distributed transactions
– in program mode

The SEND MESSAGE statement can be used to display a message on the terminal.

The message is written to the message line (bottom line of screen). If the message is longer than the message line it is truncated and output followed by ">>>". The other screen contents are unchanged.

If a dialog box is displayed on screen when SEND MESSAGE is to be executed, the message is output in a dialog box which is provided by DRIVE/WINDOWS.

```
SEND MESSAGE { expression [ mask ] | BLANK n | TABULATOR n }, ...

             [ [ WITHOUT ] WAIT ]
```

---

expression	Message to be displayed.
mask	Specifies editing of the message (see the metavariable *mask*).
BLANK n	*n* blanks are output. *n* must be an unsigned integer. The message may not exceed 79 characters including blanks. If the message is longer, the following applies: if only one *expression* is output, the message is truncated and ended with the character string ">>>". If more than one expression is output, the DRIVE program is aborted with an error message.
TABULATOR n	Specifies the column position where the cursor is to be set. A space between the message and the cursor position is filled by blanks. *n* must be an unsigned integer greater than zero. An error message is output if the current column position is greater than the specified TABULATOR position.

WAIT	After outputting the message, DRIVE/WINDOWS waits for input from the data display terminal (e.g. RETURN key). The next DRIVE statement is not executed until this input has been received.
WITHOUT WAIT	Valid in TIAM mode only. WITHOUT WAIT is ignored when a SEND MESSAGE follows a DISPLAY screen-form. After the message has been output, the next DRIVE statement is immediately executed. WITHOUT WAIT is not useful if other output directly follows the message.

Example

The message "List is being printed" is output in the message line.

```
SEND MESSAGE 'List is being printed';
```

SET
Assign value and field attribute

This application is valid

– in TIAM and UTM mode
– in program and debugging mode

SET can be used to assign values to variables. Each variable may be assigned only one value. If defined with DECLARE SCREEN, a variable (screen variable) may be assigned field and global attributes. If it is simultaneously assigned a value and a field attribute, the variable must be (part of) a screen variable.

There are 3 variants of the SET statement:

1. A value, the null value and/or field attributes are assigned to a variable.

```
SET variable { = { expression | NULL } [ NOCHECK ]
                                [ [ WITH ] ATTRIBUTE ( attribute1, ... ) ] |
                                [ WITH ] ATTRIBUTE ( attribute1, ... ) }
```

2. In the second SET variant, global attributes are assigned to all the fields of a screen variable.

```
SET { screen-form [ WITH ] ATTRIBUTE ( attribute2, ... ) }, ...
```

3. In the third SET variant, field attributes are assigned to all the fields of a screen variable whose field value EDIT_STATE≠"V" (field not error-free).

```
SET { screen-form [ WITH ] ERRORATTRIBUTE ( attribute3, ... ) }, ...
```

variable
Name of a variable to which a value or local field attribute is assigned.
The data type of *variable* must be compatible with the data type of *expression*.

If a structured variable is assigned a value that is also structured, the assignment is made component by component. The individual components must be of the same data type.

If there are redefinitions in a structured variable, only the components that are not redefined receive the value.

If only a single value is assigned to a structured variable, all components receive that value.

expression
Value assigned to the variable.

The data type of *expression* must be compatible with the data type of *variable*.

When values are assigned to numeric variables, commercial rounding is applied to decimal places if the variable declarations do not permit sufficient decimal places.

NULL
The variable is assigned the null value.

NOCHECK
When a value is assigned to *variable*, any CHECK clause applicable to *variable* is ignored. If *variable* is structured, NOCHECK is valid for all component assignments.

WITH ATTRIBUTE
Assigns field attributes to a screen variable.

attribute1
Field attribute. You can assign the following values:

- MUST, NORMALINPUT, POTMUST
- UNPROTECTED, PROTECTED
- HIGHINTENSITY, NORMALINTENSITY
- VISIBLE, SIGN, INVISIBLE
- UNDERLINE, NOUNDERLINE
- INVERSE, NOINVERSE
- BLUE, CYAN, GREEN, MAGENTA, RED, WHITE, YELLOW, NOCOLOUR
- CURSOR, NOCURSOR
- VALID, INVALID

SET ... ATTRIBUTE (attribute1,...) is not permitted in debugging mode.

screen-form
: Name of a form corresponding to the associated screen variable in which the FHS form is displayed and to which global attributes are assigned.

WITH ATTRIBUTE
: Assigns global attributes to all the fields of a screen variable.

attribute2
: Global attribute. You can assign the following values:

  – ALARM
  – HARDCOPY
  – INIT, NOINIT
  – CURSOR, NOCURSOR

  An additional possible value is DEFAULT. If it is specified, all field attributes are reset to their predefined status (blanks). In a combination of global attributes, DEFAULT must appear first in the list.

  In debugging mode, only DEFAULT may be specified for *attribute2*.

WITH ERRORATTRIBUTE
: Assigns field attributes to the fields of a screen variable which have the field value EDIT_STATE="F" or "M"

attribute3
: Field attribute. You can assign the following values:

  – MUST, NORMALINPUT, POTMUST
  – UNPROTECTED
  – HIGHINTENSITY, NORMALINTENSITY
  – VISIBLE, SIGN, INVISIBLE
  – UNDERLINE, NOUNDERLINE
  – INVERSE, NOINVERSE
  – BLUE, CYAN, GREEN, MAGENTA, RED, WHITE, YELLOW, NOCOLOUR
  – CURSOR, NOCURSOR

Examples

The variable &index is incremented.

```
SET &index = index + 1;
```

The current date in the format "*day-month-year*" is assigned to the variable &date.

```
SET &date = CURRENT DATE
```

Each of the three fields of the vector &language(3) is assigned the null value.

```
SET &language = NULL
```

The global attribute CURSOR is assigned to all the fields of the FHS partial form “mask”.

```
SET mask ATTRIBUTE (CURSOR)
```

An incorrect field in the FHS partial form “mask” is displayed in high intensity video.

```
SET mask ERRORATTRIBUTE (HIGHINTENSITY)
```

SET FILE POSITION
Position within a file

This application is valid

– in TIAM and UTM mode
– in program mode

SET FILE POSITION positions to the start, to the end or to any position previously read with the GET FILE POSITION statement in an open file.

In ISAM files, use the LOCATE FILE statement with the ISAM key to locate the file position.

```
SET FILE POSITION file TO { variable | BEGIN | END }
```

---

file	Logical name of the file in which the position is to be set. This must be the name which was used in the DECLARE FILE statement to declare the file in the program.
TO	Specifies the required position.
variable	Indicates the point to which the position is to be set (see the metavariable *variable*). *variable* must already have been assigned a value using the GET FILE POSITION statement.
BEGIN	Positions to the start of the file.
END	Positions to the end of the file.

SET SCREEN ATTRIBUTE
Assign form attribute

This statement is valid

– in UTM mode
– in program mode

SET SCREEN ATTRIBUTE assigns attributes to FHS-DE forms. You can make the following specifications:

– the number of lines to be output in list areas
– which page or scroll command should be preset for the next form output
– which selection items in selection fields should be locked on output
– which selection items in selection fields should be preselected on output
– which lines in list areas should be preselected on output

```
SET SCREEN ATTRIBUTE

   { { { LOCK | PRESELECT }  { ON | OFF } [ ITEM ( i, ...) ] } |
     LINES n |
     SCROLL char-expression }

     FOR { screen-form | field }
```

LOCK	Selection items are locked (ON) or unlocked (OFF). *field* must contain the component of a screen variable.
PRESELECT	Selection items or lines are identified as preselected (ON) or not preselected (OFF).
ON	Selection items/lines are identified as locked (in the case of LOCK) or preselected (in the case of PRESELECT).
OFF	Selection items/lines are identified as unlocked (in the case of LOCK) or not preselected (in the case of PRESELECT).

ITEM	in the case of LOCK: locks/unlocks the *i*-th selection item of a single selection field. LOCK ... ITEM is not permitted in multiple selection fields or list areas. If ITEM is not specified for single selection fields then all the selection items are either locked or unlocked. in the case of PRESELECT: identifies the *i*-th selection item of a single selection field or the *i*-th line in a list area. PRESELECT ... ITEM is not permitted in multiple selection fields. ITEM must be specified for single selection fields. However, only one selection item *i* may be specified. If ITEM is not specified for list areas then all the lines are identified as preselected.
i	Selection item/line ($i \leq$ number of selection items or list lines). *i* must be a positive integer. *i* can be specified as a numeric expression (see metavariable *value-expression*).
LINES	Lines in the *screen-form* list area are output.
n	Number of lines to be output ($n \leq$ maximum number specified for the list area with IFG). *n* must be a positive whole number.
SCROLL	Defines the scroll information characters
char-expression	Characters for scroll information. The characters + – < > and the blank (␣) are permitted for *char-expression*. If *char-expression* consists of blanks only, no scroll information is output.
screen-form	FHS-DE form with a list area. The FHS-DE form must be declared in the program using DECLARE SCREEN.
field	Selection field for which an attribute is set. The field must be part of an FHS-DE form (= component of a SCREEN variable). The associated FHS-DE form must be declared in the program using DECLARE SCREEN.

STOP
Terminate DRIVE run

This application is valid

– in TIAM mode
– in UTM mode, but only at the highest programming level in the receiving partner environment in DTP
– in interactive and program mode

STOP is used to terminate the DRIVE run.

In TIAM mode all the requested files are closed and all views and DRIVE-specific memory areas released.

If STOP or STOP WITH DISPLAY is entered in TIAM mode, and EDT work file 0 contains a file that has not been saved, the message
`"EDT WORK FILE 0 NOT EMPTY. TERMINATE 'DRIVE'? (Y=YES;N=NO)"` is displayed.

– If the response is "N", STOP is not executed. The SAVE statement is displayed so that the file can be saved.
– If the response is "Y", STOP is executed. Changes made to the unsaved file are lost.

In UTM mode all conversation lists are printed and are deleted in the central print file unless they are explicitly printed using LIST * ... DELETE. A transaction code (TAC) or KDCOFF must then be entered (only in the submitting partner environment in the case of DTP).

STOP causes a DRIVE error message if transactions are still open. DRIVE execution is not terminated. You must terminate open transactions with COMMIT, ROLLBACK or EXIT. EXIT terminates DRIVE execution.

Restrictions

– In local operation, STOP is only permitted in interactive mode whereas in program mode it is only permitted in an interactive program.

– STOP WITH DISPLAY is only permitted in an interactive program.
STOP WITH DISPLAY is not permitted in asynchronous UTM mode or in the receiving partner environment in DTP.

– STOP WITH *char-expression* is not permitted in asynchronous UTM mode or in the receiving partner environment in DTP.

```
STOP [ WITH { DISPLAY form-name |
            DISPLAY screen-form, ... [ SCREENERROR { REPEAT | CONTINUE } ] |
            DISPLAY form |
            char-expression } ]
```

DISPLAY form-name	*form-name* is displayed at the terminal (see the DISPLAY *form-name* statement).
DISPLAY screen-form	*screen-form* is displayed at the terminal (see the DISPLAY *screen-form* statement).
SCREENERROR	SCREENERROR determines the behavior of DRIVE/WINDOWS when valid field input is made.
REPEAT	Default The screen mask is redisplayed until the input is correct or an intentional abort with BREAK occurs. Execution of the STOP statement is then continued.
CONTINUE	Execution of the STOP statement continues following DISPLAY even if input is invalid.
DISPLAY form	Defines and outputs a compact screen form (see the DISPLAY FORM statement).
char-expression	Name of the UTM follow-up TAC (max. 8 characters). *char-expression* can only be specified in UTM mode and program mode. *char-expression* is not permitted in asynchronous UTM mode or in the receiving partner environment in DTP. If the UTM follow-up TAC does not exist, the program is aborted but the conversation is not terminated.

Rules for distributed transaction processing

– When STOP is issued, all receiving partner activities must be terminated in the receiving partner environment.

– In the receiving environment, STOP returns control to the submitting activity along with the appropriate RETURN parameters.

SUBPROCEDURE
Start internal subprogram

This application is valid

– in TIAM and UTM mode
– in program mode

SUBPROCEDURE is used to mark the start of an internal subprogram. An internal subprogram is a statement sequence defined within a program in the declaration section, following the DECLARE statements. Nested definitions of internal subprograms are not permitted. An internal subprogram may not itself contain any DECLARE statements.

Internal subprograms may be called as often as desired within a program. The call is made with CALL *subprog-name*.

Internal subprograms may only be called if they have previously been defined.
All structuring statements (CASE, CYCLE, DISPATCH, IF) must be terminated in an internal subprogram, i.e. the corresponding END must be included in the same internal subprogram. An END for a structuring statement may not be specified outside that internal subprogram.

A BREAK CYCLE cannot terminate a loop begun outside that internal subprogram. A BREAK PROCEDURE terminates both the internal subprogram and the main program. When a BREAK SUBPROCEDURE is encountered, the program is continued with the statement following the CALL for that internal subprogram.

The end of an internal subprogram is defined by END SUBPROCEDURE.

SUBPROCEDURE subprog-name

---

subprog-name Name of the internal subprogram (max. 31 characters).

SYSTEM
Enter BS2000 command

This application is valid

– in TIAM mode
– in interactive and program mode

You use the SYSTEM statement to enter BS2000 commands during DRIVE operation.

```
SYSTEM char-expression
```

---

char-expression

Name of the BS2000 command which you want to execute.

char-expression must be an alphanumeric string (max. 254 characters). See the metavariable *char-expression*.

The result of *char-expression* must be a syntactically correct BS2000 command.

Examples

```
SYSTEM 'PRINT-FILE *SYSLST'

SYSTEM 'SHOW-USER-STATUS'

SET &FS='SHOW-FILE-ATTRIBUTES'
SYSTEM &FS
```

TRACE
Activate trace

This application is valid

– in TIAM mode
– in debugging mode

The TRACE statement is used in debugging mode to activate a program trace. This means that the program is executed and controlled in single steps, and the associated lines of the interpreter listing are output.

The TRACE statement defines the number of program statements to be traced. The program stops when the tracepoint is reached, i.e. after the declared number of statements, assuming that the statements were executed without an error. The trace function is then automatically deactivated, and the tracepoint is deleted.

The parameters specified for a TRACE statement serve as default values for the next TRACE statement if that statement is entered without operands.

DRIVE/WINDOWS always inserts user-defined tracepoints after a statement of the DRIVE program.

TRACE [n | ALL] [OUT | LIST | BOTH]

---

n	*n* is a positive whole number. The next *n* statements are executed and monitored in single-step mode and the corresponding lines from the interpreter listing(s) are output (trace output). If a breakpoint is reached before *n* statements have been executed then tracing is deactivated and the tracepoint is deleted. Default on first TRACE call: *n*=1. If you do not specify *n* in subsequent TRACE calls then the value of *n* is taken over from the last TRACE statement.
ALL	All the statements up to the next breakpoint are executed and monitored in single-step mode and the corresponding lines from the interpreter listing(s) are output (trace output).

OUT	Default Specifying OUT displays trace output on screen. Default on first TRACE call: OUT. If you do not specify the operand in subsequent TRACE calls then the value of the operand is taken over from the last TRACE statement.
LIST	Specifying LIST sends trace output to SYSLST.
BOTH	Specifying BOTH has the same effect as specifying LIST and OUT together.

Relationship to other statements

If the DEBUG statement accesses a program that is in EDT work file 0, tracing cannot be activated.

UNSAVE
Delete program, COPY member or user label

This application is valid

– in TIAM and UTM mode
– in interactive and program mode

UNSAVE deletes objects from the DRIVE library.

– Sources, COPY members, user labels (S-members)
– Intermediate codes (X-members)
– Object codes (R-members)
– Interpreter listings (P-members)

The specified objects are deleted in the following sequence:

1. Usage references
2. Intermediate codes
3. Object codes
4. Interpreter listings
5. Sources
6. COPY members

If an error occurs when several objects are being deleted, an error message is issued indicating the object concerned. The preceding objects will have been properly deleted, but not the following ones.

If one of the individual objects specified does not exist, this is considered to be an error.

If you specify UNSAVE without any optional operands, then the sources, intermediate codes, object codes, interpreter listings and usage references which are stored under the name *member-name* are deleted. DRIVE/WINDOWS does not issue an error message if any individual object does not exist (except in the case of an S-member).

Each of the operands SOURCE, OBJECT, CODE, LIST, COPYSOURCE and USERLABEL may be specified once only in each statement. You may combine different operands.

The UNSAVE statement is not permitted in programs which are to be compiled with the DRIVE compiler DRIVE/WINDOWS-Comp.

```
UNSAVE { library(member-name) | member-name }

       [ { SOURCE | CODE | OBJECT | LIST | COPYSOURCE | USERLABEL }, ... ]
```

library	Specifies the DRIVE library (max. 54 characters) from which a member or usage reference is deleted. *library* may also be the file link name of the DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library* first as a file link name, then as a library name. If the DRIVE library has been preset with PARAMETER DYNAMIC LIBRARY then you may omit the *library* specification.
member-name	Specifies the member (max. 31 characters) that is deleted. If you do not specify *library*, the library specified in PARAMETER DYNAMIC LIBRARY is used.
SOURCE	The DRIVE program (S-member) in the DRIVE library is deleted.
CODE	The intermediate code (X-member) in the DRIVE library is deleted.
OBJECT	The object code (R-member) in the DRIVE library is deleted.
LIST	The interpreter listing (P-member) in the DRIVE library is deleted.
COPYSOURCE	The COPY member (S-member) in the DRIVE library is deleted.
USERLABEL	The user label (S-member) in the DRIVE library is deleted.

WHENEVER
Define error exit

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) 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 (see the DRIVE Programming Language [2] and the section on system variables).

If no error exit is defined, DRIVE/WINDOWS aborts the program. (Exception: the program is continued with the &ERROR entries "OK END", "TOO LONG" and "TOO SHORT" and with the &DML_STATE entries "TABLE END" and "DIRTY READ".)

```
WHENEVER { &KFKEY [ IN ( literal, ... ) ] |
           &ERROR [ IN ( error, ... ) ] |
           &DML_STATE [ IN ( status, ... ) ] }

         { 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/WINDOWS 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*.

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=**
`SET &v=&a(&i)`	INDEX ERROR	`'INDEX ERROR'`	unchanged
`OPEN` *cursor-name*	SQL ERROR	unchanged	`'SQL ERROR'`
`FETCH` *cursor-name* `INTO ...`	DIRTY READ	`'OK'`	`'DIRTY READ'`

WRITE FILE
Write to a file

This application is valid

– in TIAM and UTM mode
– in program mode

WRITE FILE writes a data record to a file which has been opened for write access. The structure and content of this record are defined in the WITH clause.

In SAM files, the record is written at the current file position. If an existing record is to be overwritten, the new record must be of the same length as the record which is to be overwritten.

In ISAM files, the record is written at the position given by the ISAM key. The ISAM key forms part of the record data.

If the record which is to be written is longer than the permitted BS2000 record length, then the system variable &ERROR receives the entry "TOO LONG".

After the write operation, the current file position points to the next record.

If a program accesses files which have been opened with the open mode UPDATE, INOUT or OUTIN, the WRITE FILE statement must not immediately follow the READ FILE statement. At least one file positioning statement (SET FILE POSITION) must occur between the WRITE FILE and READ FILE statements.

```
WRITE FILE file WITH { expression | NULL }, ...
```

---

file	Logical name of the file to which data is to be written. This must be the name used to declare the file in DECLARE FILE statement in the program.
expression	Designates the expression which is written to the file (see metavariable *expression*). *expression* describes the data record.
NULL	The character defined for representing null values is written to the file (see the DECLARE FILE statement).

Special file attributes

– The file attributes match the default settings in the BS2000 operating system. If you are working with file attributes which do not correspond to these defaults (e.g. longer records), you must use the corresponding ADD-FILE-LINK command to define the file attributes. You must then use the file link name to address the file in question (SET-FILE-LINK... LINK-NAME=)

  You must enter the ADD-FILE-LINK and SET-FILE-LINK commands:

  – in TIAM mode, before the start of the DRIVE program or in the SYSTEM statement in the DRIVE program itself. The SYSTEM statement must precede the OPEN-FILE statement.
  – in UTM mode, before the start of the DRIVE statement.

4 Report statements

The beginning of this chapter contains an overview of all report statements. This is followed by a list of the DRIVE statements permitted in report generation. A description of the restrictions applicable to report parameters and of the report set functions then follows.

In addition, all report statements are described in detail in alphabetical order in this chapter.

For information on using the report generator, refer to the DRIVE Programming Language manual [2].

Report statements are valid

– in TIAM mode

– in UTM mode; however, no screen input or output may occur between OPEN REPORT and CLOSE REPORT

– in programming mode and debugging mode. In debugging mode, no breakpoints may be set between DECLARE REPORT and END REPORT

Summary of report statements

Two steps are required for the generation of a report:

1. Defining the report
2. Executing and creating the report

All statements regarding report definition are in the declaration section of a DRIVE program. The statements for report execution are placed in the body of a DRIVE program.

The start and end of a report definition are identified by the following statements:

DECLARE REPORT	Start report definition, see page 218
END REPORT	End report definition, see page 226

The statements between DECLARE REPORT and END REPORT are used for describing the data and the layout. The following report statements can be used for this purpose:

DECLARE VARIABLE	Define report variable, see page 223
STANDARD LAYOUT	Describe layout of a standard report, see page 266
GLOBAL LAYOUT	Describe layout of an individual report, see page 229
GLOBAL LINE BASE	Define line background, see page 233
GLOBAL PAGE BASE	Start definition of page background, see page 235
PAGE PRINT	Describe page background, see page 245
OVERLAY PAGE BASE	Activate page background, see page 244
REPORT	Define list control block, see page 244
PAGE	Define page control block, see page 245
GROUP	Define group control block, see page 236
DETAIL	Define detail control block, see page 224
PRINT	Define report output, see page 252
SOURCE	Insert text files, see page 265

To execute a report, the following report statements are available:

OPEN REPORT	Start report execution, see page 239
FILL REPORT	Supply data to report, see page 227
CLOSE REPORT	End report execution, see page 217

Permitted DRIVE statements

The following DRIVE statements are permitted in report generation:

BREAK	Break a cycle. This statement is only allowed in the following form: BREAK CYCLE
CALL	Call C-modules. The CALL statement is only allowed in the following form: CALL C MODULE ... RETURN No null value indicators are transferred. INDICATOR **must not** be specified.
CYCLE	Program a cycle. The statement CYCLE *cursor-name* is meaningless in report generation because there is no cursor in the report. The statement CYCLE FOR is **not** permitted.
IF	Program conditions. Only the following comparisons are allowed for *condition* in an IF statement: – expressions with comparison operators – columns with the null value. (see the metavariable *condition*).
SET	Assign values. In a SET statement only the following entry is permitted: SET *variable* ... Attributes **must not** be assigned. Assignments are only permitted for start parameters and local variables.

Restrictions applicable to report parameters

If variables, system variables, mask control characters, expressions and transfer parameters are used in report or DRIVE statements included in a report definition, the following restrictions apply:

Literals

Literals of type INTERVAL must be entered to an accuracy of one second.

Variables

The variables declared within a report definition only apply locally. The variables must be simple.

The following **must not** be used:

- the types TIME(3) and TIMESTAMP(3)
- INIT, CHECK or REDEFINES clauses
- LIKE clauses
- variables declared outside the report definition.

System variables

Only &LINES and &PAGES can be used as system variables. They apply locally within the report definition.

Mask control characters

The following mask control characters **must not** be used:

ZZZY, ZI, ZS, ZW, BWZ, YYY, ZZY, JJJ, ZZJ.

Also **not** permitted are CHAR masks.

Names are abbreviated if Q(3), QQQ or R(3), RRR is entered. Any other input of Q or R causes the names to be output in full (see the metavariable *mask*).

Expressions

Only the following functions are allowed in expressions:

- CURRENT DATE,
- CURRENT TIME

(see the metavariables *char-prim* and *format*).

The exponential operator (**) is **not** permitted.

It is recommended that you specify masks for the output of arithmetic operations and report set functions.

Transfer parameters

Parameters passed in the USING clause must not be modified. They must not precede the equals sign in the SET statement, for example.

Report set functions in expressions

Expressions can contain the following report set functions:

```
{ COUNT | MIN | MAX | AVG | SUM }
([ REPORT | PAGE | GROUP ] [ GROUPNUM n ] [ TOTAL ] [ ALL ] [ DISTINCT ] expression)
```

where the parameters have the following meanings:

COUNT — determine the number of *expression*

MIN — determine the minimum number of *expression*

MAX — determine the maximum number of *expression*

AVG — determine the average of *expression*

SUM — determine the sum of *expression*.

The scope of application of the above-mentioned functions can be specified with:

REPORT — For the entire report.

The derived values are updated each time the fields *expression* occur and are not reset until the end of the list.

PAGE — For one page.

The calculation of the values derived from *expression* starts anew with each new page. The current values of the report set function are provided

- in all control blocks of the current page, with the exception of the page header
- on the following page in the page header for transfer of the values.

The derived values are reset after the page header and before the first group or detail control block.

GROUP — For one group.

The calculation of the values derived from *expression* begins with the first report data of a group level. The current values of a group-specific report set function are available from the group header to the group trailer of the current group level. The derived values are reset after the assigned group trailer.

If REPORT, PAGE or GROUP is omitted, a report set function refers to the control block in which the function is specified.

The scope of application of a report set function can be specified in even greater detail with:

GROUPNUM n	For a specific group. The report set function only refers to the group with the number *n*. The group number *n* is defined in the statement GROUP (see the GROUP statement). *n* must not be greater than 32767.
TOTAL	Specified in advance for all field contents of the control block for which the set function has been defined. This advance calculation makes it possible to use the calculated value within the control block itself, for example in expressions.
ALL	For all field contents. Regardless of whether the field contents of *expression* are output or not, the report set functions are applied to all field contents of *expression*. By default, only the values to be output are used for calculation.
DISTINCT	For certain field contents. Only the field contents of *expression* which differ from the contents of the field in the previous record are used for calculation.

CLOSE REPORT
End report execution

CLOSE REPORT ends report execution. It causes the report generator to close the report buffer which was opened by the associated OPEN REPORT statement together with the employed report definition (see OPEN REPORT statement).

After closing the report definition and the report buffer, the report is generated on the output device specified by OPEN REPORT.

A CLOSE REPORT statement is only permitted in the body of a program.

```
CLOSE REPORT report-name [ variable ]
```

---

report-name	Specifies the name of the report definition used. The name must be declared with a DECLARE REPORT statement (see the DECLARE REPORT statement). The corresponding report must be open (see OPEN REPORT statement).
variable	Designates the variable or variable component which was used to identify the report buffer in the OPEN REPORT statement (see OPEN REPORT statement).

DECLARE REPORT
Define report

DECLARE REPORT identifies the beginning and END REPORT the end of a report definition.

A report definition is assigned its name with the DECLARE REPORT statement. The report definition is referred to under this name when the report is executed.

The parameters to be used in a report definition must be defined with the DECLARE REPORT statement. These parameters are supplied with values during execution of the report (see the FILL REPORT statement). When defining these parameters, it is possible to take different record types into consideration.

Moreover, start parameters can be specified to which the report generator assigns values once only at the start of report execution (see the OPEN REPORT statement).

Variables which are defined in the DECLARE REPORT statement do not appear in the cross-reference list.

DECLARE REPORT is only allowed in the declaration section of a program. More than one report can be defined in a program.

```
DECLARE REPORT report-name
    [ FOR START USING { var-name1 basic-data-type [ mask ] }, ... ]

    { USING { [ level ] var-name2 { data-def |
                                LIKE { CURSOR cursor | TABLE table } } }, ... |

      { RECORD TYPE char-literal USING { [ level ] var-name2 { data-def |
                          LIKE { CURSOR cursor | TABLE table } } }, ... }, ... }
```

---

report-name — Specifies the name of a report definition. The name must be unambiguous and consist of a maximum of seven characters. Since the report definition name is used at compilation or runtime to create filenames, it must comply with the BS2000 conventions for filenames.

FOR START USING — Defines start parameters to which the current values are passed once at the start of report execution using the OPEN REPORT statement. These values may, for example, be output or be used for layout control in expressions which contain the results of report set functions.

	The number of start parameters used must correspond to the number of parameters specified in the USING clause of the OPEN REPORT statement (see OPEN REPORT statement). If the matching parameters in the statements DECLARE REPORT and OPEN REPORT have different *basic-data-types*, their values must be convertible to the appropriate type. The total length of the data values including null value displays may not exceed 31 KB.
var-name1	Specifies a simple variable of type *basic-data-type*. If more than one report is defined in a program, the names of the report parameters must be unique for all reports. The names of report parameters may be identical to the names of DRIVE variables. They are nonetheless handled differently.
basic-data-type	Specifies the type of the variable *var-name1*. *basic-data-type* can be a type defined by the user, i.e. *user-type*, or one of the following types: CHARACTER, DECIMAL, NUMERIC, INTEGER, SMALLINT, DATE, TIME, INTERVAL, CHARACTER VARYING, VARCHAR, REAL, DOUBLE PRECISION, FLOAT You may **not** use the types TIME(3) and TIMESTAMP(3) or INIT, CHECK and REDEFINES clauses and you may also not use LIKE clauses in report parameters.
mask	Designates a display option (mask control character) for the output of a start parameter *var-name1*. The *mask* specification is subject to the restrictions described elsewhere (see section “Restrictions applicable to report parameters” on page 214).
USING	Defines parameters which are supplied with data by means of the FILL REPORT statement during report execution ("net data"). The number of parameters must correspond to the number of expressions in the USING report clause of the FILL REPORT statement (see the FILL REPORT statement). If the corresponding parameters in the statements DECLARE REPORT and FILL REPORT have different types, then their values must be convertible to the appropriate type. The total length of the data values including null value displays may not exceed 31 KB.
level	Level number. The level number must be 1.

var-name2	Specifies a variable of type *data-def* or of a type defined in a LIKE clause (see the DRIVE statement DECLARE VARIABLE). So many variables may be specified that the length of the values of all parameters including the null indicator does not exceed 31 Kbytes.
data-def	Specifies the type of the variable *var-name2*. (see metavariable *data-def*). The following specifications are **not** permitted for *data-def*: INIT, CHECK, REDEFINES clauses, and LIKE clauses for report parameters. The data types TIME(3) and TIMESTAMP(3) are **not** permitted.
LIKE	Copies the structure of a cursor or table component by component to the variable *var-name2* (see the DRIVE statement DECLARE VARIABLE).
cursor	Specifies a cursor which must be declared in the DRIVE program.
table	Specifies a base table which must be declared in a connected database.
RECORD TYPE	Defines the record type *char-literal*. The structure of the record type is described in the associated USING clause. More than one record type can be defined (see the DETAIL statement).

Example 1

In the report declaration the variable &s2, which corresponds to the variable &kd_name, is defined as a start parameter. Before the report is opened, the name 'AGENT' is assigned to the variable &kd_name (customer name). This value is passed when the report is opened.

```
PROCEDURE "t1.2007";

DECLARE VARIABLE &12      VARCHAR(256) INIT '9012-2';
DECLARE VARIABLE &v11     DEC (8,2);
DECLARE VARIABLE &repv    CHAR (8);
DECLARE VARIABLE &kd_name CHAR (10);

DECLARE REPORT drirep FOR START
                      USING &s2 CHAR (10)
                      USING &repdat,
                              2 key        CHAR (6),
                              2 artname    CHAR (20),
                              2 price      NUM (8,2),
                              2 stock     NUM (5);
...
END REPORT;

SET &kd_name='AGENT';

OPEN REPORT drirep USING &kd_name RESULT list 'G207' DEVICETABLE &12;
...
CLOSE REPORT drirep;
```

Example 2

The report definition is declared with the name "statist". The data of the &data parameter, which was defined as the variable &cvar in the DRIVE program, are processed in this report definition:

```
...
/* Variable which is subsequently passed to the report buffer    */
/* during processing                                             */

DECLARE VARIABLE  1 &cvar
                    2 agent           CHAR (20),
                    2 article         NUM (6),
                    2 quantity        NUM,
                    2 unit-price      NUM,
                    2 amount          NUM;

/* Declaration of report                                         */

DECLARE REPORT statist USING &data LIKE &cvar;

   /* Description of data                                        */
   /* Description of layout                                      */

END REPORT;
...
```

DECLARE VARIABLE
Define report variable

DECLARE VARIABLE is used to declare report variables. These variables only apply locally within the report definition. A local variable can, for example, be used in expressions for controlling the layout.
Report variables do not appear in the cross-reference list.

The system variables &PAGES and &LINES are available implicitly in the following form:

```
DECLARE VARIABLE &PAGES INTEGER,
                 &LINES INTEGER;
```

These system variables are initialized with the null value and need not be explicitly declared with the DECLARE VARIABLE statement. Values cannot be assigned. The system variables &PAGES and &LINES are assumed to be local variables within the report definition.

```
DECLARE VARIABLE var-name basic-data-type [ mask ]
```

---

var-name	Specifies a simple variable of type *basic-data-type* which is initialized with the null value.
basic-data-type	Specifies the type of the variable *var-name*. *basic-data-type* can be one of the following types: CHARACTER, INTEGER, SMALLINT, DATE, TIME,INTERVAL, CHARACTER VARYING or VARCHAR, REAL, DOUBLE PRECISION, FLOAT, XDEC or EXTENDED DECIMAL, DECIMAL, NUMERIC You may **not** use the types TIME(3) and TIMESTAMP(3).
mask	Designates a display option (mask control character) for the output of a variable *var-name* (see metavariable *mask*). The *mask* specification is subject to the restrictions described elsewhere (see section “Restrictions applicable to report parameters” on page 214).

Unlike DRIVE variable declarations, report variable declarations may not contain any INIT, CHECK, REDEFINES or LIKE clauses.

DETAIL
Define detail control block

DETAIL is used to describe the detail lines. The statements which control the processing of the data records to be output are included in this control block. For example, the positions where constant text parts and variable data are to be inserted are defined relative to the current output position in the detail control block.

The DETAIL statement is only allowed within a report definition. If different record types are to be processed, a separate detail control block must be defined for each record type.

If more than one record type exists, the report generator identifies a record type with a character string and the contents of an identifier field. If the identifier field adopts the value of the character string, the program branches to the corresponding detail control block. The identifier field of a record type must be in the same position in all record types. The first field of a data record is preset as the identifier field.

If a record type other than the one defined for the control block is referenced in a detail control block, the fields must be qualified with the character string specified.

If more than one detail control block is defined for different record types, the fields of all record types are available in all detail control blocks. A record of a record type becomes no longer available when the current record is processed and the next record has the same record type.

A detail line can contain the following outputs:

– literals and texts inserted with SOURCE
– the system variables &PAGES and &LINES
– start parameters
– report set values,
– net data of the current record and other record types
– arithmetic expressions
– local variables.

```
DETAIL [ RECORD TYPE literal [ IN variable ] ] [ line-pattern ] { statement; } ...
```

RECORD TYPE literal

Assigns the record type *literal*. The statements in this control block are then only executed for this record type.

IN variable

Specifies the name of an identifier field for the record type *literal*. The default identifier field is the first field of a data record.

line-pattern

Name of the line background pattern which must be defined with the GLOBAL LINE BASE statement.
The corresponding line pattern is assigned to the detail lines as a background pattern.

statement

Designates one of the DRIVE statements BREAK, CALL, CYCLE, IF, SET or one of the report statements PRINT or SOURCE. You use these statements to describe the detail lines. The statements are subject to the restrictions described elsewhere (see section “Permitted DRIVE statements” on page 213 and section “Restrictions applicable to report parameters” on page 214).

Example

The Courier 12 font is to be used for the detail lines in a report. The variables *&data.name* and *&data.age* are to be output in this font type at the tabulator positions 20 and 30. After a line feed of 2 lines, `Rent:` is to be output at the tab position 25 and finally, the variable *&data.rent* in the’Z9 output format at position 34.

```
...
DETAIL
  PRINT SET (FONT 'COURIER', CHARACTER DENSITY 12),
    TAB 20, &data.name,
    TAB 30, &data.age,
    NL 2,

    TAB 25, 'Rent: ',
    TAB 34, &data.rent MASK 'Z9', '% ';
...
```

END REPORT
End report definition

END REPORT terminates a report definition.

```
END REPORT
```

FILL REPORT
Fill report with data

FILL REPORT transfers a data record to a report buffer. Here the report generator prepares the data in accordance with the report definition for the output device which has been selected in OPEN REPORT.

As many FILL REPORT statements as are required for the data transfer can be placed between an OPEN REPORT and its associated CLOSE REPORT statement.

A record type specified in DECLARE REPORT can also be transferred with the FILL REPORT statement.

A FILL REPORT statement is only allowed in the body of a program and only in program mode.

```
FILL REPORT report-name [ variable ] [ RECORD TYPE char-literal ]
                        USING { expression |  NULL }, ...
```

---

report-name — Specifies the name of the report definition used for generating the report. The name must be declared with a DECLARE REPORT statement (see the DECLARE REPORT statement).

variable — Specifies a variable or variable component which was specified in the OPEN REPORT statement for identification of the report buffer (see the OPEN REPORT statement).

RECORD TYPE char-literal

Transfers the data of the record type *char-literal* to the report buffer. The record type *char-literal* and its structure were defined in the DECLARE REPORT statement (see the DECLARE REPORT statement).

DRIVE/WINDOWS checks whether the expressions in the USING clause are convertible to the data types which were specified for the parameters in the DECLARE REPORT statement.

USING — Transfers the parameters which were defined in the USING clause of the DECLARE REPORT statement.

expression — Specifies the parameters.

NULL — Specifies the null value as parameter.

Example

The report definition "stat" is to be used to create a report containing the data "drive_data". The data are of the record type "record" which has been defined in the DECLARE REPORT statement.

```
...
DECLARE REPORT stat RECORD TYPE 'record' USING &data LIKE &drive_data;
...
END REPORT;
...
FILL REPORT stat RECORD TYPE 'record' USING &drive_data;
...
```

GLOBAL LAYOUT
Set global defaults for a report

GLOBAL LAYOUT is used for individual design of a report and is only permitted within a report definition.

You use the GLOBAL LAYOUT statement to declare global defaults which are valid for the entire report. The following settings can be defined by default:

– defining the sorting criteria for the transfer parameters
– determining the width of the page margins
– determining the size of the area for the page header and trailer
– selecting layout attributes.

For the specification of the page margins and the width of the page header or page trailer areas, the defaults defined in the profile of the selected output device are assumed (see the section on "The report generator" in the "DRIVE Programming Language" [2] manual).

The default display attributes which you declare with the GLOBAL LAYOUT statement can be changed or reset at any time using the PRINT statement (see PRINT statement).

```
GLOBAL LAYOUT
       { ORDER BY { {variable [ ASCENDING | DESCENDING ] }, ... } [ EXTERNAL ] |
         { TOP MARGIN n | BOTTOM MARGIN n | LEFT MARGIN n | RIGHT MARGIN n } |
         { HEADER LINES n | TRAILER LINES n | MINIMUM LINES n } |
         format-clause } ...
```

ORDER BY

Defines the sorting order of the field contents of the parameter *variable*.

If groups are defined, the field contents must be sorted in compliance with the control break fields according to their hierarchy.

If a report definition contains no ORDER clause, implicit sorting is performed if group control blocks are defined (see GROUP statement) and a single field has been defined as the group break field. In this case, sorting is performed in ascending order on the group break fields.

variable	Specifies a component of the parameter *var-name-2*, which was defined in the USING clause of the DECLARE REPORT statement. The component *variable* must be a simple type and must be uniquely qualified. The records to be processed are sorted by the field contents of these components.
ASCENDING	Default. Sorting is performed in ascending order.
DESCENDING	Sorting is performed in descending order.
EXTERNAL	Specifies that the data which is passed to the report generator has already been sorted. If you specify EXTERNAL, the report generator does not check whether or not the data is present in sorted form.
TOP MARGIN n	Specifies the number *n* of lines which the report generator is to reserve for the top margin of an output page. *n* must not be greater than 32767.
BOTTOM MARGIN n	Specifies the number *n* of lines which the report generator is to reserve for the bottom margin of an output page. *n* must not be greater than 32767.
LEFT MARGIN n	Specifies the number *n* of columns which the report generator is to reserve for the left margin of an output page. *n* must not be greater than 32767.
RIGHT MARGIN n	Specifies the number *n* of columns which the report generator is to reserve for the right margin of an output page. *n* must not be greater than 32767.
HEADER LINES n	Specifies the number *n* of lines which the report generator is to reserve for the header of an output page. *n* must not be greater than 32767.
TRAILER LINES n	Specifies the number *n* of lines which the report generator is to reserve for the trailer of an output page. *n* must not be greater than 32767.
MINIMUM LINES n	Specifies the minimum number *n* of lines which must be available for a report page. *n* must not be greater than 32767. This input is useful if the size of an output page is not yet known during the report definition. This prevents pages from being output, for example, with just page headers and trailers.

format-clause

The description of the layout attributes, which can be specified in the *format-clause* can be found in the PRINT statement.

The layout attributes SUBSCRIPT and SUPERSCRIPT **must not** be used in the GLOBAL LAYOUT statement.

Example

A report on the consumption behavior of people is to be produced. The structured variable &person is declared in the DRIVE program for the data required for the report.

The report definition is given the name "behave". Here the DRIVE variable &person is passed to the report definition via the parameter &data.

The local variable &i is declared and the sorting order defined in the report definition. Sorting is to be in ascending order by the components &data.sex, &data.social and &data.name. Moreover, the margins and the area for header and trailer lines are determined. The layout attributes selected afterwards, such as the font, character density and character type, apply to the entire report. They can, however, be changed with the PRINT statement for individual output fields or for the entire report.

```
...
/* Declare DRIVE variables to receive report data                */

DECLARE VARIABLE  1 &person,
                    2 name       VARCHAR(30),
                    2 age        SMALLINT,
                    2 sex        CHAR,
                    2 social     CHAR,
                    2 rent       INTEGER,
                    2 car        INTEGER,
                    2 consum     INTEGER;

/* Declare report */

DECLARE REPORT behave USING &data LIKE &person;

   DECLARE VARIABLE &i INTEGER;              /* Declare report variable */

   GLOBAL LAYOUT                             /* Set layout defaults   */
      ORDER BY &data.sex ASCENDING,          /* Sort criteria         */
               &data.social ASCENDING,
               &data.name ASCENDING
```

```
      TOP MARGIN 2                          /* Margins              */
      BOTTOM MARGIN 3
      HEADER LINES 2
      TRAILER LINES 5

      FONT 'CENTURY CONDENSED'              /* Display attributes   */
      CHARACTER DENSITY 10
      ITALIC 1;
      ...
END REPORT;
```

GLOBAL LINE BASE
Define line background

You use the GLOBAL LINE BASE statement to define a named background pattern (base) for a printed line. The background pattern may consist of tabs and positioned text elements. You can either enter text directly or read it from a message file via MSGSTRING.

Any line background pattern can be assigned to a variety of control blocks. The text is output as a background pattern which is overwritten by current input.
In the PRINT statement you can process the tabs defined in GLOBAL LINE BASE one by one from left to right and use these tabs to determine positions within the line.

The GLOBAL LINE BASE statement is only permitted within a report definition and must precede the first REPORT DIRECTIVE statement. If the report definition contains the GLOBAL LAYOUT statement, then GLOBAL LINE BASE must occur between the GLOBAL LAYOUT statement and the first REPORT DIRECTIVE statement.

```
GLOBAL LINE BASE line-pattern

       { tab_position |
         PATTERN position { char-literal |
                            MSGSTRING ( value-expression1 [ [, value-expression2 ],
name ] ) }
                          [ format-clause ] ... }, ...
```

---

line-pattern	Name of the line background pattern (max 54 characters).
tab_position	Defines a tab with the tab position *tab_position*. *tab_position* must be an unsigned number of type INTEGER. The position is calculated in the units (cm, inch, 1/300 inch) which have been defined in the GLOBAL LAYOUT statement. *tab_position* must lie within a line of print.
PATTERN	Text which is input as a literal or read from a message file is used as the background pattern.
position	Specifies the text position as an unsigned number of type INTEGER. The position is calculated in the units (cm, inch, 1/300 inch) which have been defined in the GLOBAL LAYOUT statement. *position* must lie within a line of print.

char-literal
: Specifies a text which is to be displayed as a background pattern (*char-literal*, see metavariable *literal*).

MSGSTRING (value-expression1 [[, value-expression2], name])
: The text which is to be used as the background pattern is read from a message file (*MSGSTRING,* see metavariable *char-prim*).

format-clause
: Specifies the display attributes of the text of the background pattern. Different text components may have different display attributes. You may specify the following display attributes in *format-clause*: FONT, NATIONAL SET, SIZE, BOLD, INVERSE, ITALIC, UNDERLINE, COLOUR FOREGROUND, COLOUR BACKGROUND.
The meaning of these individual display attributes is explained under *format-clause* in the description of the PRINT statement.

GLOBAL PAGE BASE
Define page background pattern

You use the GLOBAL PAGE BASE statement to define a named background pattern (base) for a printed page. Alongside positioned texts, lines and rectangles, the background pattern may also contain rotated texts. Certain printers also allow you to use page background patterns which are not created using DRIVE/report resources (see SOURCE statement).

You can use page background patterns to:

– output net data in a preprinted form,
– create preprinted forms using DRIVE/report resources,
– cover printed pages with a predefined pattern.

You use the OVERLAY PAGE BASE statement to activate and deactivate a page background pattern. If the page background pattern is activated, it is overlaid by the net data output.

The GLOBAL PAGE BASE statement is only permitted within a report definition and must precede the first REPORT DIRECTIVE statement. If the report definition contains the GLOBAL LAYOUT statement, then GLOBAL PAGE BASE must occur between the GLOBAL LAYOUT statement and the first REPORT DIRECTIVE statement.

```
GLOBAL PAGE BASE page_pattern_name { statement; } ...
```

---

page_pattern_name — Name of the page background pattern (max 54 characters).

statement — Designates one of the DRIVE statements CYCLE, IF, SET or one of the report statements PAGE PRINT or SOURCE. You use these statements to describe the page background pattern. The statements are subject to the restrictions described elsewhere (see section “Permitted DRIVE statements” on page 213 and section “Restrictions applicable to report parameters” on page 214).

If you specify SOURCE, then it can only be as the first and only *statement*.

GROUP
Define group control block

GROUP is used to describe a group header or group trailer for a group. A control break field identifying the group must be specified. A group header can include, for example, a title for identification of a group. Group-specific intermediate results can be contained in a group trailer for instance.

GROUP is only allowed within a report definition and may be specified more than once. The order in which the declarations are made determines the hierarchy according to which group processing is performed

```
group header a
   group header b
      group header c
         detail lines
      group trailer c
   group trailer b
group trailer a
```

If more than one group is defined and if there is a control break in the next data record to be processed, the control blocks are processed in the following order:

– all group trailers from the lowest to the control hierarchy in which the control break occurs
– all group headers from the current to the lowest control hierarchy
– all statements for the detail area, with the data of the next data record.

The data to be processed must be either transferred to the report generator already sorted or sorted by including ORDER in the GLOBAL LAYOUT statement (see the GLOBAL LAYOUT statement). In the example shown above, the sorting of a, b and c was assumed. If the data is not sorted and if the GLOBAL LAYOUT statement does not include any ORDER clause, sorting is implicitly performed if group control blocks are defined. In this case, sorting is effected by control break fields in ascending order.

Group headers and trailers may have more than one line and include the following outputs:

– literals and texts inserted with SOURCE
– the system variables &PAGES and &LINES
– start parameters
– values derived from report set functions
– arithmetic expressions
– net data of the current data record
– local variables.

```
GROUP { HEADER | TRAILER } { variable }, ... [ GROUPNUM n ] [ line-pattern ]
      { statement; } ...
```

HEADER	Defines the group header for the group *variable*. The group header is described with the following statements.
TRAILER	Defines the group trailer for the group *variable*. The group trailer is described with the following statements.
variable	Specifies a control break field used for identification of a group. A control break is initiated if the contents of the control break field are changed. *variable* must be a simple component in the set of data to be processed. More than one field (field combinations) can also be specified. A control break then occurs if the contents of one of these fields change.
GROUPNUM n	Specifies a group number *n* for the group *variable*. This group number can be entered for report set functions if a calculation is intended to refer to this group only (see section “Report set functions in expressions” on page 215). *n* must not be greater than 32767.
line-pattern	Name of a line background pattern which must have been defined using GLOBAL LINE BASE The corresponding line pattern is assigned to the group header or trailer as a background pattern.
statement	Designates one of the DRIVE statements BREAK, CALL, CYCLE, IF, SET or one of the report statements PRINT or SOURCE. You use these statements to describe the group trailer or header. The statements are subject to the restrictions described elsewhere (see section “Permitted DRIVE statements” on page 213 and section “Restrictions applicable to report parameters” on page 214).

Example

You want to use the header “Men: “ or “Women: “ for the group &data.sex

```
...
GROUP HEADER &data.sex

   PRINT NEWLINE 1, SET (NATIONAL SET 'UK-ENGLISH', BOLD 1,
                         FONT 'COURIER', CHAR DENSITY 12),
         NEED LINES 5;

   IF (&data.sex = 'm') THEN
      PRINT TAB 10, 'Men: ',
            NEWLINE 1;
   ELSE
      PRINT TAB 10, 'Women: ',
            NEWLINE 1;
   END IF;
...
```

OPEN REPORT
Start report execution

OPEN REPORT is used to start the execution of a report. The report definition is specified and a report buffer which accepts the report data is provided and initialized. At the same time, the start parameters defined in DECLARE REPORT can be transferred.

The output device for a report is also specified with the statement OPEN REPORT; the device can be a printer or a file.

The report buffer and the report definition are closed again and the execution is ended with an associated CLOSE REPORT statement. After closing the report definition and the buffer the report is generated at the selected output device.

Under UTM, no terminal entries or outputs are allowed between the execution of an OPEN REPORT statement and the associated CLOSE REPORT statement.

More than one OPEN REPORT statement may be specified.

This statement is only permitted in the body of a program.

```
OPEN REPORT report-name [ variable ] [ USING { expression | NULL }, ... ]

     RESULT { LIST [ device ] [ DEVICETABLE device-table [ spoolparameter ] ] |
              FILE file [ DEVICETABLE device-table ] }
```

---

report-name — Specifies the name of the report definition to be used. The name must be declared with a DECLARE REPORT statement (see the DECLARE REPORT statement).

variable — Designates a variable or component of type CHAR(8) which is used to identify the report buffer. You must specify *variable* if you want to generate multiple reports simultaneously, all of which use the report definition *report-name*. Different *variables* designate different reports. DRIVE/WINDOWS stores an internal report ID in *variable*.

If *variable* is specified, it must be explicitly declared in the DRIVE program and must not be changed with a SET statement or any other statement at the time of execution, between OPEN and CLOSE.

If *variable* is not specified, DRIVE uses a variable with the name *report-name* which is automatically created for each report when DECLARE REPORT is executed.

USING	Transfers values for the start parameters which have been defined in the DECLARE REPORT statement using the FOR START USING clause.
expression	Transfers the value of *expression* (see metavariable *expression*).
NULL	Transfers the null value.
RESULT LIST	Specifies that the report is to be output at a printer. If you want to output the report at a Remote Spoolout (RSO) printer, you must specify *device* and DEVICETABLE *device-table*.
device	Symbolic name for a printer (max. 256 characters). You may specify this name as a literal or as the contents of a DRIVE variable of type CHARACTER or VARCHAR. If output is performed at a system printer, the *device* specification is not evaluated. If output is performed at an RSO printer, *device* must be specified.
RESULT FILE	Specifies that the report is to be output to a file. Output to a file is performed in a device-independent format as specified in *device-table*.
file	Name of the file to which the report is to be output. The name must comply with the BS2000 conventions for filenames. You may specify this name as a literal or as the contents of a DRIVE variable of type CHARACTER or VARCHAR.
DEVICETABLE	Assigns a device table. You must specify DEVICETABLE if output is performed at an RSO printer.
device-table	This is the logical name of the output file. You may specify this name as a literal or as the contents of a DRIVE variable of type CHARACTER or VARCHAR (max. 256 characters). The device profile is stored in the file PROFILE.*device-table*. This profile contains the description of an output page and the assignment of device-independent to device-specific control codes (see the DRIVE Programming Language manual [2], chapter The report generator). If output is sent to a system printer then you must specify the *device-table* value “ND” for output at an NP laser printer or, alternatively, the value “HP" for output at an HP laser printer. If you do not specify *device-table* then "ND" is used.

spoolparameter

Specifies options for print management.
You may specify *spoolparameter* as a literal or as the contents of a DRIVE variable of type CHARACTER or VARCHAR (max. 256 characters).

DRIVE/WINDOWS automatically enters a PRINT-DOCUMENT command which is structured as follows:

```
PRINT-DOCUMENT FROM-FILE=temp-file,-
   TO-PRINTER=*PARAMETERS(PRINTER-TYPE=*HP-PRINTER),-
   DELETE-AFTER-PRINT=*YES
```

or

```
PRINT-DOCUMENT FROM-FILE=temp-file,-
   DELETE-AFTER-PRINT=*YES
```

The user can also determine further options for print output in the PRINT-DOCUMENT command. These options must conform to the syntax of the BS2000 command PRINT-DOCUMENT in SDF format (see BS2000 Commands [35]).

The *spoolparameter* string is passed to the print management function unchecked.

If you want to use multiple character sets when outputting a report at the system printer, you must specify these character sets in the PRINT DOCUMENT command (see below).

If you do not specify any print management options, then the `SPOOLDOPT` setting in the user profile is evaluated. In such cases, the user-specific user profile is evaluated before the system-specific user profile.

If there is no user profile specification then the operand `LAYOUT-CONTROL=PARAMETERS(CONTROL-CHARACTERS=PHYSICAL)` is used for the BS-2000 PRINT-DOCUMENT command (see BS2000 Commands [35]).

This default value corresponds to the minimum requirement.

No other user profile settings which relate to the spooler have any effect.

Outputting reports with multiple character sets at system printers:

The following criteria must be met if you want to use multiple character sets when outputting a report:

– The same number of character sets must be defined in the device profile (PROFILE.ND or PROFILE.HP). In the case of ND laser printers; you can specify a maximum of 4 and for HP laser printers a maximum of 64 character sets.

– In the report definition, the character sets must be used with the names which are defined in the device profile. In the case of ND laser printers, the character sets have the default names SECTION1 to SECTION4,and in the case of HP laser printers SECTION1 to SECTION64. For example, you may enter:
`PRINT expression1 ATTRIBUTE(FONT SECTION1);` and
`PRINT SET(FONT SECTION2);`

– You must specify the character sets which are to be loaded into the printer in the PRINT-DOCUMENT command (see BS2000 Commands [35]). To do this, specify CHARACTER-SETS for the operand LAYOUT-CONTROL:
`LAYOUT-CONTROL=PARAMETERS(CONTROL-CHARACTERS=PHYSICAL,`
`  CHARACTER-SETS=(`*`font1,font2, ...`*`) )`

  The first character set named in CHARACTER-SETS is assigned to the character set `SECTION1`, the second character set named is assigned to `SECTION2` etc. (valid for positional parameters).

  If you use the SECTION1 and SECTION2 character sets in the report then you might enter
`LAYOUT-CONTROL=PARAMETERS(CONTROL-CHARACTERS=PHYSICAL,`
`  CHARACTER-SETS=(`*`105,203`*`) )`
  The character set "SECTION1" designated in the FONT clause in the report is output using character set 105 while "SECTION2" is output using character set 203.

Example

The execution of multiple reports with the report definition stat is started. The variables &buf1 and &buf2 are used to identify the report buffer which is to be opened. The only start parameter to be passed is the variable &start. The output device is to be an HP laser printer.

```
...
OPEN REPORT stat &buf1 USING &start RESULT LIST DEVICETABLE 'HP';
OPEN REPORT stat &buf2 USING &start RESULT LIST DEVICETABLE 'HP';
...
```

OVERLAY PAGE BASE
Activate page background pattern

You use the OVERLAY PAGE BASE statement to activate a page background pattern which you have previously defined using GLOBAL PAGE BASE.

You may only use the OVERLAY PAGE BASE statement within a report definition.

```
OVERLAY PAGE BASE page-pattern-name { ON | OFF }
```

---

page-pattern-name — Name of a page background pattern which has been defined using GLOBAL PAGE BASE.

The OVERLAY PAGE BASE statement applies to the current printed page if it is specified as the first statement for the output page (print position: first line, first column). For example, it applies to the current page if it is specified as the first statement of a PAGE HEADER control block.
Otherwise this statement does not take effect until the next printed page.

PAGE
Define page control block

PAGE is used to describe the header or trailer for all report output pages. A page header can contain, for example, values derived from report set functions on the previous page as carry. In a page trailer, an intermediate result or a page number can be output for example.

The PAGE statement is only allowed within a report definition and may only be specified once for defining a page header and page trailer in each case. If a page header or trailer is defined, a header or trailer area must also defined in the statement GLOBAL LAYOUT (see the GLOBAL LAYOUT statement).

The page header is output at the start and the page trailer at the end of every page. An exception to this exists if a list header or list trailer requires a whole page. In such cases, neither a page header nor trailer appears on the first or last page respectively of a report.

A page header or page trailer is output if the report generator detects that the next free line is identical to the following:

– the first line of the trailer area specified in the GLOBAL LAYOUT statement or
– the first line which exceeds the page height.

A page header or trailer can include the following outputs:

– literals and texts inserted with SOURCE
– the system variables &PAGES and &LINES
– start parameters
– values derived from report set functions
– arithmetic expressions
– net data of the current data record
– local variables.

```
PAGE { HEADER |  TRAILER } [ line-pattern ] { statement; } ...
```

HEADER	Defines the page header, which is described by the following statements.
TRAILER	Defines the page trailer, which is described by the following statements.

line-pattern	Name of a line background pattern which must have been defined using GLOBAL LINE BASE. The corresponding line pattern is assigned to the group header or trailer as a background pattern.
statement	Designates one of the DRIVE statements BREAK, CALL, CYCLE, IF, SET or one of the report statements PRINT or SOURCE. You use these statements to describe the page trailer or header. The statements are subject to the restrictions described elsewhere (see section “Permitted DRIVE statements” on page 213 and section “Restrictions applicable to report parameters” on page 214).

PAGE PRINT
Describe page background pattern

You use the PAGE PRINT statement to define texts, lines or rectangles for a page background pattern.

The PAGE PRINT statement is only permitted in the definition of a page background pattern in the GLOBAL PAGE BASE statement.

```
PAGE PRINT { expression x y [ CM | INCH | UNITS ]  [ mask ]  [ ANGLE n ]
                          [ format-clause ] ... |

             SET ( { format-clause }, ... ) |

             RESET ( { FONT | NATIONAL SET | SIZE |
                       CHARACTER DENSITY | CHARACTER DISTANCE |
                       EXPANSION HORIZONTAL | EXPANSION VERTICAL |
                       INVERSE | ITALIC | BOLD | UNDERLINE |
                       COLOUR FOREGROUND | COLOUR BACKGROUND }, ... ) |

             LINE x1 y1 x2 y2 [ CM | INCH | UNITS ]
                              [ LTYPE char-literal ]
                              [ LWIDTH x [ CM | INCH | UNITS ] ] |

             BOX x1 y1 x2 y2 [ CM | INCH | UNITS ]
                             [ BTYPE { char-literal | EMPTY } ]
                             BWIDTH x
                             [ LINE [ LTYPE char-literal ]
                                    [ LWIDTH x [ CM | INCH | UNITS ] ]  ]   |

             IMAGE LENGTH n [ WIDTH n ] [ RESOLUTION n ]
                   { [ COMPRESS char-literal ] DATA hex-literal } ... }, ...
```

expression x y [CM | INCH | UNITS]

Specifies the background pattern *expression* with the output position *x y*. The coordinates of this position are relative to the origin in the top left-hand corner of the maximum print area (the axes run along the top and left-hand edges of this area). The output position is calculated in the specified unit of measurement (centimeter, inch or 1/300 inch (UNITS)). If you do not specify a unit of measurement the calculation is performed in UNITS.
x and *y* can be specified with decimal places. They must be positive and may not be greater than 32767.
If *expression* designates variables, then the specified variables may be simple variables only.

mask

Designates a display option (mask control character) for the output of *expression* (see metavariable *mask*). The restrictions described elsewhere apply to the *mask* specification (see section “Restrictions applicable to report parameters” on page 214).

ANGLE n

Specifies an angle through which *expression* is to be rotated. The angle is entered as a positive whole number. You may enter values between 0° and 360°.
HP LaserJet printers respect the values 0, 90, 180 and 270. All other values are ignored.

format-clause

Specifies display attributes for the current *expression*.The following display attributes are permitted:

FONT, NATIONAL SET, SIZE, CHARACTER DENSITY, CHARACTER DISTANCE, EXPANSION HORIZONTAL, EXPANSION VERTICAL, BOLD, INVERSE, ITALIC, UNDERLINE, COLOUR FOREGROUND, COLOUR BACKGROUND.

For a description of the individual display attributes, refer to *format-clause* in the PRINT statement.

SET ({ format-clause }, ...)

Sets or modifies default display attributes which have been specified in the GLOBAL LAYOUT statement.

You can specify the following display attributes in SET:

FONT, NATIONAL SET, SIZE, CHARACTER DENSITY, CHARACTER DISTANCE, EXPANSION HORIZONTAL, EXPANSION VERTICAL, BOLD, INVERSE, ITALIC, UNDERLINE, COLOUR FOREGROUND, COLOUR BACKGROUND.

Refer to *format-clause* for a description of the individual display attributes. The display attributes described in *format-clause* are valid from the current position as far as the position at which they are reset.

RESET (...)

Resets the display attributes which are specified in SET to the default values assigned to them in the GLOBAL LAYOUT statement. If no global defaults have been specified, RESET uses the device-specific defaults.

Refer to *format-clause* for an explanation of the keywords.

LINE x1 y1 x2 y2 [CM | INCH | UNITS]

Draws a line as a vectored graphic in a background pattern. The line is defined by the start coordinates *x1 y1* and the end coordinates *x2 y2*. The coordinates of this position are relative to the origin in the top left-hand corner of the maximum print area (the axes run along the top and left-hand edges of this area).

x and *y* can be specified with decimal places. They must be positive and may not be greater than 32767.

The positions are calculated in the specified unit of measurement (centimeter, inch or 1/300 inch (UNITS)). If you do not specify a unit of measurement the calculation is performed in UNITS.

LTYPE char-literal

Specifies whether a continuous, interrupted or dotted line is drawn. The *char-literal* specifications which are permitted for any given printer can be found in the device profile PROFILE.*device-table*.

If you do not specify LTYPE the default from the device profile is used.

LWIDTH x [CM | INCH | UNITS]

Specifies the line width.

The line width is calculated in the specified unit of measurement (centimeter, inch or 1/300 inch (UNITS)). If you do not specify a unit of measurement the calculation is performed in UNITS.

x can be specified with decimal places. It must be positive and may not be greater than 32767.

If you do not specify LWIDTH the default from the device profile PROFILE.*device-table* is used.

BOX x1 y1 x2 y2 [CM | INCH | UNITS]

A rectangle is drawn as the background pattern. The rectangle is defined by a perimeter and a size (BWIDTH). You can specify a gray value as fill pattern (BTYPE). If you want to draw the rectangle with a border you must use LINE to assign a line width to the rectangle (see LWIDTH).

The perimeter of the rectangle is defined by the start coordinates *x1 y1* and the end coordinates *x2 y2*. The coordinates are relative to an origin in the top left-hand corner of the maximum print area (the axes run along the top and left-hand edges of this area).
x and *y* can be specified with decimal places. They must be positive and may not be greater than 32767.

The positions are calculated in the specified unit of measurement (centimeter, inch or 1/300 inch (UNITS)). If you do not specify a unit of measurement the calculation is performed in UNITS.

BTYPE { charliteral | EMPTY }

Specifies the gray value of the fill pattern.

The *char-literal* specifications which are permitted for any given printer can be found in the device profile PROFILE.*device-table*.

Specify *EMPTY* if you do not want to assign any fill pattern to the rectangle.

If you do not specify BTYPE the default from the device profile is used.

BWIDTH x

Specifies the size of the rectangle (in clockwise direction).
This size is calculated in the units of measurement specified in BOX.

x can be specified with decimal places. It must be positive and may not be greater than 32767.
No rectangle is output unless *x* > 0.

BOX ... LINE

You use the LINE specification to assign a border to a rectangle which is drawn parallel to the edge of the sheet. You can specify the line type (see LTYPE) and the line width (see LWIDTH).

No border line is drawn unless you also specify a line width.

If the rectangle is not drawn parallel to the edge of the sheet, the LINE specification is ignored.

IMAGE

Imports raster graphics into a background patter.

LENGTH n

Specifies the number of lines occupied by the raster image.

n must be an unsigned integer and may not be greater than 32767.

WIDTH n

Specifies the width of the raster image in pixels.

n must be an unsigned integer and may not be greater than 32767.

If you do not specify WIDTH then the width of the printed page is used for this value.

RESOLUTION n

Specifies the resolution of the raster image in dots-per-inch.

n must be an unsigned integer and may not be greater than 32767. You may only enter the values 75, 100, 150 and 300 for HP LaserJet printers.

If you do not specify RESOLUTION the default from the device profile PROFILE.*device-table* is used.

COMPRESS char-literal

If the raster image is present in compressed form, this specification defines the mode of compression. Possible values for *char-literal* are:
UNENCODED: not compressed (default)
RLE: Run Length Encoding
TIFF: Tagged Imaged File Format rev. 4.0
DELTA ROW: Delta Row Compression

The individual lines of the raster image may be encoded in different ways.

DATA hex-literal

Specifies the data which describe the raster image as a hexadecimal string (for *hex-literal* see metavariable *literal*).

If you do not specify the compression mode of a raster image, the last value specified in COMPRESS is used for the current line.

PRINT
Define report output

PRINT specifies the following in all control blocks,

– which fields are to be output at which position
– how many lines must exist in order to continue the output on the current page
– how many empty lines are to be inserted as of the current line
– whether output is to be continued on a new page.

You can use the PRINT statement to modify the display attibutes which are preset in the GLOBAL LAYOUT statement or to reset specifications to these preset values.

If the position is not specified in a PRINT statement, output is continued from the current position. If an output position is specified by the number of columns, the defaults defined in the profile of the selected output device are assumed (see the section on "The report generator" in the "DRIVE Programming Language" [2] manual).

PRINT statements are only allowed within a report definition.

```
PRINT { NEED LINES n  |
        NEED SPACE n  { CM | INCH | UNITS } |
        NEWLINE n |
        NEWPAGE   |
        TABULATOR [ [ + ] n  ] |
        POSITION x { CM | INCH | UNITS } |
        expression [ mask | CLIPPED ]  [ ATTRIBUT ( { format-clause }, ... ) ]
                   [ MANDATORY | DISTINCT ]
                   [ CENTER [ x y [ CM | INCH | UNITS ] ] |
                     RIGHT x  [ CM | INCH | UNITS ]  ]     |

        SET ( { format-clause }, ... ) |

        RESET ( { FONT | NATIONAL SET | SIZE |
                  CHARACTER DENSITY | CHARACTER DISTANCE |
                  LINE DISTANCE | EXPANSION HORIZONTAL |
                  EXPANSION VERTICAL | SIGN | BOLD |
                  NORMALINTENSITY | INVERSE | ITALIC |
                  PROPORTIONAL | SUBSCRIPT | SUPERSCRIPT |
                  UNDERLINE | COLOUR FOREGROUND | COLOUR BACKGROUND |
                  PAPER SOURCE }, ... )

        PAGE POSITION x y [ CM | INCH | UNITS ] }, ...
```

NEED LINES n
: Number of lines *n* which must still be available for a detail area in order to continue output. If *n* lines are available then output continues on the current page. If fewer than *n* lines are available then a new page is created. *n* may not be greater than 32767.

  This entry prevents, for example, a title from being output on the last line of a page and the associated list elements on a new page.

  No page break may occur within the page header or page trailer.

NEED SPACE n { CM | INCH | UNITS }
: NEED SPACE acts in the same way as NEED LINES except that the size of the required area is not specified in lines but in CM, INCH or 1/300 inch (UNITS) (this specification can therefore lead to different results on different printers).

NEWLINE n
: Causes a line feed of *n* lines. If *n* has the value 0 then a carriage return (CR) is performed. *n* may not be greater than 32767.

NEWPAGE
Causes a page feed and is not permitted in the definition of page headers or trailers.

TABULATOR
Uses tabs which have been defined as a line background pattern in GLOBAL LINE BASE.
Output is continued at the next tab position defined in the line background pattern, working from left to right.

TABULATOR [+] n
Specifies the character-dependent position at which output is to be continued. You can specify this position in two ways:

– absolute: TABULATOR *n*
– relative: TABULATOR + *n*

For absolute positioning, the current line from column *n* onward is output.

For relative positioning, the number *n* of columns counted from the end of the previously output field (reference field) is specified so as to determine the output position. *n* must not be greater than 32767.

If the reference field for the relative positioning is a character-string field, two separate cases are distinguished:

– The output length of the reference field is independent of the final blank. In this case the position of the current output is calculated relative to the final position of the reference field, even if this contains a blank.
– The output length of the reference field is dependent on the final blank and extends only as far as the final character which is not a blank. In this case the position of the current output is calculated relative to the last output position of the reference field (see CLIPPED).

If you specify the output position using TABULATOR, then the column widths are dependent on the units used for the character set as specified in the GLOBAL LAYOUT statement (see GLOBAL LAYOUT statement).

POSITION x { CM | INCH | UNITS }
Specifies the output position *x* in the selected unit of measurement (centimeter, inch or 1/300 inch (UNITS)) measured from the left-hand edge of the page. *x* can be specified with decimal places and may not be greater than 32767.

It is a good idea to use POSITION to specify the output position if different character sizes and character weights are used within an output line.

expression
: Specifies the parameter to be output. A representation option, an output position, certain attributes and the output type for control break fields can be selected for this parameter.

  If *expression* represents a variable, the variable must be a simple variable.

mask
: Designates a display option (mask control character) for the output of *expression* (see metavariable *mask*). The restrictions described elsewhere apply to the *mask* specification.

CLIPPED
: Suppresses the output of terminating blanks for fields of type CHARACTER and VARCHAR.

  This specification is only valid if the next PRINT statement does not specify POSITION but relative positioning (see TABULATOR).

ATTRIBUTE format-clause
: Specifies the layout attributes for *expression* according to *format-clause* (see the section on "format-clause" below). Since the specified attributes only apply to *expression*, the entries LINE DISTANCE and PROPORTIONAL are **not** permitted here.

MANDATORY
: If *expression* designates a group change field, then specifying MANDATORY forces the contents of this field to be output each time. The default is for the contents of a group break field to be output only after the first group break and whenever a new page is started.

DISTINCT
: For output, only the fields of *expression* which differ from the contents of the field in the previous record are used.

CENTER [x y [CM | INCH | UNITS]]
: Causes the field to be centered relative to the left-hand and right-hand margins. You specify the margins in the values *x* (left-hand margin) and *y* (right-hand margin). The position is calculated in the specified unit of measurement (centimeter, inch or 1/300 inch (UNITS)) and represents the distance from the left-hand and right-hand page edges. *x* can be specified with decimal places and may not be greater than 32767.
  If you do not specify a unit of measurement, the position is calculated in columns.
  The exact position is calculated for non-proportional fonts.

  If you do not specify the margins, the field is centered relative to the left-hand and right-hand page edges.

  If you specify CENTER, the PRINT command may not contain any other positioning specifications.

RIGHT [x [CM | INCH | UNITS]]

Causes the field to be output right-aligned at the right-hand margin. You specify the margin in the value *x*.The position is calculated in the specified unit of measurement (centimeter, inch or 1/300 inch (UNITS)) and represents the distance from the right-hand page edge. *x* can be specified with decimal places and may not be greater than 32767.
If you do not specify a unit of measurement, the position is calculated in columns.
The exact position is calculated for non-proportional fonts.

If you do not specify the margin, the field is aligned with the right-hand page edge.

If you specify RIGHT, the PRINT command may not contain any other positioning specifications.

SET format-clause

Sets or modifies default display attributes which have been specified in the GLOBAL LAYOUT statement. Refer to *format-clause* for a description of the individual display attributes. You may not specify ROTATION. The display attributes specified in *format-clause* are valid from the current output position

– to the end of the control block in which they are set or
– to the position at which they are reset.

RESET

Resets the display attributes which are specified in SET to the default values assigned to them in the GLOBAL LAYOUT statement. If no global defaults have been specified, RESET uses the device-specific defaults.

Refer to *format-clause* for an explanation of the keywords.

PAGE POSITION x y { CM | INCH | UNITS]

Causes absolute positioning at the coordinates *x y*. The coordinates are relative to an origin in the top left-hand corner of the maximum print area (the axes run along the top and left-hand edges of this area). The position is calculated in the specified unit of measurement (centimeter, inch or 1/300 inch (UNITS)). If you do not specify a unit of measurement, the position is calculated in UNITS. *x* and *y* can be specified with decimal places and may not be greater than 32767.
y must be a positive value. If *x* is signed, the coordinates are calculated with reference to the relative output position rather than relative to the origin.

You may only specify PAGE POSITION in a DETAIL control block. In addition, if you specify PAGE POSITION then the report definition may not contain any PAGE DIRECTIVEs or GROUP DIRECTIVEs.

PAGE POSITION is intended for use in conjunction with a page background pattern (see the GLOBAL PAGE BASE statement) for form printing.

format-clause

You can use the specifications in *format-clause* to set display attributes for the entire report in the GLOBAL LAYOUT statement. If no global defaults are defined, the report generator uses the device-specific defaults. The following presettings are specified in the PRINT statement

expression ATTRIBUTE
: Only *expression* is changed.

SET
: Defaults are changed from the current position up to the end of the control block or up to the corresponding RESET.

RESET
: All the defaults or any default individually are/is reset to their/its initial value(s).

When specifying display attributes, you should note that display attributes are not implemented unless they are supported by the device selected with OPEN REPORT. You should also bear in mind that certain attributes are interdependent (see FONT). Such attributes can only be implented if the dependent attributes are also correspondingly modified. Display attributes which are not supported are ignored.

```
{ ROTATION { PORTRAIT | LANDSCAPE } |
  FONT char-literal |
  NATIONAL SET char-literal |
  SIZE x |
  CHARACTER { DENSITY | DISTANCE } x |
  LINE DISTANCE x |
  EXPANSION { HORIZONTAL | VERTICAL } n |
  SIGN |
  BOLD n |
  NORMALINTENSITY n |
  INVERSE n |
  ITALIC n |
  PROPORTIONAL |
  SUBSCRIPT |
  SUPERSCRIPT |
  UNDERLINE n |
  COLOUR { FOREGROUND |  BACKGROUND } char-literal
  PAPER SOURCE  char-literal } ...
```

ROTATION — Specifies the direction of output (portrait or landscape).

PORTRAIT — Output is performed in portrait format.

LANDSCAPE — Output is performed in landscape format.

FONT char-literal — Specifies the font *char-literal* to be used for the printed output of the report. Depending on the printer, possible entries for *char-literal* could be COURIER, ROMAN and HELVETICA. The available fonts are listed in the relevant printer manuals.

The character sets available in ND laser printers are named SECTION1 to SECTION4 and in HP laser printers SECTION1 to SECTION64 (= default). (See OPEN REPORT statement, spoolparameter operand).

If a new font is selected, the following entries must also be changed:

– CHARACTER DENSITY *n*
– PROPORTIONAL spacing (on/off).

For some printers, the type size SIZE *n* must also be newly specified. The appropriate entries can likewise be obtained from the printer manuals.

If you do not specify FONT *char-literal* or if the printer does not support the specified font, the report is output in the printer's default font.

NATIONAL SET char-literal

Specifies the font to be used for output. The font has the name *char-literal*. This entry is used to "correctly" output special national characters, for example, the German umlauts:

char-literal	hexadecimal												
	23	24	40	5B	5C	5D	5E	5F	60	7B	7C	7D	7E
INT003	#	¤	@	[	\	]	^	_	`	{	|	}	¯
INT303	#	$	@	[	\	]	^	_	`	{	|	}	~
BELGIAN	#	$	à	¨	ç	°	^	_	`	é	ù	è	~
DANISH	#	$	@	Æ	Ø	Å	Ü	_	`	æ	ø	å	ü
DUTCH	#	$	@	[	\	]	^	_	`	{	|	}	~
FRENCH	#	$	à	°	ç	§	^	_	`	é	ù	è	¨
GERMAN	#	$	§	Ä	Ö	Ü	^	_	`	ä	ö	ü	ß
ITALIAN	£	$	§	°	ç	é	^	_	ù	à	ò	è	ì
NORWEGIAN	#	$	@	Æ	Ø	Å	^	_	`	æ	ø	å	¨
SPANISH	#	$	@	¡	Ñ	¿	^	_	`	´	ñ	ç	¨
SWEDISH	#	¤	É	Ä	Ö	Å	Ü	_	é	ä	ö	å	ü
SWISS	ç	$	§	à	é	è	^	¨	`	ä	ö	ü	_
UK-ENGLISH	£	$	@	[	\	]	^	_	`	{	|	}	¯

If the specified character set is not available or if you do not specify NATIONAL SET *char-literal*, the report generator uses either the printer's default font or the last character set used when performing output.

SIZE x

Specifies the character size *n* of the font selected with FONT *char-literal*. *n* is specified in units of 1/300 inches (0.0085 cm) and may not be greater than 32767.

If you specify an illegal size *x* then output is performed in the default size for the printer. If you omit SIZE *x* the printer again switches to its default size.

CHARACTER	Specifies the character density or the spacing between the characters to be output for the font selected with FONT *char-literal*. If you do not specify the character density or the spacing between characters or if the specified values are illegal, the default values for the printer are used for output.
DENSITY x	The print density is specified as *x* characters per inch (CPI). *x* may not be greater than 32767. In a non-proportional font, the character density for each size is the same for all characters. If the selected font *char-literal* has proportional spacing, this specification has no effect.
DISTANCE x	The intercharacter spacing *x* is specified in units of 1/300 inch (0.0085 cm). *x* may not be greater than 32767. You can, for example, specify a large intercharacter spacing in order to create letterspaced type.
LINE DISTANCE x	Specifies the distance between two lines to be output which are separated from each other by a line feed (LF). The distance *x* is specified in units of 1/300 inches (0.0085 cm). *x* must not be greater than 32767. If the line distance is not specified or if it is illegal, the printer retains its previous setting.
EXPANSION	Specifies the factor *n* by which all characters are expanded in the horizontal or vertical direction. The factor *n* can have a value from 0 up to and including 6, where the values have the following meanings: 0: no expansion 6: expansion to doubled width or height. The baseline remains in its original position, regardless of the type and direction of expansion. If the specification is omitted, or if the factor *n* is illegal, the printer retains its previous setting.
HORIZONTAL n	The characters are expanded in width by the factor *n*.
VERTICAL n	The characters are expanded in height by the factor *n*.
SIGN	Activates or deactivates "flashing" output on a terminal (SIGN has no effect because screen outputs are not yet possible).

BOLD n

Activates and deactivates bold type during printing.

n = 0: deactivates bold type
n = 1: activates normal bold type
n = 2: activates extra-bold type

If the printer does not support extra-bold type, ordinary bold is used in its place.

If the printer does not support bold type, ordinary type is used.

NORMALINTENSITY n

Switches between normal and half intensity video when output is sent to the screen.

n = 0: activates normal intensity video
n = 1: activates half intensity video

(Has currently no effect as screen output is not yet possible.)

INVERSE n

Activates inverse mode for output.

n = 0: deactivates inverse mode
n = 1: activates inverse mode

If the printer does not support inverse mode, ordinary type is used.

ITALIC n

Activates and deactivates italic type during printing.

n = 0: deactivates italic type
n = 1: activates normal italic type
n = 2: activates sharply inclined italic type

If a printer cannot print italics with strong inclination, the output is made with normal italics. If the printer does not support italics, the entry is ignored.

PROPORTIONAL

Activates or deactivates proportional spacing.

If proportional spacing is activated, a character spacing specified with CHARACTER DISTANCE *n* is ignored.

SUBSCRIPT

Subscripts type by half a line or resets to normal type. This entry is **not** supported in the GLOBAL LAYOUT statement.

If the printer does not support subscript, the entry is ignored.

SUPERSCRIPT

Superscripts type by half a line or resets to normal type. This entry is **not** supported in the GLOBAL LAYOUT statement.

If the printer does not support superscript, the entry is ignored.

UNDERLINE n

Activates or deactivates underlined type during printing.

n = 0: deactivates underlined type
n = 1: activates underlined type.

If the underlining function is activated, all characters, including blanks, are underlined. The position and thickness of the underline depend on the font *char-literal* selected and cannot be changed.

If the printer does not support the underline function, the entry is ignored.

COLOUR

Switches to a foreground or background color. Foreground color is taken to mean the color with which, for example, texts and lines are output. Areas containing neither text nor lines can be filled in with a background color.

FOREGROUND char-literal

Specifies the color with the name *char-literal* to be the foreground color.

BACKGROUND char-literal

Specifies the color with the name *char-literal* to be the background color.

If the printer supports a grid generator, a name for a grey shade or a shading pattern can be specified with *char-literal*.

PAPER SOURCE char-literal

Specifies the printer's paper tray.
During printing, paper is fed from the paper tray specified in *char-literal*.
Refer to the corresponding device profile for the specifications permitted for individual printers.

REPORT
Define list control block

REPORT is used to describe the list header and list trailer of a report. A list header appears just once at the beginning of a report; it can, for example, contain a title for the whole report. A list trailer is likewise output once only at the end of a report; report results, for example, can be included in the trailer.

The REPORT statement is only allowed once within a report definition and may only be specified once for each list header and list trailer.

A list header or trailer can include the following output:

– literals and texts inserted with SOURCE
– the system variables &PAGES and &LINES
– start parameters
– arithmetic expressions
– local variables.

Moreover, the data in the first data record can be contained in the list header, and the values derived from report set functions as well as the data in the last data record can be contained in the list trailer.

```
REPORT { HEADER | TRAILER } [ line-pattern ] { statement; } ...
```

HEADER	Defines the list header which is described by the following statements.
TRAILER	Defines the list trailer which is described by the following statements.
line-pattern	Name of a line background pattern which must have been defined using GLOBAL LINE BASE. The corresponding line pattern is assigned to the list header or trailer as a background pattern.
statement	Designates one of the DRIVE statements BREAK, CALL, CYCLE, IF, SET or one of the report statements PRINT or SOURCE. You use these statements to describe the page trailer or header. The statements are subject to the restrictions described elsewhere (see section “Permitted DRIVE statements” on page 213 and section “Restrictions applicable to report parameters” on page 214).

Example

A list header is to contain the name of a "Market Research PLC" at the top-left margin and the current date at the top-right margin. The title of the report is to be output below these and is to be centered on the page:

```
...
REPORT HEADER
      PRINT NEWLINE 1,
            SET (ITALIC 1, FONT 'CENTURY CONDENSED', CHAR DENSITY 10),

            TAB 5,  'Market Research PLC',
            TAB 55, 'Date: ', CURRENT DATE MASK 'DD''.''MO''.''YYYY',
            NEWLINE 1,
            TAB 5,  'Beaver Estate Unit 4',
            NEWLINE 1,
            TAB 5,  'Baltimore',
            NEWLINE 4,

            SET (ITALIC 1, FONT 'COURIER', NATIONAL SET 'GERMAN',
                 EXPANSION HORIZONTAL 1, EXPANSION VERTICAL 1, BOLD 1),

            TAB 8,  '*** ',
                    'CONSUMER BEHAVIOUR' ATTRIBUTE (UNDERLINE 1),
                    ' ***',
            NEWLINE 3;
...
```

OUTPUT:

Market Research PLC *Date: 13.03.1992*
Beaver Estate Unit 4
Baltimore

*** CONSUMER BEHAVIOUR ***

SOURCE
Insert text file

SOURCE is used to insert text files within a report. The report generator outputs the contents of the specified file at the current output position without changing them. If the text covers more than one page, no page headers or trailers are output.

```
SOURCE file [ n [ n ] ]
```

---

file — Designates the file which contains the text for insertion.

PostScript files must be present in Encapsulated PostScript format (EPS format). The file is inserted in such a way that the top left-hand corner of the bounding box which is specified in the file is located at the current output position.

n — The first *n* gives the number of lines on the last page of the text in *file-name*. The second *n* indicates the number of pages. If the number of pages is not specified, the report generator interprets the first *n* as the number of lines in the whole file.

Using *n*, the report generator can adapt internal page and line counters to take into account the inserted source. As a result, correct page breaks and page numbers can be generated after output of the text.

Example

The data in the variables &key, &artname and &price are to be output in a row in a list. The contents of the file "d.report.source" are to follow each of these output lines.

```
...
DETAIL
  PRINT NEWLINE 1,
        TAB  2, &key,
        TAB 10, &artname,
        TAB 40, &price;

  SOURCE "d.report.source";
...
```

The filename has to be enclosed in quotes because it contains special characters.

STANDARD LAYOUT
Describe layout of standard report

STANDARD LAYOUT is used to generate a standard report. The statement must be entered between the statements DECLARE REPORT and END REPORT. In the case of a standard report, the report definition is derived from the record description of the parameters which are adopted by the report generator from the DRIVE program.

Three predefined formats are provided for the layout of a standard report:

TABLE	output in tabular format
LINE	output in line (vertical) format
SEQUENCE	output in sequential (horizontal) format.

```
STANDARD LAYOUT { TABLE [ FILL char-literal ] |
                  LINE |
                  SEQUENCE }
```

TABLE

Requests tabular format.
In tabular format, a title line is generated with the names of the components of the lowest level. The field contents are listed under each corresponding name.

The amount of space available in a line for the field contents of a component depends either on the length of the name or on the longest field contents. The column width is calculated according to the longer of the two.

If the sum of all the column widths in a table line exceeds the width that can be output at the selected device, the output line is cut off at the right-hand edge and the DRIVE program is aborted with an error following the CLOSE REPORT statement.

Matrices and repeating groups cannot be output in TABLE format.

FILL char-literal

Defines the characters *char-literal*, which are used for separating output fields adjacent to each other.

Default: a blank.

LINE

Requests vertical format. In this format, components of data groups (see metavariable *data-group*) and repeating groups (see metavariable *repeating-group*) are output underneath one another. Components of basic data type with a repetition factor are output in the form *element-name: value1 value2* ... . The components of repeating groups may not contain any repetition factor, that is to say they must be one-dimensional. Matrices are output in matrix form. The matrix name appears at the start of every matrix line.

Adjacent output fields are separated by a blank (default).

In the case of a line overflow the output line is cut off at the right-hand edge and the DRIVE program is aborted with an error following the CLOSE REPORT statement.

SEQUENCE

Requests horizontal format. The output data is written sequentially in the form *name: value* or *component-name[(n)]: value* in a single line.

The output of matrices and repeating groups is not supported.

Adjacent output fields are separated by a blank (default).

In the case of a line overflow, the output line is cut off at the right-hand edge and the DRIVE program is aborted with an error following the CLOSE REPORT statement.

Example

The following variable definition demonstrates the three formats which you can request in the STANDARD LAYOUT statement:

```
/* The variables which are to be processed are declared in the DRIVE program */

DECLARE VARIABLE
  1 &address(2),
    2 town(2)     CHAR(8),
    2 tel         CHAR(7);
.
.
/* The description of the DRIVE data is                                    */
/* passed to the report definition. The repetition factor is not passed  */

DECLARE REPORT location USING &data LIKE &address;

    STANDARD LAYOUT TABLE FILL '|';    /* for output in tabular format       */

    or

    STANDARD LAYOUT LINE;              /* for output in vertical format      */

    or

    STANDARD LAYOUT SEQUENCE;          /* for output in horizontal format    */

END REPORT;

/* The report is opened in the DRIVE program, filled with data       */
/* and closed.                                                       */

OPEN REPORT location RESULT LIST 'print' DEVICETABLE '9002';
   FILL REPORT location USING &address(1);
   FILL REPORT location USING &address(2);
CLOSE REPORT location;
```

Output with STANDARD LAYOUT TABLE

```
1994 Sep 12  16 : 08   -   Standard Report TABLE   -   Page:   1

TOWN(1)    TOWN(2)    TEL
————————————————————————
BOSTON  | NEWARK  | 1234567
NEWARK  | SEATTLE | 1234567
```

Output with STANDARD LAYOUT LINE

```
1994 Sep 12  16 : 15   -   Standard Report LINE   -   Page:   1

1 DATA
2 TOWN: BOSTON NEWARK
2 TEL: 1234567
1 DATA
2 TOWN: NEWARK   SEATTLE
2 TEL: 1234567
```

Output with STANDARD LAYOUT SEQUENCE

```
1994 Sep 12  16 : 23   -   Standard Report SEQUENCE   -   Page:   1

TOWN(1): BOSTON   TOWN(2): NEWARK  TEL: 1234567
TOWN(1): NEWARK   TOWN(2): SEATTLE TEL: 1234567
```

If you choose the format STANDARD LAYOUT LINE you can also output repeating groups. However, the components of these repeating groups may not contain any repetition factors (i.e. the components must be one-dimensional).

```
DECLARE VARIABLE
  1 &address(2),
    2 town        CHAR(20),
    2 tel         CHAR(7);
.
.

DECLARE REPORT location USING 1 &data(2),
                              2 town   CHAR(20),
                              2 tel    CHAR(7);
      STANDARD LAYOUT LINE;
END REPORT;

OPEN REPORT location RESULT LIST 'print' DEVICETABLE '9002';
   FILL REPORT location USING &address;
CLOSE REPORT location;
```

Output

```
1994 Sep 12  17 : 19  -   Standard Report LINE   -   Page:   1

1 DATA (1)
2 TOWN: BOSTON    NEWARK
2 TEL: 1234567
1 DATA (2)
2 TOWN: NEWARK    SEATTLE
2 TEL: 1234567
```

5 DRIVE metavariables

This chapter describes complex parts of DRIVE statements in alphabetical order. These parts possess the following hierarchical relationship:

attribute

expression
- char-expression
  - char-prim
- date-time-expression
  - date-time-term
- interval-expression
  - interval-term
- value-expression
  - num-term
    - set-function
    - value-function

base-type
- INIT clause*
- REDEFINES clause *
- check
- mask

condition

data-type
- basic-data-type
- structure-type
  - vector
  - data-group
  - repeating-group
  - user-type *
  - LIKE clause *
- matrix *

date-time-unit
- date-time-field

format

null-value

programming

value
- char-prim
- variable
- literal
- aggregate *

The statement parts which are marked with a * are not described as independent metavariables but can be found in the description of the next higher ranking metavariable.

attribute
Describe field attribute

attribute describes the field properties of forms. (For detailed information see IFG for FHS [28], FHS [29] as well as FORMANT [39] and UTM Formatting System [32].)

```
attribute::=[ UNPROTECTED | PROTECTED ] |
            [ HIGHINTENSITY | NORMALINTENSITY ] |
            [ VISIBLE | SIGN | INVISIBLE ] |
            [ INIT | NOINIT ] |
            [ VALID | INVALID ] |
            [ HARDCOPY | ALARM | DEFAULT ] |
            [ MUST | POTMUST | NORMALINPUT ] |
            [ INVERSE | NOINVERSE ] |
            [ UNDERLINE | NOUNDERLINE ] |
            [ CURSOR | NOCURSOR ] |
            [ BLUE | CYAN | GREEN | MAGENTA | RED | WHITE | YELLOW | NOCOLOUR ]
```

Field properties can be subdivided into two groups, namely field attributes which relate to a particular screen field and global attributes which relate to the entire screen. Field attributes are combined to form field attribute groups.

Field attributes

A field attribute may relate to one screen field or one variable since screen fields can only be addressed via variables.

Field attribute group affecting input

MUST	Mandatory input: An entry must be made in the field.
POTMUST	One-time mandatory input, i.e. in the case of repeated input the entry need not be made once again.
NORMALINPUT	Input not mandatory.
UNPROTECTED	Unprotected field: The field contents can be changed via the keyboard. Any character can be entered.
PROTECTED	Protected field: The field contents must not be changed via the key.

Field attribute group affecting display

HIGHINTENSITY	High intensity: This value results in high intensity display on the screen, or bold type in the case of printer terminals (provided the device supports this function). Default value for identifying errors in screen fields.
NORMALINTENSITY	Normal intensity.
VISIBLE	The field contents are completely visible.
SIGN	The field contents are displayed flashing; in the case of output to printers this value causes shadow printing (provided the device supports this function).
INVISIBLE	The field contents are not visible and not printable on hardcopy.
UNDERLINE	The field contents are displayed underlined. Default value for identifying screen fields containing errors.
NOUNDERLINE	The field contents are not displayed underlined.
INVERSE	The field contents are displayed in inverse video.
NOINVERSE	The field contents are not displayed in inverse video.

Field attribute group affecting cursor

CURSOR	The cursor is positioned to the first input position in the field. CURSOR may also be specified as a global attribute. FHS only evaluates the CURSOR field attribute if the CURSOR global attribute is specified at the same time.
NOCURSOR	The cursor is not positioned to the field. NOCURSOR can also be specified as a global attribute.

Field attribute group affecting color

BLUE through NOCOLOUR

A color may be specified for data output on multicolor terminals:

BLUE - blue
CYAN - cyan (blue-green)
GREEN - green
MAGENTA - magenta (violet)
RED - red
WHITE - white
YELLOW - yellow
NOCOLOUR - no color, i.e. normal screen display

Further field attributes

VALID	The field contents are checked and without error, i.e. no edit routine is required.
INVALID	The field contents are checked and contain errors.

Global attributes

Global attributes each refer to a screen form, i.e. they affect the complete screen.

INIT	Data transfer areas that already contain data are reset to their initial states; thus, all field attributes in the form are set to their default values.
NOINIT	Output without initialization.
HARDCOPY	Automatic hardcopy: Following output of the message, the contents of the entire screen are output automatically on the hardcopy device (provided the terminal has been generated with hardcopy).
ALARM	Output with alarm. This global attribute functions only on devices that support an alarm function (optical and/or acoustical).
DEFAULT	Reset all attributes to their default values. DEFAULT affects all addressing aid field attributes. All field attribute groups (with the exception of the field attributes VALID and INVALID) are reset to their default values (for information on the default values of the individual field attributes, refer to the manual "XXX UTM (SINIX): Formatting System" [XXX]). Global attributes are not reset to their default values.

CURSOR	The field attribute CURSOR is evaluated.
NOCURSOR	The field attribute CURSOR is not evaluated.

Possible combinations of field attributes (for dynamic attributes)

UNPROTECTED may be combined with all other field attributes in any way desired. For the other field attributes, the following combinations are possible:

Field attribute	Permitted combinations							Group
HIGHINTENSITY	*	*				*		1
NORMALINTENSITY			*	*			*	
VISIBLE	*	*	*	*				
SIGN						*	*	
INVISIBLE					*			
UNDERLINE	*		*					
NOUNDERLINE		*		*				
GREEN		*						2
RED	*							
WHITE			*					
YELLOW				*				

On a color screen, the specifications for intensity, visibility and underlining (group 1) are represented in the colors from the table (group 2). Specifying HIGHINTENSITY, VISIBLE, UNDERLINE therefore has the same effect as specifying RED.
Likewise, if RED is specified on a monochrome screen, this is converted to the combination HIGHINTENSITY, VISIBLE, UNDERLINE.

Only field attributes from group 1 or group 2 may be specified. In cases of duplicate specification, field attributes from group 1 take precedence over field attributes from group 2.

base-type
Define clause

base-type defines the following clauses:

INIT clause	Assigns an initial value to a variable.
REDEFINES clause	A number of different descriptions are specified for the memory area of a variable.
CHECK clause	A condition is declared for a variable and checked during execution of the program (see metavariable *check*).
MASK clause	The way in which data values are represented is defined for input and output fields (see metavariable *mask*).

For additional data definitions, see the metavariable *data-type*.

```
base-type::={ INIT expression [ NOCHECK ] |
              REDEFINES { variable | character [ suffix ] } }
            [ check ] [ mask ]
```

INIT	INIT assigns an initial value to a variable.
expression	*expression* can only contain literals, NULL or functions whose arguments are literals. *expression* must not contain CURRENT DATE / TIME / TIMESTAMP. *expression* must be able to be calculated at compilation time.
NOCHECK	NOCHECK causes the CHECK clause to be ignored for the initial value assigned with INIT *expression* (see metavariable *check*). NOCHECK is not permitted for redefined variables or variables which define another variable.
REDEFINES	Identifies the variable being redefined (base variable). The variable being redefined must not be longer than the base variable. If the variable being redefined is a component of a structure, the base variable must be a component of the same base structure (structure with the level number 1). The redefined variable may not have the data type VARCHAR.

variable	Name of the base variable being redefined. It must already have been defined and must not itself be a redefining variable. If the base variable is a structure, the redefining variable must not be a component of that structure. There must be no repeating groups between the beginning of the smallest substructure containing both the base variable and the redefining variable and the base variable or redefining variable itself.
character	Character string for the name of a component. 0 < number(*character*) < 32;
suffix	See the metavariable *variable*.
check	See the metavariable *check*.
mask	See the metavariable *mask*.

Examples

REDEFINES for another variable:

```
DECLARE VARIABLE &a CHAR (8),
                 &b CHAR (2) REDEFINES &a;
```

&b reassigns the first 2 bytes of the area assigned to &a.

REDEFINES and INIT:

```
DECLARE VARIABLE &a CHAR (8),
                 &b CHAR (2) INIT 'INVALID' REDEFINES &a;
                                            *
```

□ Error: not permitted with INIT

basic-data-type
Data types

The basic data type can be subdivided into four groups:

– Character-string (CHARACTER, CHARACTER VARYING, VARCHAR)
– Numeric (DECIMAL, EXTENDED DECIMAL, XDEC, NUMERIC, INTEGER, SMALLINT, REAL, FLOAT, DOUBLE PRECISION)
– Date-time data types (DATE, TIME, TIME(3), TIMESTAMP(3))
– Data type INTERVAL
– User-defined data type

These basic data types are also referred to as "atomic types".

For additional data definitions, see the metavariable *data-type*.

```
basic-data type::={ CHARACTER [ ( length ) ] |

                  { DECIMAL | NUMERIC | EXTENDED DECIMAL | XDEC }
                                    [ ( number-of-digits [, decimal-places] ) ] |

                  INTEGER |
                  SMALLINT |
                  DATE |
                  TIME [ (3) ] |
                  TIMESTAMP(3) |
                  INTERVAL date-time-unit |
                  CHARACTER VARYING ( length ) |
                  VARCHAR ( length ) |
                  REAL |
                  DOUBLE PRECISION |
                  FLOAT |
                  user type }
```

CHARACTER	Specifies the data type character-string for a string.
length	Specifies the length for the data type CHARACTER (0 < *length* ≤ 32000). Default value: 1
DECIMAL	Specifies the numeric packed data type for a string.

NUMERIC
Specifies the numeric unpacked data type for a string.

EXTENDED DECIMAL, XDEC
Specifies the data type extended decimal for a string.

number-of-digits
Specifies the number of digits (≙ precision) for the data types DECIMAL, NUMERIC, EXTENDED DECIMAL and XDEC.

DECIMAL and NUMERIC (0 < *number-of-digits* ≤ 15)
Default value for DECIMAL: 15
Default value for NUMERIC: 8

EXTENDED DECIMAL and XDEC (0 < *number-of-digits* ≤ 32)
Default value for EXTENDED DECIMAL and XDEC: 32

decimal-places
Specifies the number of decimal places for the data types DECIMAL, NUMERIC, EXTENDED DECIMAL and XDEC.

The following applies for DECIMAL and NUMERIC:
0 ≤ *decimal-places* ≤ *number-of-digits*.
If *number-of-digits* = 15, *decimal-places* must be less than *number-of-digits*.

The following applies for EXTENDED DECIMAL and XDEC gilt:
0 ≤ *decimal-places* ≤ *number-of-digits*.
If *number-of-digits* = 32, *decimal-places* must be less than *number-of-digits*.

Default: 0

INTEGER, SMALLINT
Specifies the data type integer for a string.

The value range of INTEGER lies between -2^{31} and $2^{31} - 1$, i.e. between -2147483648 and 2147483647.
The value range of SMALLINT lies between -2^{15} and $2^{15} - 1$, i.e. between -32768 and 32767.

DATE
The data type "DATE" is specified for a string. The string may only contain valid date specifications in the format *year-month-day* (0001-01-01 through 9999-12-31, see *date-time-literal* under the metavariable *literal*).

Negative year specifications are not valid.

TIME
The data type "TIME" is specified for a string. The string may only contain valid time specifications in the format *hour*:*minute*:*second* (00:00:00 through 23:59:59, see *date-time-literal* under the metavariable *literal*).

TIME(3)
: The data type "TIME(3)" is specified for a string. The string may only contain valid time specifications in the format *hour*:*minute*:*second*.*fraction* (00:00:00.000 through 23:59:59.999, see *date-time-literal* under the metavariable *literal*).

TIMESTAMP(3)
: The data type "TIMESTAMP(3)" is specified for a string. The string may only contain valid timestamp specifications in the format *year*-*month*-*day hour*:*minute*:*second*.*fraction* (0001-01-01 00:00:00.000 through 9999-12-31 23:59:59.999, see *date-time-literal* under the metavariable *literal*).

  Negative year specifications are not valid.

INTERVAL
: The data type "INTERVAL" is specified for a sequence of data (up to 32 characters).

  mask cannot be used to represent the entry or output fields.

date-time-unit
: Specifies the unit for the interval (see the metavariable *date-time-unit*).

  You may only specify one component of a date or time (*date-time-field2* must be identical with *date-time-field1*, see metavariable *date-time-unit*).

CHARACTER VARYING, VARCHAR
: Specifies an alphanumeric data type with a variable number of characters for a string.

length
: Specifies the length of a data type ($0 < length \leq 32000$).

REAL
: Specifies the floating point numeric data type with a precision of seven decimal places for a string.
  The range of values for REAL is subject to the hardware used.

  If the accuracy of an arithmetic operation is of prime importance, you should not select the data type REAL. Inaccuracies may occur when assigning and outputting a data value.

DOUBLE PRECISION, FLOAT
: Specifies the floating point numeric data type with a precision of 15 decimal places for a string.
  The range of values for DOUBLE PRECISION is subject to the hardware used.

user-type
: User-defined data type which must have been declared with DECLARE TYPE in the DRIVE program. *user-type* does not automatically create a structure.

  user-type is not permitted in SQL statements.

char-expression
Define a character expression

char-expression defines character expressions. The data type of char-expression must be alphanumeric.

char-expression can also be an empty string. The maximum length is 32000 characters.

See the metavariable *expression* for other expressions.

```
char–expression::={ char–prim | char–expression || char–prim }
```

---

char-prim	See the metavariable *char-prim*.
char-expression	*char-expression* can also be a literal or a variable. If *char-expression* has the value null, *char-prim* also has the value null.
||	The concatenation operator (||) is used to join character strings together. The second string is appended to the first. The maximum length of a concatenated string is 32000 bytes. The expressions used must contain neither structured variables nor aggregates.

char-prim
String functions

char-prim is used to perform the following functions:

- Concatenating strings
- Selecting substrings
- Replacing and modifying substrings
- Deleting substrings
- Left-justified output of substrings
- Converting characters to character strings
- Outputting messages from a message file
- Concatenating all the atomic fields of structured variables

---

```
char-prim::={ value1 |
            column |
            ( char-expression ) |
            CONCAT ( char-expression1, char-expression2 ) |
            SUBSTRING ( char-expression3, start-pos1 [, length1 ] ) |
            UPDSTRING ( char-expression4, char-expression5, upd-pos ) |
            DELSTRING ( char-expression6, start-pos2 [, length2 ] ) |
            SHIFTLEFTSTRING ( char-expression7 ) |
            UPPERSTRING ( char-expression8 ) |
            LOWERSTRING ( char-expression9 ) |
            TRSTRING ( char-expression10, char-expression11, char-expression12) |
            MSGSTRING ( value-expression1 [ [, value-expression2 ], name ] ) |
            CHARACTER ( { date-time-expression [, char-literal1 ] |
                          value2 |
                          value-expression3 [, char-literal2 ] } ) }
```

---

value1	The data type of *value1* must be CHARACTER (see the metavariable *value*). An aggregate must not be specified for *value1*.
column	Name of a column of type CHARACTER or VARCHAR
char-expression	If *char-expression* has the null value, *char-prim* also has value null (see the metavariable *char-expression*).

CONCAT

Can be used to concatenate strings. The maximum length of a concatenated string is 32000 bytes.

The expressions used must contain neither structured variables nor aggregates.

If one of the arguments contains NULL, CONCAT returns NULL.

char-expression1, char-expression2

The string *char-expression2* is appended directly to the string *char-expression1*.

If one of the two expressions (*char-expression1* or *char-expression2*) contains NULL, the entire expression contains NULL.

SUBSTRING

Can be used to select a substring from a string. The substring begins with the character which is at the *start-pos1* position and ends at the end of the entire string or after the specified length. (*start-pos1* + *length1* - 1) must not be longer than the length of the entire string.

The expressions used must contain neither structured variables nor aggregates.

If one of the arguments contains NULL, SUBSTRING returns NULL.

char-expression3

See the metavariable *char-expression*.

start-pos1

Start of the substring. *start-pos1* must be a numeric expression with no decimal places.

start-pos1 must be greater than 0.

length1

Length of the substring. If *length1* is not specified, it is calculated from (total length of string - *start-pos1* + 1). *length1* must be a numeric expression with no decimal places.

length1 must be greater than 0 and the following must be true:
0 < *start-pos1* ≤ length (*char-expression3*)

UPDSTRING

A substring of a string is substituted by another string, where the length of *char-expression4* ≥ the length of *char-expression5* + *upd-pos* -1.

The expressions used must contain neither structured variables nor aggregates.

If one of the arguments contains NULL, UPDSTRING returns NULL.

char-expression4, char-expression5

See the metavariable *char-expression*.

upd-pos	Update position. *upd-pos* must be a numeric expression without decimal places. (0 < *upd-pos* ≤ length (*char-expression4*) - length (*char-expression5*) + 1.
DELSTRING	Can be used to delete substrings. The parameters used must contain neither structured variables nor aggregates. If one of the parameters has the null value, the entire expression has the null value.
char-expression6	See the metavariable *char-expression*.
start-pos2	Start of the substring. *start-pos2* must be a numeric expression without decimal places. The value of *start-pos2* must be greater than zero, but may not exceed *expression* in length.
length2	Must be a numeric expression without decimal places.
	Specification of *length2* causes the characters from the *start-pos2* to the value of *length2* to be deleted. DELSTRING (*char-expression6*, *start-pos2*, *length2*) is equivalent to SUBSTRING (*char-expression3*, 1, *start-pos1 -1*) || SUBSTRING (*char-expression3*, *start-pos1* + *length1*, *length of char-expression3* - *length1* -*start-pos1* + *1*).
	If *length2* is not specified, the result is a character string which extends only as far as *start-pos2*, i.e. all characters from *start-pos2* onwards are deleted.
	DELSTRING (*char-expression6*, *start-pos2*) is equivalent to SUBSTRING (*char-expression3*, 1, *start-pos1* - *1*).
SHIFTLEFTSTRING	All left-justified characters are deleted whose hexadecimal representation is less than or equal to a blank.
	If *char-expression7* contains NULL, SHIFTLEFTSTRING returns NULL.
char-expression7	If *char-expression7* is of data type CHARACTER, it is padded from the right with blanks. If *char-expression7* is of data type VARCHAR or CHARACTER VARYING, its length is reduced to comply with the length specification of these data types.

UPPERSTRING	All lowercase characters in *char-expression8* are replaced by uppercase characters. All other characters remain unchanged. Conversion is carried out in accordance with the country-specific settings in the current system environment. The expressions used must not contain structured variables or aggregates. If *char-expression8* contains NULL, UPPERSTRING returns NULL.
char-expression8	See metavariable *char-expression*
LOWERSTRING	All uppercase characters in *char-expression9* are replaced by lowercase characters. All other characters remain unchanged. Conversion is carried out in accordance with the country-specific settings in the current system environment. The expressions used must not contain structured variables or aggregates. If *char-expression9* contains NULL, LOWERSTRING returns NULL.
char-expression9	See metavariable *char-expression*
TRSTRING	The characters in *char-expression10* are converted as follows: a character *c* from *char-expression10* is only converted if it also occurs in *char-expression11*. The position *n* of the character in *char-expression11* defines the character with which it is to be replaced. The replacement character is located at position *n* in *char-expression12*. A character is not converted if its position in *char-expression10* is greater than the length of *char-expression11*. The expressions used must not contain structured variables or aggregates. If one of the arguments contains NULL, TRSTRING returns NULL.
char-expression10	Characters in *char-expression10* are converted one-by-one. Characters are converted if they occur in *char-expression11* and if they are located at a position *n* in *char-expression11* which is less than the number of characters in *char-expression12*. Characters remain unchanged if they do not occur in *char-expression11* or if they are located at a position *n* in *char-expression11* which is greater than the number of characters in *char-expression12*.

char-expression11

char-expression11 defines the characters to be replaced in *char-expression10*. The position of the character in *char-expression11* defines the replacement character.

Characters in *char-expression11* must not be repeated.

char-expression11 containing the characters to be replaced and *char-expression12* containing the replacement characters should be of the same length. If *char-expression11* is longer than *char-expression12*, characters whose position in *char-expression11* is greater than the length of *char-expression12* are not converted.

char-expression11 must not be longer than 256 characters.

char-expression12

char-expression12 defines the replacement characters. The position of the character to be replaced in *char-expression11* determines the position of the replacement character in *char-expression12*.

char-expression12 containing the replacement characters and *char-expression11* containing the characters to be replaced should be the same length. If *char-expression12* is shorter than *char-expression11*, characters whose position in *char-expression11* is greater than the length of *char-expression12* are not converted.

char-expression12 must not be longer than 256 characters.

MSGSTRING

Accesses a message in the message file (= current MIP file). The message is determined by the parameters.

If you specify three parameters (*value-expr1, value-expr2, name*) then *value-expr2* is ignored.
If you specify two parameters then the second parameter must be *name* (*value-expr1, name*).

If no unique message can be found, DRIVE/WINDOWS returns the message: `MESSAGE UNDEFINED`.

The expressions used must not contain structured variables or aggregates.

If one of the arguments contains NULL, MSGSTRING returns NULL.

With DECLARE statements (e.g. DCL VAR, DCL CONST etc.), messages are accessed at compile time, in executable statements (e.g. SET) at execution time. In completely language-independent programs, access must take place at execution time.

value-expression1 Message number (= second part of message key).

You do not need to specify leading zeroes in the message number.

value-expression1 must be a numeric expression without decimal places.

value-expression2 *value-expression2* is ignored if a value is specified.

value-expression2 is supported simply to ensure DRIVE/WINDOWS (BS2000) and DRIVE/WINDOWS (SINIX) compatibility.

name Message class (= first part of message key). *name* may be a maximum of three characters in length.

If you do not specify *name* then DRIVE/WINDOWS uses the name DRI.

CHARACTER The argument (*date-time-expression*,*value2*,*value-expression3*) is converted to a result of the type CHARACTER. The data type of the expression is alphanumeric, i.e. digits and separators are shown as characters.

The length of a string for a date is ten characters (e.g. '1911-11-11'),
for a time without fractions of a second it is eight characters (e.g. '17:35:12'),
for a time with fractions of a second it is twelve characters (e.g. '17:35:12.361') and
for a timestamp it is 23 characters (e.g. '1911-11-11␣17:35:12.361').

If one of the arguments contains NULL, CHARACTER returns NULL.

date-time-expression See metavariable *date-time-expression*

char-literal1, char-literal2

char-literalx must meet the *mask* conditions. If the first component of the expression is a *date-time-expression* then *mask-literal* must contain control characters for the time data types. If you do not specify *char-literalx* a standard mask is used (see also metavariable *literal*).

value2 *value* must be a data group or an aggregate. The data group must not be part of a repeating group, nor may it contain a repeating group. The data type of all base fields must be alphanumeric. If a variable is specified for *value*, *char-prim* is a concatenation of all the base fields (see CONCAT above).

value-expression3 See metavariable *value-expression*

Example 1

```
DECLARE VARIABLE &a CHAR (7)  INIT 'LONDON␣',
                  &b CHAR (11) INIT 'POLICEWOMAN';

CONCAT (&a,&b)   → 'LONDON␣POLICEWOMAN'

SUBSTRING (&b,7)   → 'WOMAN'

UPDSTRING (&b,'MAN␣␣',7)   → 'POLICEMAN␣␣'
```

Example 2

```
DECLARE VARIABLE &a CHAR (250),
                  &b CHAR (250);
...
SET &a = SUBSTRING(CONCAT(&a,&b), 180, 200);
```

Example 3

The value of "MSGSTR (17)" is assigned at compile time.

```
DECLARE CONSTANT &c MSGSTR (17);
SET &v = &c;
```

Example 4

The value of "MSGSTR (17)" is assigned at runtime.

```
...
SET &v = MSGSTR (17);
...
```

Example 5

The data type of the result of an arithmetic operation is alphanumeric.

```
DECLARE VARIABLE &e CHAR (8);
...
SET &e = CHARACTER (5967 / 17);
```

Example 6

The characters from the variable &text are to be converted. The variable with the characters to be replaced is &char-old and the variable with the replacement characters is &char-new.

```
DECLARE VARIABLE &text CHAR (6)  INIT 'PEKING';
DECLARE VARIABLE &char-old CHAR (5)  INIT 'JKPTU',
                 &char-new CHAR (5)  INIT 'IJBDO';
...
SET &text = TRSTRING(&text,&zeichenalt,&zeichenneu);
```

The original contents of the variable &text ("PEKING") have been converted to "BEJING".

check
Define a check clause

A CHECK clause specifies a check condition. If, at runtime, the check condition for a variable is not assigned when a value is assigned (SET, CYCLE ... INTO, SELECT ... INTO, FETCH ... INTO, CALL ... RETURN, FILL ... RETURN etc.), an entry is made in &ERROR (see the WHENEVER statement).

If the check condition for the input in a DISPLAY statement is not fulfilled, the screen is output again.

check applied to a component of a structured variable

If the component is part of a repeating group, the CHECK clause applies to all occurrences of the component.
Individual vector or matrix components must not be specified in the CHECK clause.

```
check::=CHECK [ ( ] condition [ ) ] [ MESSAGE char-expression ]
```

CHECK	The CHECK clause must be compatible with the INIT clause or the standard initialization, or NOCHECK must be specified for *base-type*.
( )	The parentheses must be specified in SQL statements for SESAM V2.x
condition	Check condition. *condition* must contain only the variable to which the CHECK clause belongs. The variable must not contain any index specifications. The variable need not be qualified, even if ambiguous. No structured comparison is possible for this variable. The *condition* may contain the keyword VALUE instead of the variable. For use in a DECLARE TYPE statement, refer to the description of the DECLARE TYPE statement.

MESSAGE

The MESSAGE clause is used in form input to specify a message that is output in place of the standard message when *condition* is not satisfied during dynamic form generation or in the MOVE DATA statement.

With SET statements and parameter passing, a standard message is output. Any MESSAGE specification is ignored.

The MESSAGE clause must not be specified in SQL statements for SESAM V2.x.

char-expression

char-expression defines the message for output (maximum 79 characters).

The message is only output if the *condition* is not satisfied.
In the *char-expression* of MESSAGE, *value* may only contain the keyword VALUE or a literal.

condition
Define condition

A condition consists of one or more logical expressions and the logical operators AND, OR or NOT.

The following conditions can be formulated:

- Compare expressions by means of comparison operators (see page 296)
- Compare an expression with a value range (see page 297)
- Compare an expression with a list of values (see page 298)
- Check whether a value contains the null value (see page 300)
- Check whether a string is numeric. (see page 301)

Notes on evaluating conditions

- null values in conditions

  If null values occur in conditions, the result of the condition can be satisfied, not satisfied or unknown. The circumstances under which these possible results apply is given in the description of each condition.

- Comparison of alphanumeric values

  Two strings are compared from left to right. If the strings are of different lengths, the shorter string is padded with blanks. Two character strings are equal if they contain the same characters at every position. Otherwise, the first character which differs between the two strings defines which of the strings is greater.

- Comparison of numeric values

  Two numeric values are equal if they have the same sign and the same quantity.

condition has the following syntax:

```
condition::=[ NOT ] { ( condition1 [ { AND | OR } condition2 ) ] ... ) |
                      ( expression1 { = | < | > | <> | <= | >= } expression2 ) |
                      ( expression3 [ NOT ] BETWEEN expression4 AND expression5 ) |
                      ( expression6 [ NOT ] IN ( value, ... ) ) |
                      ( value IS [ NOT ] NULL ) |
                      ( char-expression IS [ NOT ] NUMERIC ) }
```

The following results are possible for *condition* if AND, OR or NOT are applied:

AND

Both conditions combined by AND must be satisfied for the entire condition to be satisfied.

	condition2		
condition1	satisfied	not satisfied	unknown
satisfied	satisfied	not satisfied	unknown
not satisfied	not satisfied	not satisfied	not satisfied
unknown	unknown	not satisfied	unknown

OR

At least one of the two conditions combined by OR must be satisfied for the entire condition to be satisfied.

	condition2		
condition1	satisfied	not satisfied	unknown
satisfied	satisfied	satisfied	satisfied
not satisfied	satisfied	not satisfied	unknown
unknown	satisfied	unknown	unknown

NOT

Negation: a condition preceded by NOT must not be satisfied if the entire condition is to be satisfied. If the result of the condition preceded by NOT is unknown, the result of the entire condition is also unknown.

condition	NOT *condition*
satisfied	not satisfied
not satisfied	satisfied
unknown	unknown

condition — As long as *condition* is satisfied, the required function is executed.

Rules

– The data types of *expression1* and *expression2* must be comparable (both must be of the type, alphanumeric, date-time or INTERVAL).

– When date-time data types are compared, both must be of the same type, i.e. a date, a time or a timestamp.

– If *expression* is a structured value, there may be only one comparison using '=' or '<>'.

– The NULL constant must not be specified.

– When you combine the logical operators AND, OR and NOT, the usual precedence rules apply for evaluation:

  NOT before AND before OR

  If you want to change this order, you must use parentheses. Operators within parentheses take precedence.
  Operators with the same priority are processed from left to right.

  If the *condition* of an IF statement generates the truth value UNKNOWN or FALSE, a branch is made to the ELSE path.

  If the *condition* of a CYCLE statement generates the truth value UNKNOWN or FALSE, execution of the statement is terminated.

Comparing expressions using comparison operators

You can use comparison operators to compare the values of two expressions.

Comparison operator	Meaning
=	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

The condition is satisfied if the comparison is true.

The result of the condition is unknown if at least one of the expressions has the value NULL, Otherwise the condition is not satisfied.

Rules

– The data types of *expression1* and *expression2* must be comparable (both must be of the type, alphanumeric, date-time or INTERVAL).

– When date-time data types are compared, both must be of the same type, i.e. a date, a time or a timestamp.

– Vectors must not occur in a comparison using '<', '>', '<=' or '>='.

– The NULL constant must not be specified.

Example

```
SELECT designation
  FROM project
  WHERE budget >= &minbudget

IF &budget < &var ... THEN
```

Comparing an expression with a value range

You can check whether the value of the expression is within or outside a value range.

```
expression1 [ NOT ] BETWEEN expression2 AND expression3
```

BETWEEN ... AND
: The result of the condition is the same as for the condition *expression2* ≤ *expression1* AND *expression1* ≤ *expression3*.

  The condition is satisfied if the value of *expression1* is within the value range.

NOT BETWEEN ... AND
: The result of the condition is the same as for the condition *expression1* < *expression2* OR *expression1* > *expression3*.

  The condition is satisfied if the value of *expression1* is outside the value range.

Rules

– The data types of *expression1* and *expression2* must be comparable (both must be of the type, alphanumeric, date-time or INTERVAL).

– When date-time data types are compared, both must be of the same type, i.e. a date, a time or a timestamp.

– The NULL constant must not be specified.

– Vectors are not permitted in a comparison with BETWEEN... AND.

Example

```
SELECT last-name, salary
   FROM staff-member
   WHERE salary BETWEEN &lower-limit AND 6000

IF &salary BETWEEN 1000 AND &upper-limit THEN ...
```

Comparing an expression with a list of values

The expression is compared with a value or a list of values.

```
expression [ NOT ] IN ( value, ... )
```

IN	The condition is satisfied if the comparison is true for at least one value. The condition is not satisfied if the comparison is not true for any value. Otherwise, the result of the condition is unknown.
NOT IN	The condition is satisfied if the comparison is true for all values of *expression*. The condition is not satisfied if the comparison is true for at least one value. Otherwise, the result of the condition is unknown.
value	Value with a numeric or alphanumeric data type or the data type INTERVAL specified by a constant or variable (see metavariable *value*).

Rules

– The data type of *expression* must be compatible with the data type of the values specified in IN (numeric, alphanumeric or INTERVAL data type).

– When time data types are compared, all the values specified for IN must be either a date, a time or a timestamp.

– Vectors are not permitted in a comparison with IN.

Examples

```
SELECT dept-staff-no, seq-no, last-name
       FROM staff-member
       WHERE city IN ('Manchester','London', &varcity)
```

```
SELECT dept-staff-no, seq-no, last-name
       FROM staff-member INTO &var
       WHERE city NOT IN ('Birmingham','Liverpool','Bristol', &varcity)
```

```
SELECT staff-no, last-name
       FROM staff-member INTO &var
       WHERE salary IN (SELECT salary
                        FROM staff-member
                        WHERE salary > 600000 AND salary < &max)
```

```
IF &city IN ('Birmingham','Liverpool','Bristol') ... THEN
```

Comparing a value with the null value

```
value IS [ NOT ] NULL
```

value	Value that is to be tested for NULL.
IS NULL	The condition is satisfied if *value* contains the null value. Otherwise, the condition is not satisfied.
IS NOT NULL	The condition is satisfied if *value* does not contain the null value. Otherwise, the condition is not satisfied.

Example

```
SELECT dept-staff-no, seq-no, last-name
       FROM staff-member
       WHERE proj-team IS NULL

IF &var IS NULL THEN ...
```

Check whether a character string is numeric

A character string is checked as to whether it represents a numeric value.

numeric-predicate may be specified only for CYCLE, IF and CASE, but not for SQL statements.

```
char–expression IS [ NOT ] NUMERIC
```

---

char-expression	Character string to be checked.
IS NUMERIC	The condition is satisfied if the *value-function* NUMERIC is applied to the character string without specifying *[, char-literal]* (i.e. using the standard mask) and does not result in a conversion error.
IS NOT NUMERIC	The condition is satisfied if the *value-function* NUMERIC is applied to the character string without specifying *[, char-literal]* (i.e. using the standard mask) and results in a conversion error.

Examples

```
SET &c = 'ABCDEF';

  IF &c IS NOT NUMERIC
     THEN DISPLAY FORM 'Non–numeric';
  END IF;

SET &c = '1000';

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

data-group
Define data group

data-group specifies the data type "data group" for a variable. *data-group* consists of components that may have any data types. The structure of *data-group* is determined by the sequence of the components. The components themselves are also data groups if they in turn consist of different components.

For the specification of data groups, *level* can be used to assign level numbers, whose values determine the structure levels. A data group is indicated when a specification of one level is followed by a specification whose level number is higher. Conversely, a component of a data group is indicated when a specification of one level is preceded by a specification whose level number is lower. The specification with the highest level number is a simple component.

The nesting depth of data groups is 49; no more than three repeating groups can be nested in each other.

In SQL statements, *data-group* can only be specified with UDS databases.

For additional structure types, see the metavariable *structure-type*.

```
data-group::=[ REDEFINES { variable | character1 [ suffix ] } ]
             {, level { character2 | FILLER }
                      { basic-data-type [ base-type ] | structure-type } }
```

REDEFINES

REDEFINES identifies the variable being redefined (base variable).

The variable being redefined must not be longer than the base variable. If the variable being redefined is a component of a structure, the base variable must be a component of the same base structure (structure with the level number 1).

variable

Name of the base variable being redefined. It must already have been defined and must not itself been defined by LIKE or REDEFINES. If the base variable is a structure, the redefining variable must not be a component of that structure.

There must be no repeating groups between the beginning of the smallest substructure containing both the base variable and the redefining variable and the base variable or the redefining variable itself.

character1	Alphanumeric characters for the name of a component. 0 < number(*character*) < 32
suffix	See the metavariable *variable*.
level	*level* specifies the level numbers. The first level number must always be "1".
character2	Alphanumeric characters for the name of a component. 0 < number(*character*) < 32
FILLER	Can be specified in place of *character* for levels > 1. A field identified with FILLER cannot be separately accessed, but instead can be used only within *aggregate* (see the metavariable *value*).
basic-data-type	See the metavariable *basic-data-type*.
base-type	See the metavariable *base-type*.
structure-type	See the metavariable *structure-type*.

Example

Variable "a" is a data group. The components are "b", "b1" through "b31" and "c". "b1", "b2", "b31" and "c" are simple components.

```
DECLARE 1 &a,
          2 b,
            3 b1 INTEGER,
            3 b2 NUM (7,2),
            3 b3,
              4 b31 CHAR (10),
          2 c CHAR (8);
```

data-type
Define data type

data-type specifies the data type for variables and user-defined data types.

```
data-type::={ basic-data-type [ base-type ] |
              structure-type |
              ( rows, columns ) basic-data-type [ base-type ] }
```

---

basic-data-type	See metavariable *basic-data-type*
base-type	See metavariable *base-type*
structure-type	See metavariable *structure-type*
rows	*rows* defines the number of vertical components in a variable with two coordinates (matrix) (0 < *rows* < 256).
columns	*columns* defines the number of horizontal components in a variable with two coordinates (matrix) (0 < *columns* < 256).

date-time-expression
Calculate date or time

date-time-expression defines a valid date or time. The data type of *date-time-expression* one of the date-time data types (DATE, TIME, TIME(3) or TIMESTAMP(3)).

date-time-expression must not be used in SQL statements for SESAM V1.x and UDS. *date-time-expression* can only be used in SQL statements for SESAM V2.x if it contains CURRENT DATE / TIME / TIMESTAMP.

For additional expressions, see the metavariable *expression*.

```
date-time-expression::={ date-time-term1 |
                       date-time-expression1 { + | - } interval-term |
                       date-time-expression2 || date-time-term2 }
```

---

date-time-term1 — See metavariable *date-time-term*

date-time-expression1

The data type of *date-time-expression1* must be a date-time data type. *date-time-expression1* must not contain structured variables or aggregates.

\+ — Sum operator

– — Difference operator

interval-term — See metavariable *interval-term*.

interval-term must have the interval unit YEARS, MONTHS or DAYS if a date (data type DATE) is specified for *date-time-expression*.

interval-term must have the interval unit HOURS, MINUTES, SECONDS or FRACTIONS if a time (data type TIME or TIME(3)) is specified for *date-time-expression*.

date-time-expression2

date-time-expression2 must have the data type DATE, TIME or TIME(3).

If *date-time-expression2* has the data type DATE, *date-time-term2* must have the data type TIME or TIME(3).

If *date-time-expression2* has the data type TIME or TIME(3), *date-time-term2* must have the data type DATE.

If *date-time-expression2* has the data type TIME, fractions of seconds are filled with zeros (HH:MI:SS.000).

||

The concatenation operator || is used to concatenate the strings *date-time-expression2* and *date-time-term2*. The second string is appended directly to the first. The result has the data type TIMESTAMP(3).

date-time-term2

date-time-term2 must have the data type DATE, TIME or TIME(3).

If *date-time-term2* has the data type TIME or TIME(3), *date-time-expression2* must have the data type DATE.

If *date-time-term2* has the data type DATE, *date-time-expression2* must have the data type TIME or TIME(3).

If *date-time-term2* has the data type TIME, fractions of seconds are filled with zeros (HH:MI:SS.000).

Examples

The date which results when 30 days are added to the current date is to be assigned to the variable &plus30.

```
DECLARE VARIABLE &date DATE;
DECLARE VARIABLE &plus30 DATE;
...
SET &date=CURRENT DATE;
SET &plus30=&date + 30 DAYS;
```

The date which results when the number of days elapsed since the first moon landing (20.07.69) are added to the current date is to be assigned to the variable &plus.

```
DECLARE VARIABLE &date DATE;
DECLARE VARIABLE &landing DATE INIT DATE(1969-07-20);
DECLARE VARIABLE &plus DATE;
...
SET &date=CURRENT DATE;
SET &plus=&date + (&date - &landing) DAYS;
```

date-time-field
Define components of a date or time

date-time-field is used to specify the components of a date (day, month, year) or a time (hour, minute, second, fraction).

```
date-time-field::={ YEAR | MONTH | DAY | HOUR | MINUTE | SECOND | FRACTION }
```

YEAR	The unit year is specified.
MONTH	The unit month is specified.
DAY	The unit day is specified.
HOUR	The unit hour is specified.
MINUTE	The unit minute is specified.
SECOND	The unit second is specified.
FRACTION	The unit fraction is specified.

date-time-term
Define date or time

date-time-term specifies a time (date, time or timestamp) or converts a character expression into a result of the type DATE, TIME(3) or TIMESTAMP(3).

The data type of *date-time-term* is a date-time data type (see metavariable *basic-data-type*).

```
date-time-term::=
    { value |
    ( date-time-expression ) |
    { DATE | TIME | TIMESTAMP } { ( char-expression ) | ( date-time-expression1 ) } |
    CURRENT { DATE | TIME | TIMESTAMP } |
    CONCAT { date-time-expression2, date-time-expression3 } }
```

---

value — The data type of *value* must be DATE or TIME. If *value* contains the null value, *date/time-term* also contains the null value. *value* must not be a structured variable or an aggregate.

date-time-expression — The data type of *date-time-expression* must be DATE or TIME. If *date-time-expression* contains the null value, *date-time-term* also contains the null value.

DATE — *char-expression* or *date-time-expression1* is converted to a value of the type DATE.

TIME — *expression* is converted to a value of the type TIME(3).

TIMESTAMP — *expression* is converted to a value of the type TIMESTAMP(3).

char-expression — *char-expression* must evaluate to the printable form of a valid time value (see *date-time-literal* under the metavariable *literal*). If *char-expression* contains NULL, *date-time-term* also contains NULL.

char-expression must not be a structured variable or an aggregate and must contain the appropriate separators (see *date-time-literal* under the metavariable *literal*).

date-time-expression1

The data type of *date-time-expression1* must be a date-time data type.

– In the case of DATE, *date-time-expression1* must have the data type DATE or TIMESTAMP(3).

– In the case of TIME, *date-time-expression1* must have the data type TIME, TIME(3) or TIMESTAMP(3).

If *date-time-expression1* has the data type TIME, fractions of sections are filled with zeroes on conversion (HH:MI:SS.000).

If *date-time-expression1* has the data type TIMESTAMP(3), *year-month-day* are removed on conversion (YYYY-MO-DD).

– In the case of TIMESTAMP(3), *date-time-expression1* may be of the data type DATE, TIME, TIME(3) or TIMESTAMP(3).

If *date-time-expression1* is of type DATE, the time *hour*:*minute*:*seconde*.*fraction* is filled with zeroes on conversion (00:00:00.000).

If *date-time-expression1* has the data type TIME, the current date *year-month-day* (YYYY-MO-DD) is extended on conversion and the fractions of seconds are filled with zeroes (HH:MI:SS.000).

If *date-time-expression1* has the data type TIME(3), the current date *year-month-day* (YYYY-MO-DD) is extended on conversion.

If *date-time-expression1* contains NULL, *date-time-term* also contains NULL.

CURRENT DATE — Evaluates to the current date in the format *year-month-day* (see *date-time-literal* under the metavariable *literal* for details on the format).

CURRENT TIME — Evaluates to the current time in the format *hour:minute:second.fraction* (see *date-time-literal* under the metavariable *literal* for details on the format). If CURRENT TIME is specified for a field with the data type TIME, DRIVE/WINDOWS truncates the fractions of a second.

CURRENT TIMESTAMP

Evaluates to the current time in the format *year-month-day␣hour:minute:second.fraction* (see *date-time-literal* under the metavariable *literal* for details on the format).

CONCAT — The strings *date-time-expression1* and *date-time-expression2* are concatenated (see metavariable *date-time-expression*). The result has the data type TIMESTAMP(3).

date-time-expression2

date-time-expression2 must have the data type DATE, TIME or TIME(3).

If *date-time-expression2* has the data type DATE, *date-time-expression3* must have the data type TIME or TIME(3).

If *date-time-expression2* has the data type TIME or TIME(3), *date-time-expression3* must have the data type DATE.

If *date-time-expression2* has the data type TIME, fractions of seconds are filled with zeros (HH:MI:SS.000).

date-time-expression3

date-time-expression3 must have the data type DATE, TIME or TIME(3).

If *date-time-expression3* has the data type TIME or TIME(3), *date-time-expression2* must have the data type DATE.

If *date-time-expression3* has the data type DATE, *date-time-expression2* must have the data type TIME or TIME(3).

If *date-time-expression3* has the data type TIME, fractions of seconds are filled with zeros (HH:MI:SS.000).

Examples

The variable &date is assigned the current date, the variable &time is assigned the current time.

```
DECLARE VARIABLE &date DATE;
DECLARE VARIABLE &time TIME;
...
SET &date=CURRENT DATE;
SET &time=CURRENT TIME;
```

The variable &end-of-century is assigned the string "1999-12-31".

```
DECLARE VARIABLE &end-of-century DATE;
...
SET &end-of-century=DATE(1999-12-31)        or
SET &end-of-century=DATE('1999-12-31')
```

date-time-unit
Define unit for a time period

date-time-unit defines the unit for time periods (intervals) e.g. years, days or minutes.

```
date-time-unit::={ date-time-field1 TO date-time-field2 |
                   UNITS date-time-field3 |
                   YEARS |
                   MONTHS |
                   DAYS |
                   HOURS |
                   MINUTES |
                   SECONDS |
                   FRACTIONS }
```

---

date-time-field1, date-time-field2

date-time-field1 must always be identical with *date-time-field2*.

You can only specify interval units with a single component, e.g. MONTH TO MONTH.

UNITS — Specifies the units for date and time. Abbreviations are possible, e.g. YEARS for UNITS YEAR (see also the *date-time-field* metavariable).

date-time-field3 — See metavariable *date-time-field*

YEARS — Specifies the unit "years".

MONTHS — Specifies the unit "months".

DAYS — Specifies the unit "days".

HOURS — Specifies the unit "hours".

MINUTES — Specifies the unit "minutes".

SECONDS — Specifies the unit "seconds".

FRACTIONS — Specifies the unit "fractions".

Example

The period of time up to the end of the millennium in days is assigned to the variable &period.

```
DECLARE VARIABLE &date DATE;
DECLARE VARIABLE &mill-end DATE INIT DATE(2000-01-01);
DECLARE VARIABLE &period INTERVAL DAYS;
...
SET &date=CURRENT DATE;
SET &period=&mill-end - &date;
```

expression
Expressions

expression describes numeric, character, date-time and interval expressions.

```
expression::={ char-expression | date-time-expression |
               interval-expression | value-expression }
```

char-expression — See metavariable *char-expression*:

```
char-expression::={ char-prim | char-expression || char-prim }
```

date-time-expression — See metavariable *date-time-expression*:

```
date-time-expression::=
 { date-time-term | date-time-expression { + | - } interval-term
```

interval-expression — See metavariable *interval-expression*:

```
interval-expression::=
 { interval-term | ( date-time-expression - date-time-term ) }
```

value-expression

See metavariable *value-expression*:

```
value-expression::=
 value-term1 [ [ * | / | % | ** ] [ + | - ] value-term2 ]
```

format
Define format for screen or list

format specifies the format for screens and lists.

```
format::={ FREE |
          { TABLE | LINE | SEQUENCE } [ NAMES [ VALUES ] | VALUES ] } }
```

FREE — Default

FREE, or no format specification, results in unformatted output: The data contents of the elements to be output are output in a continuous stream, without regard for the end of a line. The names of the data contents are not output.

Example

```
STEEL          MAX            MAIN RD       80469BALTI
MORE
```

– Output occurs sequentially, without spaces between the individual fields. However, it may be controlled as desired with the aid of NEWLINE, NEWPAGE, TABULATOR or BLANK.

– Exceeding the end of a line automatically causes a line break.

TABLE — The elements to be output are output in table form.

The width of the columns in the table is determined by the longest element name or the longest data value. Neither the element name nor the data contents are truncated. The columns in the table are separated from each other by a space.

TABLE is only permitted if the complete output format fits in a line. Otherwise the procedure is aborted with the error message `DRI0049 MAXIMUM LINE LENGTH ((&00)) EXCEEDED`.

In program mode, TABLE means that a title and data line will always be printed. This does not apply to matrices.

In interactive mode, the lines which follow the title contain the data values of the elements only.

If you specify TABLE for *format*, you may not subsequently specify NEWLINE and NEWPAGE.

NAMES VALUES

NAMES VALUES is the default setting for the TABLE operand.

The names are output in the first line. The following lines contain the data contents of the elements which are to be output.

Example

```
NAME          FIRST-NAME     STREET
STEEL         MAX            MAIN RD
```

– Only the names of the control elements are output. Matrices are output in matrix format and the name appears once only.
– Literals are repeated in each line.

NAMES

The TABLE NAMES specification results in a tabular format with the names being output without the data contents.

VALUES

The TABLE VALUES specification results in a tabular format with the data contents being output without the names.

LINE

The elements are output line by line.

If you specify LINE for *format*, the TABULATOR and BLANK specifications result in a line feed with a blank line.

NAMES VALUES

NAMES VALUES is the default setting for the LINE operand.

Each line contains the level number and name followed by the associated data contents.

Example

```
1 NAME: STEEL
1 FIRST-NAME: MAX
1 STREET: MAIN RD.
```

– Repeating groups are output underneath one another.
– Line overflows (depending on the value of COLUMNS) create an automatic line wrap.
– Vector elements are output side by side.
– Matrices are output in matrix format and the name appears at the start of each matrix line.
– A blank is output between any two fields.

NAMES

If you do not specify VALUES then the output format is vertical and names are output without the data contents.

VALUES

LINE VALUES creates a vertical format and the data contents are output without the names.

SEQUENCE

The elements are output sequentially.

NAMES VALUES

NAMES VALUES is the default setting for the SEQUENCE operand.

The elements for output each consist of a name and the associated data contents. The elements for output are separated from one another by a blank.

Example

```
NAME: STEEL          ␣FIRST-NAME: MAX          ␣STREET: MAI
N RD.       ␣
```

- Repeating groups are output side by side
- Line overflows (depending on the value of COLUMNS) create an automatic line wrap.
- Vector elements are output side by side.
- Matrix elements are output side by side and the name appears at the start of each matrix line.
- Only the names of the control elements are output.
- A blank is output between any two fields.

NAMES

If you do not specify VALUES then the output format is horizontal and the names are displayed without the data contents.

VALUES

SEQUENCE VALUES creates a horizontal format and the data contents are output without the names.

interval-expression
Calculate a time period

interval-expression specifies a valid date or time interval.

The data type of *interval-expression* is INTERVAL (see the metavariable *basic-data-type*).

interval-expression must not be specified in SQL statements for SESAM and UDS.

```
interval-expression::={ interval-term |
                       ( date-time-expression - date-time-term ) |
                       interval-expression { + | - } interval-term }
```

---

interval-term — See the metavariable *interval-term*.

date/time-expression, date/time-term

date/time-expression and *date/time-term* must both contain date or time specifications.

date/time-expression and *date/time-term* must contain neither structured variables nor aggregates.

The result has the following data type:

Data type	**for difference between:**
INTERVAL DAYS	DATE - DATE DATE - TIMESTAMP(3) TIMESTAMP(3) - DATE
INTERVAL SECONDS	TIME - TIME
INTERVAL FRACTIONS	TIME - TIME(3) TIME - TIMESTAMP(3) TIME(3) - TIME TIME(3) - TIME(3) TIME(3) - TIMESTAMP(3) TIMESTAMP(3) - TIME TIMESTAMP(3) - TIME(3) TIMESTAMP(3) - TIMESTAMP(3)

interval-expression	If *interval-expression* is of the data type INTERVAL with either YEARS or MONTHS as units, then *interval-term* must be of the data type INTERVAL with either YEARS or MONTHS as units. The result always has MONTHS as units. If *interval-expression* is of the data type INTERVAL with DAYS, HOURS, MINUTES, SECONDS or FRACTIONS as units, then *interval-term* must be of the data type INTERVAL with DAYS, HOURS, MINUTES, SECONDS or FRACTIONS as units. The result has SECONDS as units if neither *interval-expression* nor *interval-term* has FRACTIONS as units. In all other cases, the result has FRACTIONS as units.
interval-term	If *interval-term* is of the data type INTERVAL with either YEARS or MONTHS as units, then *interval-expression* must be of the data type INTERVAL with either YEARS or MONTHS as units. If *interval-term* is of the data type INTERVAL with DAYS, HOURS, MINUTES, SECONDS or FRACTIONS as units, then *interval-expression* must be of the data type INTERVAL with DAYS, HOURS, MINUTES, SECONDS or FRACTIONS as units. The result is of the data type INTERVAL with the units described in *interval-expression*.
+	Sum operator
-	Difference operator

When calculating intervals as the difference between two times, DRIVE/WINDOWS does not calculate beyond midnight but instead outputs a negative result (see example).

Examples

The number of days until the end of the millennium is assigned to the variable &time-period.

```
DECLARE VARIABLE &date DATE;
DECLARE VARIABLE &mill-end DATE INIT DATE(2000-01-01);
DECLARE VARIABLE &time-period INTERVAL DAYS;
...
SET &date=CURRENT DATE;
SET &time-period=&mill-end - &date;
```

The number of seconds elapsed in the current day is assigned to the variable &period.

```
DECLARE VARIABLE &time TIME;
DECLARE VARIABLE &start TIME INIT TIME(00:00:00);
DECLARE VARIABLE &period INTERVAL SECONDS;
...
SET &time=CURRENT TIME;
SET &period=&time - &start;
```

The variable &period is assigned the value -3 (hours). Calculations do not extend beyond midnight (09:00:00 - 12:00:00 = 21:00:00).

```
DECLARE VARIABLE &period INTERVAL HOURS;
...
SET &period=TIME(09:00:00) - TIME(12:00:00);
```

interval-term
Define a time period

interval-term defines a valid time period (interval). The data type is INTERVAL and uses the interval unit defined by UNITS or by the variables used.

interval-term must not be specified in SQL statements for SESAM or UDS.

```
interval-term::={ value-term { UNITS date-time-field | YEARS | MONTHS | DAYS |
                             HOURS | MINUTES | SECONDS | FRACTIONS } |

                  ( interval-expression ) |
                  interval-term { * | / } value-term }
```

---

value-term — See the metavariable *value-term*.

date/time-field — See the metavariable *date/time-field*.

interval-expression — *interval-expression* is treated as being numeric with a precision of 15 and a scale factor of 0 (see the metavariable *interval-expression*).

interval-term — Interval as a factor or dividend

The result will be of the following data type depending on the unit used in the INTERVAL data type:

Data type of *interval-term*	**Data type of result**
INTERVAL YEARS	INTERVAL MONTHS
INTERVAL MONTHS	INTERVAL MONTHS
INTERVAL HOURS	INTERVAL SECONDS
INTERVAL MINUTES	INTERVAL SECONDS
INTERVAL SECONDS	INTERVAL SECONDS
INTERVAL FRACTIONS	INTERVAL FRACTIONS

* — Multiplication operator

/ — Division operator

Example

The value of 30 days is assigned to the variable &month for the calculation of interest.

```
DECLARE VARIABLE &month INTERVAL DAYS;
...
SET &month=30 DAYS;
```

The keyword "DAYS" can be omitted from the SET statement, since the interval unit for the variable &month has been defined in the DECLARE statement:

```
...
SET &month=30;
```

literal
Define a literal

literal contains a string with a constant value. Empty literals may also be assigned (e.g. `SET &vc = '';`).

```
literal::={ char-literal | numeric-literal  | date-time-literal | interval-literal |
            hex-literal }
```

---

char-literal

```
char-literal::='string' [ (n) ]

string::=[  character ] ...
```

string
: String with alphanumeric data type.
string must be enclosed in single quotes ('). If a single quote is used in *string*, it must be doubled. The doubled single quotes are treated as a single character.
string may be void and may consist of a maximum of 256 characters.

n
: Repetition factor ($1 \leq n \leq 256$).

character
: EBCDIC character

numeric-literal

```
numeric-literal::=
{ [ + | - ] integer [ { . | , } integer ] |
  [ + | - ] integer [ { . | , } integer ] E [ + | - ] integer }
```

numeric-literal contains a fixed-point value, whose data type is numeric (max. 32 digits). The precision of a fixed-point value refers to the number of digits. The scale factor of a fixed-point value refers to the number of decimal places (max. 31).

integer
: Only digits may be used for *integer*.

E
: exponential representation to base 10
Any characters preceding or following *E* must be specified without blanks (␣).

date-time-literal

```
date-time-literal::=
{ DATE (year-month-day) |
  TIME (hour:minute:second [.fraction ] ) |
  TIMESTAMP (year-month-day hour:minute:second.fraction ) }
```

date-time-literal contains a valid date (data type: DATE), a valid time (data type: TIME or TIME(3)) or a valid timestamp (data type: TIMESTAMP(3)) .

year
: Four-digit integer 0000 through 9999, specifying the year

month
: Two-digit integer 1 through 12, specifying the month

day
: Two-digit integer 1 through 31 (depending on the month), specifying the day

hour
: Two-digit integer 00 through 23, specifying the hour

minute
: Two-digit integer 00 through 59, specifying the minute

second
: Two-digit integer 00 through 59, specifying the second

fraction
: Three-digit integer 000 through 999, specifying the fractions of a second (1/1000 second)

The separators between the components must be strictly observed:
Hyphen (-) between year, month and day
Blank (␣) between day and hour
Colon (:) between hour, minute and second
Period (.) between second and fraction of a second

interval-literal

```
interval-literal::=INTERVAL ({ + | - } integer) date-time-unit
```

integer
: Only digits (max. 32) are permitted for *integer*.

date-time-unit
: See metavariable *date-time-unit*

hex-literal

```
hex-literal::=X'string' [ (n) ]
```

string
: Hexadecimal string (max 512 characters). *string* may only contain the digits "0" to "9" and the characters "A" to "F".

n
: Repetition factor ($1 \leq n \leq 256$)

mask
Define a MASK clause

Definition of the representation options for entry and output fields.

```
mask::=MASK char-literal
```

MASK — MASK defines how data values are represented in entry and output fields.

char-literal — See the metavariable *literal*.

A mask has two principal parts, that can be specified in *char-literal*: mask control characters and user text.

Mask control characters control how the data value to be output is represented in an output field. There are control characters for numeric, character-string and date/time variables.

User texts are text parts that may be freely defined by you. They may be specified at any desired positions within a mask.
They must be enclosed in doubled single quotes (' '), which, however, do not appear in the output.
The user texts are ignored when data is entered.

Mask control characters and user text may be defined in any order within a mask (except, the order of some mask control characters for numeric data is fixed). There are, however, fixed sequences of control characters that must not be interrupted by user text.

The contents of a mask must be specified such that the result edited for input or output does not exceed 256 characters. The mask itself may be up to 256 characters in length.

Mask control characters for numeric data types:

Mask control character	Output
X	Any character from the EBCDIC character set. The position is always output.
X(n)	Method for specifying the mask control character X n times in a row (0 < n).

The value to be entered or output must match the mask. Otherwise, spaces on the right are removed or added.

A value entered must be able to be stored in the data field.

Mask control characters for numeric data types:

Mask control character	Output
9	Character position for a digit; is always output.
Z	Leading numeric digit position. If this position contains a leading zero, a blank is output.
*	Check security symbol: leading numeric digit position. If this position contains a leading zero, "*" is output.
P	Character position for a decimal point. A "." or a "," is output, depending on the global setting. "P" may only occur once as a control character.
+	Character position for the plus sign "+" or minus sign "-". The sign is always output, depending on the data value in the field.
–	Character position for a negative sign. If the data value is negative, a minis sign is output and if the data value is positive, a blank is output.

Mask control character	Output
S	Leading numeric digit position. This position can contain the sign of a data value if the first control character of a mask is a "+" or "-" and if this position contains a leading zero. If "S" is specified, it must always occur before the control characters "Z" or "*". If the control character of the mask is not "+" or "-", "S" has the same effect as "Z" or "*"
E	Control character for floating point representation. It is followed by the length specification, which controls the number of decimal places to be output. The following applies to a float mask: En (8 < n < 23) I.e. the number of decimal places to be output must be between 1 and 14. "E" may only occur once as a control character.
BWZ	Control character for an entire field. If the data value is 0 or an empty string, the entire field is filled with blanks when output to screen. A field filled with blanks during input combined with the control character "BWZ" leads to a data value of 0. "BWZ" must be specified at the end of the mask and be separated from the rest of the mask by at least one blank.

Mask control character	**Output**
, (comma) . (period) B (= insertion control character)	Insertion control characters are character positions at which a comma (,), a period (.) or a blank (B) are inserted. Insertion control characters embedded in the sequence of "Z", "*" or "S" control characters or following immediately to the right of this sequence are a part of this sequence. If this sequence of control characters results in the output of blanks or "*" characters through suppression of leading zeros, blanks or "*" characters are likewise output at the positions of the embedded insertion control characters or at the positions of those appearing immediately to the right. By analogy, in insertion control characters appearing within or to the right of sequences of "S" control characters, the sign floats. Insertion control characters may only be specified to the left of "P".
9(n) Z(n) *(n) S(n)	Alternative method of specifying the various control characters n times in succession (0 < n).

The value to be input must fit the mask in accordance with the conversion rules. Otherwise, decimal places can be truncated or empty positions filled.

Mask control characters for date-time data types:

Mask control character	**Output**
YYYY	4-digit year specification
ZZZY	4-digit year specification; leading zeros are output as blanks
MO	2-digit month value
ZO	2-digit month value; a leading zero is output as a blank
DD	2-digit day value

Mask control character	Output
ZD	2-digit day value; a leading zero is output as a blank
HH	2-digit hour value
ZH	2-digit hour value; a leading zero is output as a blank
MI	2-digit minute value
ZI	2-digit minute value; a leading zero is output as a blank
SS	2-digit second value
ZS	2-digit second value; a leading zero is output as a blank
FFF	3-digit fraction value
ZZF	3-digit fraction value; leading zeros are output as blanks
WW	2-digit week value
ZW	2-digit week value; a leading zero is output as a blank
JJJ	3-digit, Julian day specification (= day in year)
ZZJ	3-digit, Julian day specification; leading zeros are output as blanks
Q...Q	Printable name of day
Q(n)	Printable name of day with n positions (0 < n)
R...R	Printable name of month
R(n)	Printable name of month with n positions (0 < n)
AP	Specification for a.m. (ante meridiem = morning) or for p.m. (post meridiem = afternoon)

The input value must be able to be assigned clearly to a date-time data type (DATE, TIME, TIME(3) or TIMESTAMP(3)). Otherwise the control characters can be mixed with user text as required.

Mask control characters for the data type INTERVAL:

Mask control character	Output
9	Character position for a digit; is always output.
Z	Leading numeric digit position. If this position contains a leading zero, a blank is output.
+	Character position for the plus sign "+" or minus sign "-". The sign is always output, depending on the data value in the field.
–	Character position for a negative sign. If the data value is negative, a minis sign is output and if the data value is positive, a blank is output.
S	Leading numeric digit position. This position can contain the sign of a data value if the first control character of a mask is a "+" or "-" and if this position contains a leading zero. If "S" is specified, it must always occur before the control characters "Z" or "*". If the control character of the mask is not "+" or "-", "S" has the same effect as "Z" or "*".
BWZ	Control character for an entire field. If the data value is 0 or an empty string, the entire field is filled with blanks when output to screen. A field filled with blanks during input combined with the control character "BWZ" leads to a data value of 0. "BWZ" must be specified at the end of the mask and be separated from the rest of the mask by at least one blank.
9(n) Z(n) *(n) S(n)	Alternative method of specifying the various control characters n times in succession (0 < n).

The value which is to be input or output must be appropriate for the mask in accordance with the conversion rules as unoccupied positions may otherwise be filled.

No blanks may appear between the mask control characters and a single quote.

Rules for numeric data fields

– "+" and "−" are mutually exclusive. Thus, only one of these characters may occur within a mask, and it must then be the first character in the mask. If neither of these characters is specified, no sign is output. The sign is not replaced by a blank.

– "Z" and "*" may not occur together within a mask. No "Z" or "*" may occur to the right of the decimal point control character "P".
Exception: All character positions in the mask except for "P" and insertion control characters are "Z" or "*".

– A mask for exponential representation may contain only "E" with a length specification.

Rules for DATE/TIME

– The various control characters may occur no more than once within a mask, and must stand immediately next to each other. Thus, in the case of YYYY, control characters, the four "Y"s may not be separated by user text.

– Simultaneous specification of the control characters for numeric and textual output of the day or month value is permitted.

– If too few Q (or R) control characters are specified for data input and output, the internal assignment of the day or month value to the textual input is ambiguous. In such cases the following rules apply:

  If DD/ZD (or M0/Z0) control characters are simultaneously present in the mask, the day (or month) value is given by the input at these positions.

  If no DD/ZD (or M0/Z0) control characters are specified, the program is aborted. A message is output indicating that the data input is ambiguous.

– The control characters specified in the mask must correspond to the date and time specifications for the relevant variables. When entering data, you must ensure that data entered in a mask can be unambiguously assigned to the relevant DATE, TIME or TIMESTAMP variable.

Examples

Data value	Definition for representation	Output
12.60	NUM(6,2) MASK 'ZZZZP99'	␣␣12.60
12.60	NUM(6,2) MASK '+SZZZP99'	␣+␣12.60
12.60	NUM(6,2) MASK '****P99'	**12.60
12.60	NUM(6,2) MASK 'ZZZZP99''␣DM'''	␣␣12.60␣DM
-12.60	NUM(6,2) MASK '-ZZZZP99'	-␣␣12.60
12.60	NUM(6,2) MASK '-ZZZZP99'	␣␣␣12.60
12.60	NUM(6,2) MASK '99'	12
12.60	NUM(6,2) MASK '9999'	0012
1260	NUM(6,2) MASK 'ZZ,ZZZPZZ'	␣1,260.00
-1260	NUM(6,2) MASK 'ZZ,ZZZPZZ'	␣1,260.00
1260	NUM(6,2) MASK '-ZZ,ZZZPZZ'	␣␣1,260.00
126000	NUM(8,2) MASK 'ZZZBZZZPZZ'	126␣000.00
12.60	NUM(8,2) MASK 'ZZZBZZZPZZ'	␣␣␣␣␣12.60
12.60	NUM(6,2) MASK 'ZZZZP99 BWZ'	␣␣12.60
0	NUM(6,2) MASK 'ZZZZP99 BWZ'	␣␣␣␣␣␣␣
12.60	NUM(6,2) MASK 'E9'	␣1.2E+001
12.60	NUM(6,2) MASK 'E10'	␣1.26E+001
12.60	NUM(6,2) MASK 'E16'	␣1.26000000E+001
1260	INT MASK 'E9'	␣1.2E+003
12	INT MASK 'ZZP99'	12.00
1260	INT MASK 'ZZP99'	Error message
1260	REAL MASK 'ZZ99'	1260
12.60	REAL MASK 'ZZ99'	␣␣13
1260	REAL MASK 'E9'	␣1.2E+003
1.2601260126	FLOAT MASK 'E16'	␣1.26012601E+000
DER␣PATE	CHAR (8) MASK '''THE␣BOOK:␣''XXXXXXXX'	THE␣BOOK:␣THE␣GODFATHER
DER␣PATE	CHAR (8) MASK '''THE␣BOOK:␣''X(8)'	THE␣BOOK:␣THE␣GODFATHER
1789-07-14	DATE MASK 'DD''.''MO''.''YYYY'	14.07.1789
0622-06-15	DATE MASK 'ZD''.''ZO''.''ZZZY'	15.␣6.␣622
1982-08-13	DATE MASK 'Q(10)'',␣THE␣''ZD''TH'''	FRIDAY␣␣␣␣,␣THE␣13TH
20:15:26	TIME MASK 'HH'':''MI'␣HOURS	20:15␣HOURS
16:09:56.127	TIME(3) MASK 'HH'':''ZI'␣HOURS	16:␣9␣HOURS

null-value
Define the representation of the null value

null-value specifies the representation of the null value.

```
null-value::=[ CHARTYPE=char-literal1 ] [ NUMTYPE=char-literal2 ]
```

CHARTYPE=char-literal1

Specifies the representation of the null value for the data types CHARACTER, DATE, TIME, TIME(3) and TIMESTAMP(3) (see metavariable *basic-data-type*) (max. 1 character).

CHARTYPE may be specified only once.

NUMTYPE=char-literal2

Specifies the representation of the null value for the data types NUMERIC, DECIMAL, INTEGER, SMALLINT and INTERVAL (max. 1 character).

The following characters are permitted:
Digits 0 through 9, comma (,), period (.) and the special characters * + –

NUMTYPE may be specified only once.

The default setting for null value representation for printer output is the period (.).
The default setting for null value representation for screen input/output is the character X'00'.

programming
Define statements for the body of a program

programming specifies a DRIVE statement for the body of a program.

```
programming::={ statement; |
                CASE; END CASE; |
                CYCLE; END CYCLE; |
                DISPATCH; END DISPATCH; |
                IF; END IF; } ...
```

---

statement	Linear program statement for the body of a program (see DRIVE Programming Language manual [2]).
CASE	Programs a conditional branch (see the CASE statement).
CYCLE	Programs a loop (see the CYCLE statement).
DISPATCH	Programs a dispatch block (see the DISPATCH statement).
IF	Programs a condition (see the IF-statement).

repeating-group
Define repeating group

repeating-group specifies the data type "repeating group" for a variable. *repeating-group* consists of a fixed number of components, all having the same data type. The number of components is specified by the repetition factor *n*.

repeating-group may only be specified in SQL statements for UDS databases.

For additional structure types, see the metavariable *structure-type*.

```
repeating-group::=( n ) data-group
```

---

n	Repetition factor (0 < *n* < 31000).
data-group	See the metavariable *data-group*.

set-function
Specify set functions

A set function calculates a value from a set of values.

```
set-function::={ SUM | AVG | MAX | MIN } ( [ ALL ] expression )
```

SUM	Calculates the sum of the values in a set
AVG	Calculates the average value in a set of values
MAX	Determines the maximum value in a set
MIN	Determines the minimum value in a set
ALL	All values are considered including duplicated values. In the case of MAX and MIN, ALL is syntactically permissible, but has no meaning.
expression	Argument to which the set function is applied. *expression* must be a non-structured column. In the case of AVG and SUM, *expression* must be numeric. (See the metavariable *expression*)

The result of the set function has the following data type:

Set function	**Data type of the result**
MIN and MAX	The same data type as *expression*
SUM	DECIMAL with a precision of 15. The number of decimal places corresponds to the number of decimal places in the *expression* specified.
AVG	DECIMAL with a precision of 15. The number of decimal places corresponds to the number of decimal places in the *expression* specified.

SUM - Calculate sum

SUM calculates the sum of the values in a set.

```
SUM ( [ ALL ] expression )
```

ALL	Default All values are considered including duplicated values.
expression	Expression which evaluates to a numeric value.

AVG - Arithmetic mean

AVG calculates the arithmetic mean of a set of values.

```
AVG ( [ ALL ] expression )
```

ALL	Default All values are used, including duplicates.
expression	Expression which evaluates to a numeric value.

MAX - Determine maximum

MAX determines the maximum value in a set.

```
MAX ( [ ALL ] expression )
```

ALL	ALL is syntactically permissible, but has no meaning.
expression	Expression which evaluates to a numeric or character value.

MIN - Determine minimum

MIN determines the minimum value in a set.

```
MIN ( [ ALL ] expression )
```

ALL	ALL is syntactically permissible, but has no meaning.
expression	Expression which evaluates to a numeric or character value.

structure-type
Define a structured variable

There are four kinds of structured data types:

– vectors (multiple fields)
– data groups
– repeating groups
– redefined variables

Two structured data types are considered to be equal

– if their component compositions are equal, and
– if the corresponding simple components in each are of the same data type.

For additional data definitions, see the metavariable *data-type*.

Do not use the REDEFINES clause in programs which you wish to use on different computers, as it is machine-dependent. This could lead to different results on different machines.

```
structure-type::={ vector |
                 data-group |
                 repeating-group |
                 user-type |
                 [ ( n ) ] [ REDEFINES { variable1 | character [ suffix ] } ]
                                                          LIKE variable2 }
```

---

vector	See the metavariable *vector*.
data-group	see the metavariable *data-group*.
repeating-group	see the metavariable *repeating-group*.
user-type	This data type must have been defined in the DRIVE program with DECLARE TYPE.
n	Repetition factor (0 < *n* < 256).

REDEFINES	REDEFINES identifies the variable to be redefined (base variable). The redefining variable must not be longer than the base variable. If the redefining variable is a component of a structure, the base variable must be a component of the same main structure (structure with the level number 1). REDEFINES must not be specified in INIT and USING clauses.
variable1	Name of the base variable to be redefined. It must already have been defined and must not itself be a variable defined with LIKE or REDEFINES. If the base variable is a structure, the redefining variable must not be a component of this structure. There must be no repeating groups between the beginning of the smallest structure element containing the base variable and the redefining variable and the base variable or the redefining variable itself.
character	Alphanumeric character string for the name of a component. 0 < number(*character*) < 32000
suffix	See the metavariable *variable*.
LIKE	LIKE is used to copy the structure of an already defined data group or repeating group to a variable, component by component. The level numbers are adjusted during copying. The specifications for *n* and the REDEFINES clause are not taken over. These variables may only be used in the program body. If the structure of a variable is determined with the LIKE clause, only components of the same or a lower level number are permitted as additional components for that variable. LIKE specifications must not be nested.
variable2	Name of the variable whose structure is to be copied. *variable2* must already have been defined as a data group or repeating group. *variable2* may be qualified, but not indexed. It must not be a higher-level structure with regard to the variable to be defined.

Examples

REDEFINES within a structure

```
DECLARE VARIABLE 1 &alpha,
                   2 a1,
                     3 a11 CHAR (10),
                     3 a12 CHAR (10),
                   2 a2 INT REDEFINES &a1,
                   2 a3 CHAR (2);
```

&a2 reassigns the area assigned to &a1, i.e. the area assigned to &a11 and &a12. &a2 can, however, only redefine an area insofar as its size allows. In this case, It redefines the area up to the first four characters of &a11.

LIKE applied to another group (= data group or repeating group)

The structure of the data group &v1 is transferred to the variable &v2. &v2 is identical to &v1.

```
DECLARE VARIABLE 1 &v1,
                   2 v11 CHAR,
                   2 v12 INT,
                   &v2 LIKE &v1;  →  1 &v2,
                                       2 v11 CHAR,
                                       2 v12 INT;
```

LIKE within a group

The component "v32" is defined exactly as "v31".

```
DECLARE VARIABLE 1 &v3,                    1 &v3,
                   2 v31,                    2 v31,
                     3 v311 CHAR,              3 v311 CHAR,
                     3 v312 NUM (4,2),         3 v312 NUM (4,2),
                   2 v32 LIKE &v31;  →     2 v32,
                                             3 v311 CHAR,
                                             3 v312 NUM (4,2);
```

value
Define a data value

value specifies the value of a variable or a component of a variable.

Since the NULL value is not permitted in every situation in which *value* is permitted it does not form part of the description of the metavariable *value*.

```
value::={ char-prim | variable | literal | aggregate | VALUE | $PI }
```

---

char-prim

See the metavariable *char-prim*.

variable

variable must not contain the null value if it is referenced by SQL statements accessing UDS databases (see the metavariable *variable*).

literal

See the metavariable *literal*.

aggregate

```
aggregate::= < { value | NULL } >, ...
```

aggregate specifies an aggregate, a structured value whose components are specified by *value*. *aggregate* must not contain more than 255 *values*. Structured variables must not be specified for *value*.

value

Literals, variables and $PI can be specified for *value*. In the case of structured variables, it is only possible to reference the lowest structure and not the entire structure.

NULL

The component of *aggregate* is assigned the null value. The null value can be assigned to any variable and any column of a table (without a non-NULL condition) (see the SET statement in the DRIVE Directory [3] and INSERT and UPDATE in the SQL directories [4-6].

Example

```
address=<'Waldweg 4',80462,&city>
```

VALUE	VALUE may be specified only for *check*. When *check* is used, the variable which the check condition refers to can be referenced, i.e. the name of the variable need not be repeated.
$PI	$PI stands for the number 3.141592653...

value-expression
Define a numeric expression

value-expression defines numeric expressions.

The basic arithmetic operations are performed with a precision of up to 32 places (data type XDEC).

In old style operation, basic arithmetic operations are performed to an accuracy of up to 15 places (NUMERIC data type).

In programs which have been compiled using the DRIVE compiler DRIVE/WINDOWS-COMP, the basic arithmetic operations are performed using fixed point or decimal arithmetic. Provided that addition, subtraction and multiplication operations only involve values of type INTEGER or SMALLINT, fixed point arithmetic is used. In other cases, decimal arithmetic is used. (See DRIVE Compiler [16])

For additional expressions, see the metavariable *expression*.

```
value-expression::=value-term1 [ [ * | / | % | ** ] [ + | - ] value-term2 ]
```

---

value-term1, value-term2

The first character of *value-term* (see metavariable *value-term*) must not be "+" or "–".

The data type of *value-term* must be numeric, otherwise DRIVE/WINDOWS issues an error message.

In the case of addition, subtraction, multiplication, division and the calculation of percentages, *value-expression* is assigned the data type DECIMAL. In the case of the exponential operator, the data type is FLOAT.

The precision P and the scale factor S of *value-expression* depend on the precision of P1 and P2, the scale factors S1 and S2 and on the way in which the *value-terms* are related to each other.

Arithmetic operation	**P and S for *value-expression***
`value-term1 * value-term2`	`P=MIN(15,P1+P2)` `S=MIN(15,S1+S2)`

Arithmetic operation	P and S for *value-expression*
value-term1 / value-term2	P=15 S=MAX(15-P1+S1-S2,0)
value-term1 { + | - } value-term2	P=MIN(15,MAX(P1-S1,P2-S2) + MAX(S1,S2) + 1) S=MAX(S1,S2)

The following precedence rules apply when the operators are evaluated:

Parentheses
- before sign operators
- before exponentiation
- before multiplication, division, percentage calculation
- before addition, subtraction.

Equivalent operators are processed from left to right.

If *value-term* has the null value, *value-expression* also has the null value.

* Multiplication operator.

/ Division operator.

% Percentage operator.

The scale factor of *value-expression* is equal to the sum of the scale factors of value-term1 and value-term2.

"%" must not be used in SQL statements.

** Exponential operator.

The scale factor of *value-expression* is 6.

The exponent (value-term2) must be an integer. This is checked at program execution.
If value-term1 = 0, then value-term2 > 0.

"**" must not be used in SQL statements.

\+ Sign operator or sum operator.

\- Sign operator or difference operator.

Used as a sign operator, "–" reverses the sign of *value-term*.

value-function
Value function

value-function calculates a value by applying a value function to exactly one argument.

value-function must not be specified in SQL statements for a database.

```
value-function::=
    { { SIN | COS | TAN | LN | LG | ABS | EXP | SQR | SQRT } ( value-expression ) |
      CHARLENGTH ( char-expression1 ) |
      ROUND ( value-expression1 [, s1 ] ) |
      TRUNC ( value-expression2 [, s2 ] ) |
      NUMERIC ( char-expression2 [, char-literal ] ) |
      POSITION ( char-expression3 IN char-expression4 [, n2 ] ) |
      LENGTH ( char-expression5 ) |
      MODULO ( value-expression3, value-expression4 ) }
```

SIN	The sine function is applied.
COS	The cosine function is applied.
TAN	The tangent function is applied.
LN	The natural logarithm is applied.
LG	The base 10 logarithm is applied.
ABS	The absolute value is applied.
EXP	The exponential function is applied.
SQR	The square function is applied.
SQRT	The square root function is applied.
value-expression	Argument to which the value function is to be applied. The trigonometric functions apply to arguments in radians. *value-expression* must not be an aggregate or a structured variable.
CHARLENGTH	CHARLENGTH returns the length of the string *char-expression1*. The position of the last alphanumeric character of *char-expression1* is returned. This character can also be a blank. If *char-expression1* contains the null value, CHARLENGTH returns the null value.

char-expression1 — The data type of *char-expression* must be CHARACTER.

ROUND — ROUND rounds numeric values up/down to the specified number of decimal places.
The rounding factor is: $5 * 10^{-s1-1}$

If one of the arguments contains the null value, ROUND returns the null value.

value-expression1 — See *value-expression*.

s1 — *s1* specifies the number of decimal places to which *value-expression1* is to be rounded. *s1* must be an integer.

If *s1* is not specified, the first decimal place is rounded. *value-expression1* becomes an integer.

If *s1* is positive, the decimal place *s1*+1 is rounded. *value-expression1* receives *s1* decimal places.

If *s1* is negative, the integer digit -(*s1*) is rounded.

The following limits apply: $-126 \leq s1 \leq 128$

If *value-function* consists of *literal* or *variable*, the following applies: *s1* ≤ decimal places of *value-expression1* or -(*s1*) ≤ integer digits of *value-expression1*

Example

```
ROUND (3469.87126, 0)   = 3470
ROUND (3469.87126, 4)   = 3469,8713
ROUND (3469.87126, –3)  = 3000
```

TRUNC — Truncates numeric values at a specified place.

In the case of a decimal place, all values following the position are truncated.

In the case of an integer digit, this position and all integer digits which follow it are assigned the value 0.

If one of the arguments contains the null value, TRUNC returns the null value.

value-expression2 — See *value-expression*.

s2 — *s2* specifies the position after which *value-expression2* is truncated. *s2* must be an integer.

If *s2* is not specified, the integral part of *value-expression2* is returned.

If *s2* is positive, *value-expression2* is truncated after decimal position *s2*.

If *s2* is negative, all positions after the integer digit -(*s2*) in *value-expression2* are assigned the value 0.

The following limits apply: -126 ≤ *s2* ≤ 128

If *value-function* consists of *literal* or *variable*, the following applies: *s2* ≤ decimal places of *value-expression1* or -(*s2*) ≤ integer digits of *value-expression1*

Example

```
TRUNC (3469.87126, 0)   = 3469
TRUNC (3469.87126, 4)   = 3469,8712
TRUNC (3469.87126, -3)  = 3000
```

NUMERIC

The argument *char-expression2* is converted into a result of type NUMERIC.

If one of the arguments contains the null value, NUMERIC returns the null value.

char-expression2

char-expression2 must be able to be described by *char-literal* (see metavariable *literal*).

If *char-expression2* represents a numerical value, *char-literal1* must contain numeric control characters. This can be determined at runtime using the predicate IS NUMERIC (see metavariable *condition*).

char-literal

char-literal must satisfy the conditions of *mask* (see metavariable *mask*). The contents comprise mask control characters which must be compatible with the data type.

If *char-literal* is not specified, a standard mask is used (see metavariable *mask*).

POSITION

Returns the position of strings within strings.

If one of the parameters has the null value, the result is the null value.

If the repetition factor *n2* is specified, POSITION determines the position at which *char-expression3* occurs for the *n2*-th time in *char-expression4*. (*char-expression3* * *n2*) must be ≤ *char-expression4*.

If *n2* is not specified, POSITION returns the value of the first position at which *char-expression3* occurs in *char-expression4*. *char-expression3* must be ≤ *char-expression4*.

If *char-expression3* is not contained in *char-expression4* or is without the specified repetition factor, POSITION returns the value "0".

char-expression3, char-expression4
: The data type of *char-expression* must be CHARACTER.

n2
: *n2* specifies the repetition factor. *n2* must be a positive integer.

LENGTH
: LENGTH returns the position in a string *char-expression5* which is followed by blanks only. There is no blank at the position returned.

  If *char-expression5* only contains blanks, LENGTH returns the value 0.

  If *char-expression5* contains the null value, LENGTH returns the null value.

char-expression5
: The data type of *char-expression5* must be alphanumeric.

MODULO
: MODULO returns the remainder value from the division of *value-expression3* by *value-expression4*.

  If *value-expression3* and/or *value-expression4* are decimal numbers, the following applies:
  The number of decimal places in the remainder value is equal to the number of decimal places in *value-expression3* or *value-expression4*, whichever is the larger.

  The relationship between MODULO and TRUNC is shown in the following equation:
  MODULO(a,b) = a - (TRUNC(a/b) * b)

  If one of the arguments contains the null value, MODULO returns the null value.

value-expression3, value-expression4
: See *value-expression*.

Rules

– The value functions SIN, COS, TAN, LN, LOG, EXP, SQR and SQRT are calculated with a precision of 15 digits, with 6 decimal places.

– The value function ABS is calculated with a precision of 15 digits, but with an unchanged number of decimal places.

value-term
Define a numeric term

```
value-term::=[ + | - ] { value |
                       column |
                       set-function |
                       value-function |
                       ( value-expression ) |
                       interval-term }
```

+	Sign operator or sum operator.
-	Sign operator or difference operator.
	Used as a sign operator, "–" reverses the sign of *value-term*. The first character before or after the monadic operators "+" and "–" must not also be "+" or "–". The data type of *value-term* must be numeric without decimal places.
value	See the metavariable *value*.
column	The columns which are specified in *value-term* must come from the same base table.
	For *column* see the SQL directories [4-6]
set-function	See the metavariable *set-function*.
value-expression	See the metavariable *value-expression*.
value-expression	See metavariable *value-expression.,*
interval-term	See the metavariable *interval-term*.
	interval-term is treated as numeric with a precision of 32 and a scale factor of 0 (see also the metavariable *interval-expression*).
	interval-term may not be specified in SQL statements for SESAM and UDS.

variable
Define a simple variable or reference a component

variable references a simple variable or a component of a structured variable. All components which occur on the next level can be specified with partially qualified names (".*"). The *variable* name can have a maximum of 32 characters.

```
variable::={ [ char-literal: ] var-name1 [ suffix ] |

             [ char-literal: ]
             { var-name2 [ ( { index1 | range1 }, { index2 | range2 } ) ] } }
```

---

char-literal

char-literal can only be specified in a report definition. It identifies the type of record to which the value is to refer.

var-name1

Name of a simple variable or the first qualifier for a component.

var-name1 must start with "&" and must not exceed 32 characters in length.

suffix

```
suffix::={ group-component | index-component }
```

A *group-component* or an *index-component* can be specified as the suffix.

```
group-component::= . { * | component [ suffix ] }
```

. Qualification operator

* abbreviation for a list of all variable components that are on the next lower level.
In SQL statements "*" must not be specified for columns.

component
Part of a column of a database (see statement DECLARE VARIABLE ... LIKE CURSOR/TABLE) or of a structured variable (= data group or repeating group).
The components of the highest structure level are known as simple components.
component is specified without the character "&" and may consist of a maximum 31 characters.
component [suffix] may not be part of a repeating group.

```
index-component::={
     { ( { index | var-name [ .component ] ... }
         [ .component [ suffix ] ) |
       ( range ) }
```

index-component
: Must be specified if a value is to be specified for a field component of a structured field with occurrences or of a multiple field. *index* or *range* must be supplied with a value when specifying a field component of a multiple field, i.e. you may not specify an index variable.

index
: Designates a vector or repeating group occurrence
The values which you may specify for *index* are as follows:
0 < *index* ≤ *repetitions*, where *repetitions* is the specification in the type definition of the component (see metavariables *vector* and *repeating-group*).

```
range:=index1 – index2
```

range
: Designates an occurrence range. The range limits must not be index variables and the following rule must be observed:
0 < *index1* < *index2* ≤ *repetitions*, where *repetitions* is the specification in the type definition of the component.

var-name2
: Name of a matrix.

var-name2 must start with the character "&" and may consist of a maximum of 32 characters.

index1, index2
: Designates a matrix occurrence, i.e. a matrix element.

range1, range2
: Designates an occurrence range. The range limits must not be index variables and the following rule must be observed:
lower limit < upper limit ≤ designates a matrix occurrence, i.e. a matrix element.

Example

`"language(1)"` specifies a field component with an occurrence of 1.
`"language(2–5)"` specifies a field component with an occurrence of 2 through 5.

In *suffix* you may not specify more than three index components since a maximum of three repeating groups can be nested in data groups. Only the last index component to be specified may be a range.

Example

```
DECLARE VARIABLE 1 &address
                   2 city CHAR (10),
                   2 street CHAR (20);
```

&address.* is an abbreviation for &address.city, &address.street.

The component "city" can be addressed as follows:

```
SET  &address.city='Munich';
```

or as an "independent" variable:

```
SET  &city='Munich';
```

vector
Define a vector

vector is used to specify the data type "vector" or "multiple field" for a variable. *vector* consists of a fixed number of components, all having the same basic data type. The number of components is specified by the repetition factor *n*.

For additional structure types, see the metavariable *structure-type*.

```
vector::=( n ) basic-data-type base-type
```

---

n	Repetition factor (0 < *n* < 256).
basic-data-type	See the metavariable *basic-data-type*.
base-type	See the metavariable *base-type*.

Example

The variable &sales (12) is a vector comprising 12 fields. Each field has 7 digits before the decimal point and 2 decimal places and is initialized with 0.

```
DECLARE VARIABLE &sales (12) NUMERIC (9,2) INIT 0;
```

The value "12345.67" is assigned to the seventh field.

```
SET &sales (7)=12345.67;
```

6 Messages

This chapter lists all the DRIVE messages which DRIVE/WINDOWS outputs on the various platforms (BS2000, SINIX and MS-WINDOWS). They are identified by a unique key.

DRI0008 PLEASE ENTER STATEMENT

Meaning
DRIVE is in interactive mode and is expecting a statement.

DRI0009 STATEMENT EXECUTED

Response
Enter next statement.

DRI0010 MEMORY BOTTLENECK

Response
Release memory space, for example by
- deleting EDT work files
- calling smaller programs
- releasing all views in interactive mode
- using an extended system (XS)

DRI0011 SYNTAX ERROR

Meaning
The statement entered does not comply with the syntax rules.You can obtain an explanation of the syntax using the HELP statement. Possible causes of error:
- The name used is a DRIVE keyword.
- The syntax element is not permitted at the position indicated.
- A literal is too long or incorrectly specified.

Response
Correct the syntax using the HELP statement. Repeat statement.

DRI0012 VARIABLE MUST NOT BE INDEXED

DRI0013 MEMORY REQUIREMENT FOR VARIABLE / EXPRESSION TOO LARGE

Meaning
The memory requirement for the variable or expression is greater than the maximum permissible size of approx. 31 Kbytes.

DRI0014 OBJECT NOT DEFINED

Meaning
The object specified has not been made known to DRIVE in a declaration or a database object has been defined for a different environment.

Response
Declare the object or check the name specified.

DRI0015 ILLEGAL DATA TYPE

Meaning
This data type is not permitted here. The length of a variable of the CHARACTER data type may be illegal.

DRI0016 ILLEGAL VALUE

Meaning
The value specified is not within the permitted range of values.

DRI0017 OBJECT ALREADY EXISTS

Meaning
The name of the object has already been used.

Response
Use a different name.

DRI0018 ENTRY ONLY PERMITTED AT HIGHEST LEVEL

DRI0019 NO MEMORY SPACE AVAILABLE FOR '(&00)'

Meaning
(&00): Name of the system in which the memory bottleneck has occurred. Name of file for which the memory bottleneck has occurred.

DRI0020 ILLEGAL MATHEMATICAL OPERATION OR FUNCTION

Meaning
The function or mathematical operation specified is not permitted in the current statement (e.g. numeric predicates and the value $PI are not permitted in DB statements).

DRI0021 ONLY SIMPLE FIELDS PERMITTED

Meaning
Structured fields can only be used together with the comparison operator '='. Other comparison operators are only permitted together with simple fields.

Response
Compare individual components.

DRIO022 '(&00)' MESSAGE: (&01) ((&02))

Meaning
(&00): BS2000/DMS/EDT/FHS/LMS/TIAM/TOM-REF/UTM
(&01),(&02): Message numbers:

LMS:	(&01): LMS return code; (&02): PLAM and DMS return codes.
EDT:	(&02): EDT return code.
UTM:	(&01): KDCS error code; (&02): internal UTM error code.
BS2000:	(&01): INTTRACE; (&02): DMS return code.
FHS:	(&01): Main return code; (&02): category, reason.
TIAM:	(&02): TIAM return code.
TOM-REF:	(&02): Input/output status (see COB-1 manual).

Response
Information on return codes can be found in the respective system manuals or can be obtained via the BS2000 command /HELP-MESS at system level.

DRIO023 STATEMENT LOCKED

Meaning
This statement has been locked for the user.

Response
Do not use this statement or have the administrator remove the lock.

DRIO024 ILLEGAL ORDER OF STATEMENTS

Meaning
Permitted order of statements:
- OPTION statement(s)
- PROCEDURE statement(s)
  declarative statements
  executable statements
- END PROCEDURE statement

For reports: OPEN REPORT must precede FILL REPORT or CLOSE REPORT.

Response
Enter statements in the correct order.
For reports: Include OPEN REPORT statement.

DRIO025 OBJECT MISSING

Meaning
The object (e.g. library, member) is missing or not available in the format requested (e.g. library is not a PLAM library).

Response
Create the object or select an existing object. If the object is a library, it is possible that a file of this name that is not a PLAM library exists.

DRI0026 OBJECT LOCKED

Meaning
The object (e.g. a member of a PLAM library) is locked.

Response
Wait until the object is released or select a different object.

DRI0027 STATEMENT NOT PERMITTED IN '(&00)' MODE

Meaning
This statement may not be used in the specified mode.
(&00): UTM
TIAM
PROGRAM
INTERACTIVE
IDP
ENTER
DISPATCH
SERVER
BATCH

DRI0028 NO MORE THAN THREE INDEX LEVELS PERMITTED

DRI0029 AMBIGUOUS NAME

Meaning
The specified name must be identified unambiguously.

DRI0030 '(&00)' EXPECTED

Meaning
The DRIVE syntax element specified is missing at the position marked.
(&00): DRIVE syntax element.

DRI0031 NAME TOO LONG

DRI0032 '(&00)' CONTAINS (&01) ERRORS

Meaning
(&00): program name
(&01): number of errors

Response
Correct and analyze source program again.

DRI0033 DYNAMIC STATEMENT NOT PERMITTED

Meaning
This statement is not permitted in the EXEC statement.

DRI0034 ILLEGAL GROUP DEFINITION

Meaning
A variable group may not be defined.

DRI0035 COMPONENTS NOT IN SAME GROUP

DRI0036 VARIABLE IN HIGHER-LEVEL GROUP

Meaning
In a REDEFINES or LIKE specification, the variable itself may not be in a higher-level group.

DRI0037 ONLY A GROUP MAY BE SPECIFIED

Meaning
- A LIKE specification must refer to a group.
- A .* specification is only permitted for a variable of the group type.

DRI0038 ILLEGAL INDEX SPECIFICATION(S)

Meaning
- CHECK condition: No index may be specified since the check condition is valid for all occurrences of a vector or repeating group.
- An index may contain only constant entries.
- The indexed variable is not a vector or repeating group.
- An index must be numeric with a scale factor 0.

DRI0039 CHECK CONDITION NOT SATISFIED

Meaning
The condition specified in the CHECK clause is not satisfied.

Response
Modify the value in accordance with the check condition specified in the definition of the variable.

DRI0040 ILLEGAL VARIABLE SPECIFICATION

Meaning
- Only the defined variable itself is permitted in the check condition.
- Variables may not be specified as the index in DB statements.

DRI0041 VARIABLE ALREADY REDEFINED

Meaning
A variable that has already been redefined was addressed when redefining a variable.

DRI0042 NO FURTHER COMPONENTS PERMITTED IN GROUP CREATED WITH 'LIKE'

Meaning
A variable or component created with LIKE may not contain any additional components.

DRI0043 '(&00)' IS EMPTY

Meaning
(&00): - EDT work file 0: no statement held
- Form name: a dynamic form requires neither a TTITLE nor a BTITLE definition; no FILL statement was specified for the form before the DISPLAY statement.
- List file name: the list file is empty.
- DATA DICTIONARY.

Response
- Fill the EDT work file.
- Fill the form using a FILL statement.
- Fill the list file.
- Create or fill the data dictionary.

DRI0044 STATEMENT '(&00)' IN PROGRAM BLOCK (&01) ILLEGAL

Meaning
The indicated statement is not permitted in the program block of the current procedure.
(&00): ADD WINDOW/NEW WINDOW/NEXT WINDOW
(&01): BODY/SCRIPT-INIT/SCRIPT-ON

Response
Remove/modify statement/part of statement.

DRI0045 '(&00)' NOT COMPATIBLE WITH CURRENT DRIVE VERSION

Meaning
The program specified cannot be compiled or executed with the DRIVE version in use.
(&00): program name
(&00): program name with suffix CODE: intermediate code

Response
Convert the program in accordance with the DRIVE version in use or create new intermediate code

DRI0046 OVERWRITE '(&00)'? REPLY: (Y=YES; N=NO)

Meaning
(&00): program name.

Response
Y: The EDT work file 0 respectively source program is overwritten.
N: The EDT work file 0 respectively source program is not overwritten.

DRI0047 *** ERROR IN LINE (&00) OF THE EXPANDED MEMBER

Meaning
An error has occurred in the specified line of the expanded copy member. The precise error position is stored in EDT work file 9.

DRI0048 MAXIMUM NUMBER OF LINES PERMITTED ((&00)) EXCEEDED

DRI0049 MAXIMUM LINE LENGTH ((&00)) EXCEEDED

Meaning
(&00): maximum number of characters permitted per line, e.g.
- 256 in EDT work files
- 80 on the terminal
- according to the specification in the DECLARE FORM statement.

Response
- Shorten line.
- Define a form with longer lines.

DRI0050 '(&00)' NAMING CONVENTIONS VIOLATED

Meaning
The naming conventions of the subsystem or of DRIVE have not been observed.
(&00): DRIVE
EDT
PLAM.

Response
Check the naming conventions in the respective manuals.

DRI0051 '(&00)' PARAMETER ALREADY SUPPLIED

Meaning
Values that have already been supplied may not be changed.

DRI0052 '(&00)' PARAMETER NOT SUPPLIED

Meaning
(&00): parameter that has not been supplied.

Response
Supply the parameter using a PARAMETER statement.

DRI0053 '(&00)' CANNOT BE ACCESSED

Meaning
An attempt has been made to access a library (LIBRARY, FORMLIB) that is under a different user ID and is not available for foreign access.

Response
Make library available for foreign access or use a different library.

DRI0054 '(&00)' LOCKED

Meaning

(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code
program name with suffix LIST: compiler listing
filename: file
The specified object cannot be accessed from DRIVE because it is locked by another user.

(&00): DATA DICTIONARY
Access is currently not possible due to parallel updating accesses of more than one task.

Response

Initiate object release.

DRI0055 '(&00)' NOT FOUND

Meaning

(&00): module name: the module could not be dynamically loaded.
library name: the library does not exist, is not shareable or cannot be initialized under a foreign ID.
member name with no suffix: source program
The intermediate code, compiler listing and usage reference are deleted in the DATA DICTIONARY even if the message follows an 'UNSAVE member-name' for a member that does not exist.
object name: object code
Loading of object code with version suitable to the run time system's version failed.
member name with suffix CODE: intermediate code
member name with suffix LIST: compiler listing
DATA DICTIONARY: the data dictionary does not exist or PARAMETER DD has not been initiated.

The following also applies to the SINIX operating system:
Directory: part of the pathname does not exist.
File: file of the specified name does not exist.

Response

- FHS modules must be held in the library with the link name FORMOML.
- Initialize library or library member.
- Generate object code with a compiler's version matching the run time system's version and insert object code into the object library.
- Create data dictionary.

The following also applies to the SINIX operating system:
- Check and correct pathname.
- Check and correct pathname.

DRIO056 POSITION OF '(&OO)' IN XS SPACE NOT CONSISTENT WITH OTHER MODULES

Meaning
Different address space was addressed during dynamic loading of a module in an XS system.

Response
Inform administrator.

DRIO057 STATEMENT TRUNCATED TO PERMITTED LENGTH

Meaning
- The first statement in the copy member does not fit on the screen and has therefore been truncated to the maximum length permitted.
- The statement repeated in REPEAT was too long and has been truncated.

Response
Shorten the statement (e.g. by removing blanks).

DRIO058 VIEW DECLARATION OF A VIEW NOT PERMITTED

Meaning
A view may not be specified in a FROM clause when defining a view.

DRIO059 ONLY ONE VIEW OR BASE TABLES PERMITTED

Meaning
Only one view or one or more base tables may be specified in the FROM clause.

DRIO060 SPECIFIED STATEMENT INCOMPLETE

DRIO061 '(&OO)' '(&O1)' INVALID OR NOT GENERATED

Meaning
(&00): TAC:
The generation error depends on the transaction code specified (&01):

DRISQL:	only permitted as FIRST-TAC
DRISQLF/SQLNEXT:	only permitted as NEXT-TACs
SQLENTER/SQLLIST:	only permitted as asynchronous TACs
SQLRMT/SQLRMTA/SQLRET:	TACs for distributed transaction processing

Additional transaction codes may mean:
- the transaction code has not been generated
- no dialog TAC in the case of a dialog call
- no ATAC in the case of an asynchronous call

(&00): LTERM:
(&01): LTERM name

Response
Regenerate the application with KDCDEF, specifying the correct TAC or LTERM name.

DRI0062 INVALID K OR F-KEY

Meaning
Input was made via a K or F key that was not defined with PARAMETER KFKEY or declared via the UTM SFUNC macro.

Response
Repeat entry using another key.

DRI0063 DOLINE=(&00); RESUME PROGRAM? REPLY: (Y=YES; N=NO)

Meaning
The number of program statements set in PARAMETER DIAGNOSIS DOLINE has been reached.
(&00): current value of the DOLINE parameter

Response
Y: execution of the program is resumed with the next statement.
N: execution of the program is aborted.

DRI0064 '(&00)' ABORTED WITH '(&01)'

Meaning
Execution of the program has been aborted due to
- EXIT,
- a BREAK statement,
- the DOLINE value being reached and the prompt response 'No',
- BREAK being entered at the terminal, K1 key etc.
- calculation overflow or divide error.

(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code
(&01): EXIT/BREAK/PROGRAM ERROR

DRI0065 MEMORY BOTTLENECK WHEN STORING INTERMEDIATE CODE

Meaning
Program analysis successfully completed; the objects to be generated as specified in the OPTION clause (e.g. compiler listing, where-used information) have been generated. However, a memory bottleneck occurred when accessing the PLAM-X member for the intermediate code.

Response
Release memory and repeat COMPILE statement with OPTION CODE=ON.

DRI0066 OBJECT LOCKED DURING INTERMEDIATE CODE STORAGE

Meaning
Program analysis successfully completed; the objects to be generated as specified in the OPTION clause (e.g. compiler listing, where-used information) have been generated. However, the PLAM-X member for storage of the intermediate code is locked.

Response
Wait until the object is no longer locked or select a different object. Repeat the COMPILE statement with OPTION CODE=ON.

DRI0067 '(&00)' MESSAGE: (&01) ((&02)) DURING INTERMEDIATE CODE STORAGE

Meaning
Program analysis successfully completed; the objects to be generated as specified in the OPTION clause (e.g. compiler listing, where-used information) have been generated. However, a status error occurred when accessing the PLAM-X member for the intermediate code.
(&00): PLAM
(&01): PLAM return code
(&02): DMS return code

Response
The information on return codes can be found in the respective system manuals or can be obtained via the BS2000 command /HELP-MESS at system level. Repeat COMPILE statement with OPTION CODE=ON.

DRI0068 INVALID '(&00)' ENTRY

Meaning
(&00): Invalid entry, e.g.
- STATUS, FILE, LTERM, DEVICE in LIST statement
- LIST in DRIVE formatting
- SCHEMA, PASSWORD, USERGROUP, USERNAME in PERMIT statement
- ITEM entry

Response
Make numeric specification for selection identifier in PRESELECT ITEM statement.

DRI0069 'SYSTEM' COMMAND INVALID OR ILLEGAL

DRI0070 PARAMETER TRANSFER AREA LARGER THAN (&00) BYTES

Meaning
The sum (value range, description) of the parameter values specified in the USING clause exceeds the maximum area permitted.
(&00): maximum size of the transfer area.

Response
Transfer fewer or shorter parameters.

DRIO071 ERROR IN THE IMPLICIT 'COPY' MEMBER

Meaning
DECLARE SCREEN statement:
EUA addressing aid incorrect or missing.
USE VIEWS statement:
The read-in view declaration contains errors.

Response
Generate a new addressing aid or store view declaration again.

DRIO072 RECURSIVE '(&00)' CALL NOT PERMITTED

Meaning
DRIVE does not permit any recursive program or subprogram calls.
(&00): SUBPROCEDURE
program name with no suffix: source program
program name with suffix CODE: intermediate code

DRIO073 '(&00)' CONTAINS INPUT/OUTPUT STATEMENTS

Meaning
The program cannot be used as an ENTER program as this must not contain any input/output statements.
(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code

DRIO074 SYSTEM PROGRAM '(&00)' NOT FOUND/INCORRECT

Meaning
(&00): name of the system program

Response
Inform administrator.

DRIO075 SYSTEM LIBRARY '(&00)' NOT FOUND

Meaning
(&00): name of the system library

Response
Inform administrator.

DRIO076 NO METAINFORMATION FOUND

Meaning
No information on the specified view or cursor etc. was found by a SHOW statement.

DRI0077 NO 'DRIVE' STATEMENT FOUND

Meaning
- An analysis cannot be carried out because either the EDT work file 0 or the PLAM member does not contain any DRIVE statements.
- The REPEAT statement found no previously stored statements.

DRI0078 INTERNAL 'DRIVE' ERROR '(&00)' IN '(&01)' PROCEDURE

Meaning
DRIVE has been aborted due to internal inconsistencies. At the same time, diagnostic information has been generated in the form of a dump.
(&00): internal error number
(&01): name of an internal procedure

Response
Forward diagnostic information to the administrator.

DRI0079 SPECIFICATION ONLY PERMITTED WITH SIMPLE VARIABLES

Meaning
The indicated clause is only permitted if no expression is involved or if the variable referred to in the clause is not a group.

Response
Remove the clause concerned or do not use a structured variable.

DRI0080 INDICATED CLAUSE ALREADY SPECIFIED IN THE STATEMENT

Response
Enter the clause only once.

DRI0081 SPECIFICATION ONLY PERMITTED FOR FORMS

Meaning
The IMAGE or INIT clause is only permitted for forms, and not for lists or in the SEND MESSAGE statement.

Response
Delete IMAGE or INIT clause.

DRI0082 ONLY 'FHS' FORMS PERMITTED

Meaning
Only FHS forms are permitted in a DISPLAY statement containing more than one form name.

Response
Change or delete form name in the DISPLAY statement.

DRI0083 ILLEGAL '(&00)' SPECIFICATION

Meaning
- NEWLINE and NEWPAGE are not permitted in a SEND MESSAGE statement.
- TABLE may not be specified in conjunction with NEWLINE or NEWPAGE.
- A DISPLAY statement within a COMMIT or STOP statement may not have the effect of writing to a list form.
- RETURN may only be specified in conjunction with forms, not lists.
- RETURN may not be specified immediately after a NAMES specification.
- NEWPAGE is not permitted in a DECLARE FORM statement.
- A DISPLAY statement may only contain a SCREENERROR clause if the statement refers to an FHS form.
- USING may not be specified if no USING clause is defined in the program called.
- INVISIBLE may not be specified immediately after a RETURN specification.
- TRACE is not permitted in DEBUG if no interpreter listing is currently available or the interpreter listing and the source program do not match.
  (&00): TRACE

Response
Remove the illegal specification.
Generate valid interpreter listing.

DRI0084 ENTRY ONLY PERMITTED WITH NO TRANSACTION OPEN

Meaning
- In interactive mode, no program may be called while a transaction is open.
- The PARAMETER DYNAMIC NORMSQL statement is only permitted when no transaction is open.
- The STOP statement is not executed.

Response
End transaction and repeat entry.

DRI0085 '(&00)' VARIABLE CONTAINS NULL VALUE

Meaning
- At execution time, a DRIVE variable that is to be transferred to the DB system may not contain the NULL value.
- The NULL value may not be assigned to the variable.

(&00): variable name

Response
Supply the variable with a permitted value.

DRI0086 TRANSACTION TERMINATED; PRESS SEND KEY

DRI0087 ERROR: '(&00)' ABORTED

Meaning

(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code
- Runtime errors occurred during execution of the program initiated with DO or ENTER. For ENTER, the associated detailed error list has been written to SYSLST.

(&00): name of the DRIVE variant loaded
- DRIVE has been aborted due to an error. A corresponding message is output or diagnostic information generated.

Response

- Correct the ENTER procedure involved and restart.
- Forward diagnostic information to the administrator if necessary.

DRI0088 '(&00)' TERMINATED NORMALLY

Meaning

- normal termination of DRIVE
  (&00): program name of the DRIVE variant loaded
- normal termination of the application program (&00).
  (&00): program name with no suffix : source program
  program name with suffix CODE: intermediate code

DRI0089 *** SERIOUS ERROR. PROGRAM ANALYSIS ABORTED

Meaning

A serious error occurred during program analysis. The rest of the program could not be analyzed.

DRI0090 BOTTLENECK IN CLASS 5 MEMORY DURING DMS MACRO EXECUTION

Response

- Inform administrator.
- Enlarge class 5 memory.

DRI0091 UTM MESSAGE IN '(&00)'. KCRC (&01) ((&02))

Meaning

(&00): UTM function (e.g. INIT, MGET, ...)
(&01): KCRCCC = UTM error code
(&02): KCRCDC = internal UTM error code
For further information, see the UTM manual.

Response

Remove the error, e.g. by changing the entries in KDCDEF, and restart DRIVE.

DRI0092 STATEMENT NOT PERMITTED IN THIS OPERATING SYSTEM

Meaning
A statement was entered in the SINIX or BS2000 operating system which is only permitted in the BS2000 or SINIX operating system respectively.

DRI0093 CONTROL VARIABLE MUST NOT BE CHANGED

Meaning
The attempt was made to change the control variable of a CYCLE FOR statement during cycle execution. For report generation: An attemp was made to change a variable defining a report between OPEN REPORT and CLOSE REPORT.

Response
Replace the control variable by a different variable (e.g. by SET) in the indicated statement.

DRI0094 DD LOCKED; IDDS STATUS: (&00), (&01)

Meaning
A data dictionary, an entity or relationship cannot be accessed.
Possible reasons:
- Record locked due to parallel transactions
- Dictionary was deactivated

Any active transactions are normally rolled back; refer the DRIVE manual for more information.
(&00): error number
(&01): error text

Response
The IDDS code gives the exact reason for the lock. Repeat DD request after releasing locked object.

DRI0095 DD RESOURCE BOTTLENECK; IDDS STATUS: (&00), (&01)

Meaning
A bottleneck occurred in resources for ERMS. Possible reasons:
- no cursor ID available
- no memory space available

The current transaction was rolled back.
(&00): error number
(&01): error text

Response
Refer to ERMS manual for the meaning of the IDDS code. The ERMS administrator may have to take action.

DRI0096 ERROR IN '(&00)' IN LINE (&01):

Meaning
Header for a list of errors.
Another error occurred in line (&01) of program (&00) during processing of a WHENEVER event (see compiler listing).
Under MS-Windows in RTS mode, an invalid statement in the start file may be the cause of this error message.
(&00): program name with no suffix : source program
program name with suffix CODE: intermediate code
Name of the start file under MS-Windows
(&01): line number in the compiler listing
line number in the start file

Response
Diagnose the error using the specified program name and line number in the compiler listing.
In the case MS-Windows, correct invalid statement in start file.

DRI0097 DISPLAY ERROR? REPLY: (EDT=DISPLAY; BREAK=ABORT)

Meaning
Following program analysis in the EDT work file 0, the EDT statement is used to branch to the editor. The analyzed source program is stored in work file 0 together with the inserted error messages, the complete compiler listing is stored in work file 9.

DRI0098 '(&00)' FORM NOT YET DISPLAYED

Meaning
- Parallel processing of more than one list form is not possible.
  A list form filled with FILL must be output using a DISPLAY statement before the next form can be filled.
- All screen outputs must be completed before a program called with DO can be terminated.
- All temporary forms declared in a program called with CALL must be completed before that program can be terminated.

(&00): form name

Response
- Display the list form previously processed using a DISPLAY statement, or
- Change the form name specified in the FILL statement to the current form name.

DRI0099 '(&00)' CONTAINS LOCKED STATEMENTS

Meaning
The called program includes statements that are locked for the user by a PARAMETER statement.
(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code

Response
The locked statement can only be changed in a new DRIVE session.

DRI0100 '(&00)' ONLY EXECUTABLE WITH 'CALL'

Meaning
A program can only be executed with CALL if it contains output parameters, for instance.
(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code

DRI0101 OPEN TRANSACTION IN '(&00)' ROLLED BACK

Meaning
All transactions must be closed before terminating a program called with DO, i.e. a COMMIT WORK or ROLLBACK WORK statement must be executed dynamically before any of the statements END PROCEDURE, BREAK PROCEDURE, STOP or another DO statement.
(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code

DRI0102 ILLEGAL PERIOD OF VALIDITY ('TEMPORARY', 'PERMANENT')

Meaning
The redefined variable and the variable to be redefined have different TEMPORARY and PERMANENT specifications.

Response
Declare both variables with either TEMPORARY or PERMANENT.

DRI0103 RANGE SPECIFICATION ONLY PERMITTED IN LAST COMPONENT

DRI0104 'LIKE' COMPONENT CANNOT BE REDEFINED

Meaning
The group contains components that redefine other components.

Response
Declare the variable without LIKE and remove the REDEFINES specification from LIKE group.

DRI0105 INDEX SPECIFICATION REQUIRED

Meaning
An index must be specified as the variable is a vector.

DRI0106 '(&00)' OPTION ONLY PERMITTED WITH MEMBER-NAME

Meaning
As no member name has been declared, the specified option in the COMPILE statement cannot be fulfilled.
(&00): LISTING
CODE

Response
Specify a member name or omit option.

DRI0107 '(&00)' ALREADY USED AS INPUT FIELD

Meaning
An input field has been specified more than once in FILL statements that refer to the same DRIVE form.
(&00): variable name

DRI0108 INCORRECT STRUCTURE IN DRIVE-MESSAGE '(&00)'

Meaning
The message held in the messagefile does not comply with the conventions for messages with responses.
(&00): message number

Response
Change DRIVE message according to the conventions.

DRI0109 TOTAL LENGTH OF INPUT DATA TOO SHORT

Meaning
During reading in of values to a DRIVE form,
- the end marker was entered in an input field, or
- a record that is too short was read during processing with the SYSDTA file.

Response
- Remove end marker, or
- enlarge the record in the SYSDTA file to the expected input length.

DRI0110 TOTAL LENGTH OF INPUT DATA TOO GREAT

Meaning
A record that is too long was read during reading in of values to a DRIVE form via the SYSDTA file.

Response
Shorten the record in the SYSDTA file to the expected input length.

DRIO111 '(&00)' HAS ILLEGAL FILE CHARACTERISTICS

Meaning
(&00): link name (DRILIST, INTTRACE)
The following file characteristics are required:

Characteristic	DRILIST	INTTRACE
FCBTYBE	ISAM	ISAM
RECFORM	V	V
BLKSIZE	(STD,b) b<=16	(STD,16)
KEYPOS	5	5
KEYLEN	24	32
OPEN	INOUT	INOUT
SHAREUPD	YES	YES
SPACE	(b*2+1,b)	(33,16)

Response
Delete errored file. If necessary, DRIVE automatically recreates the INTTRACE file.

DRIO112 EXIT ROUTINE CANNOT BE EXECUTED

Meaning
The exit routine could not be found in the F.EXITLIB library.

DRIO113 ILLEGAL 'FHS' FORM SPECIFICATION

Meaning
The FILL statement is not permitted for FHS forms.

DRIO114 '(&00)' NOT OPEN OR GENERATED

Meaning
A parent window specified for ADD WINDOW must have been generated and opened.
(&00): window name

DRIO115 WINDOW '(&00)' NOT DEFINED

Meaning
The window to be opened via ADD/NEW/NEXT WINDOW does not exist.

DRIO116 '(&00)' CANNOT BE EXECUTED/COMPILED IN '(&01)' MODE

Meaning
The program cannot be executed/compiled in the specified terminal mode.
(&00): program name
(&01): 'WINDOW' or 'ALPHA'

DRI0117 WINDOW '(&00)' ALREADY OPEN

Meaning
The window to be opened via ADD/NEW/NEXT WINDOW is already open.
(&00): window name

DRI0118 '(&00)' NOT PERMITTED ON SCREEN IN ASYNCHCRONOUS UTM OPERATION

Meaning
PTRACE outputs to the screen and DOLINE-PROMPTING when the DOLINE value is reached are not permitted in asynchronous UTM operation.
(&00): PTRACE
DOLINE-PROMPTING

DRI0119 NO ACCESS RIGHT FOR SPECIFIED TABLE

Meaning
In interactive UTM applications the access rights for the tables used must be held in the data dictionary if PARAMETER PERMISSION=ON has been set. If the access right for the statement issued (e.g INSERT) has not been granted, the statement is rejected.

Response
Grant new access rights, if necessary.

DRI0120 'OF' TYPE NOT COMPATIBLE WITH 'CASE' TYPE

Meaning
'value-expr' was specified for 'OF', whereas 'search-cond' was specified for 'CASE', or vice versa.

DRI0121 SPECIFIED OBJECT NOT DYNAMIC

Meaning
A DROP statement may only refer to objects (view, cursor) generated dynamically (EXECUTE statement).

DRI0122 NESTED 'COPY' CALLS NOT PERMITTED

Meaning
A COPY statement is not permitted in a copy member.
Exception: Internally, DRIVE executes the DECLARE SCREEN and USE VIEWS statements with the aid of COPY statements. These statements are also permitted in copy members.

Response
Enter call without nesting.

DRI0123 COMPILER LISTING / SOURCE PROGRAM VERSIONS DO NOT MATCH

Meaning
PTRACE cannot be executed because the date in the compiler listing does not match the date in the current program.

Response
Generate a new compiler listing.

DRI0124 STATEMENT MISSING FROM COMPILER LISTING

Meaning
The statement was not transferred to the compile list, perhaps because the statement did not begin in the source program until column 256. For this reason, the statement can also not be output using PTRACE.

Response
Correct the source program accordingly.

DRI0125 CURRENT 'SPWA' (=(&00)) TOO SMALL

Meaning
The SPWA was generated too small, meaning the ENTER statement could not be executed.

Response
Increase SPWA size.

DRI0126 'DRILOG' ABORTED, 'LOG' FUNCTION RESET

Meaning
The DRILOG request could not be executed due to an ITC error.

Response
Ask the administrator which problem occurred in the ENTER procedure DRI.ENT.DRILOG. Remove the error and repeat the PARAMETER statement to start DRILOG (PARAMETER DYNAMIC LOG).

DRI0127 'DRILOG' NOT LOADED, 'LOG' FUNCTION NOT EXECUTABLE

Meaning
The ENTER procedure DRI.ENT.DRILOG could not be started.

Response
- Check whether the ENTER procedure DRI.ENT.DRILOG and the program PRO.DRILOG are under the same user ID.
- Check whether BS2000 system overload caused the batch task not to be started. Repeat the PARAMETER statement to start DRILOG.

DRI0128 EDT WORK FILE 0 NOT EMPTY. TERMINATE 'DRIVE'? (Y=YES; N=NO)

Meaning
The EDT work file 0 still had unsaved contents when STOP was entered.

Response
Y: DRIVE is terminated; the unsaved contents are lost.
N: DRIVE is not terminated.
Enter a SAVE statement; repeat STOP statement.

DRI0129 UP TO (&00) 'FHS' FORMS PERMITTED

Meaning
No more than one FHS form may be specified in the DISPLAY statement per screen line.
(&00): maximum number of forms permitted in a DISPLAY statement

Response
Reduce the number of forms used.

DRI0130 THE FOLLOWING ERROR MESSAGES COULD NOT BE ASSIGNED:

Meaning
The listed error messages could not be correctly assigned to the errored DRIVE statements (EDT work file 0) because the compiler listing has not been fully read in to EDT work file 9.

Response
Shorten the program.

DRI0131 SPECIFIED LIBRARY AND 'OLD-STYLE' PLAM LIBRARY DO NOT MATCH

Meaning
An attempt to start an 'old-style' program was made in mixed operation. However, the library from which this program was to be started does not match the PLAM library previously defined for 'old-style' operation. The program could not be started as only one PLAM library can be defined in 'old-style' operation.

Response
Change library specification and restart program.

DRI0132 VARIABLE NOT PART OF AN 'ADDRESSING AID'

Meaning
The variable must be implicitly defined in a DECLARE SCREEN statement before an ATTRIBUTE clause is permitted.

Response
Generate a new addressing aid with IFG or correct entry.

DRI0133 ONLY ONE ATTRIBUTE PERMITTED PER ATTRIBUTE CLASS

Meaning
A maximum of one attribute from an attribute class may be specified in an ATTRIBUTE clause (e.g. two colors may not be specified at the same time).

DRI0134 'DEFAULT' MUST BE THE FIRST ATTRIBUTE

Meaning
If more than one global attribute is specified, DEFAULT must be first.

DRI0135 NAME OF 'LIST' FILE CHANGED BEFORE RESTART

Meaning
Before DRIVE crashed, a list file with a different name from the one entered for the file at restart was being processed.

Response
Assign the old list file.

DRI0136 INCORRECT 'TOM-REF' MODULE VERSION

Meaning
The version of TOM-REF used for dynamic loading was incorrect (an attempt to generate a data dictionary connection was made).

Response
Inform administrator.

DRI0137 'INDEX ERROR': '(&00)' NOT IN INDEX RANGE OF '(&01)'

Meaning
The value of the index variable is not within the index range of variable(&01).
(&00): index variable name
(&01): variable name

Response
Correct index variable value (e.g. using SET statement).

DRI0138 '(&00)' IN REDEFINED VARIABLE '(&01)'

Meaning
The value of the redefined variable (&01) is only correct in conjunction with the redefinition of (&01).
(&00): CONVERSION ERROR:
The value is not consistent with the data type of (&01).
CHECK ERROR:
The value of (&01) is consistent, but violates the CHECK clause for (&01).

Response
Before incorrectly referencing (&01), correct the value using a SET statement or use a redefined variable instead of (&01).

DRIO139 DIMENSIONS NOT COMPATIBLE

Meaning
The following applies to vektor arithmetics: For +,-, the dimensions must be identical; for *, one factor must be a scalar; for /,%, the divisor or the percentage factor must be a scalar.

DRIO140 (&OO)TH PARAMETER CONTAINS NULL VALUE

Meaning
(&00): position of the errored current parameter in the parameter list. In mixed operation a parameter of an 'old-style' program may not contain the NULL value.

DRIO141 SCHEMA UNKNOWN

Response
Specify the schema name or declare it using a PARAMETER statement.

DRIO142 INTEGER EXPECTED

Meaning
The expression may not contain any variables of the type DECIMAL, NUMERIC or numeric literal with a decimal point, for instance.

DRIO143 PARAMETER MUST CONSIST OF ONLY ONE VARIABLE

Meaning
Only one variable or variable component is permitted as a parameter here (no literals, expressions etc.).

DRIO144 (&OO). PARAMETER NOT TRANSFERRABLE

Meaning
The data type of the specified current parameter cannot be transferred to the formal parameter.
(&00): position of the errored current parameter in the parameter list

Response
Adapt current parameter to the formal interface definition.

DRIO145 'RETURN' ENTRY IN (&OO)TH PARAMETER INVALID

Meaning
The current parameter indicated may not be specified with RETURN since the corresponding formal parameter does not have a RETURN clause.
(&00): position of the errored current parameter in the parameter list

Response
Adapt the specification of the current parameter to the formal interface definition.

DRI0146 '(&00)' ENTRY IN '(&01)'TH PARAMETER MISSING

Meaning
- The parameter must be specified with RETURN if the corresponding formal parameter has a RETURN clause.
- The parameter must be specified with INDICATOR if a NULL value is to be transferred.

(&00): INDICATOR
RETURN
(&01): position of the errored current parameter in the parameter list

Response
- Adapt the specification of the current parameter to the formal interface definition.
- Insert an INDICATOR clause or do not transfer a NULL value.

DRI0147 NUMBER OF CURRENT AND FORMAL PARAMETERS DOES NOT MATCH

Response
Adapt the specification of the current parameter to the formal interface definition.

DRI0148 '(&00)' SPECIFIED WITH 'RETURN' MORE THAN ONCE

Meaning
A parameter or parameter component specified with RETURN is permitted no more than once in a USING clause.
(&00): name of the current parameter

Response
Check the parameter specification.

DRI0149 STATEMENT ABORTED; (BREAK=CLEAR SCREEN)

Meaning
The program analysis or run contained errors.

Response
Acknowledge the prompted BREAK statement with the SEND key. The screen is cleared and new statements can be entered.

DRI0150 NOT ALL THE DB OBJECTS DECLARED COULD BE RELEASED

Meaning
The objects declared by the program analysis in the DB system could not all be released. The declarations are still valid. If an intermediate code was generated, it is retained.

DRI0151 STATEMENT EXECUTED; (BREAK=CLEAR SCREEN)

Response
Acknowledge the prompted BREAK statement with the SEND key. The screen is cleared and new statements can be entered.

DRI0152 PLEASE ENTER TRANSACTION CODE (&00)

Meaning
The DRIVE conversation has been terminated. The user can enter the next transaction code in the input field provided with blanks (&00).

DRI0153 LOAD ERROR IN 'OLD-STYLE' MODULE

Meaning
Possible causes:
1. 'Old-style' operation is not installed, i.e. mixed operation is not possible.
2. The 'old-style' module library has not been assigned.

Response
2. Assign the module library correctly and start mixed operation again.

DRI0154 'DRIVE' SYSTEM LIMITS REACHED ((&00);'(&01)')

Meaning
Possible causes:
- Internal table too large, e.g. in
  - the declaration of a very large variable,
  - the declaration of an FHS form with more than input/output fields.
- A statement too deeply nested or too complex.
- A DB statement too long for the interface to the DB system.

(&00): internal error number, returned by an internal procedure (&01)

Response
- Shorten or simplify statement.
- Define less input/output fields.
- Inform administrator.

DRI0155 MIXED OPERATION ONLY PERMITTED IN 'SESAM' DB SYSTEMS

Meaning
Mixed operation is only possible if DRIVE is running in conjunction with a SESAM data base system.

DRI0156 MAXIMUM NUMBER OF PARAMETERS PERMITTED (128) EXCEEDED

Meaning
A maximum of 128 parameters can be transfered when calling an OLD-STYLE program in mixed mode.

DRI0157 ERMS SESSION OPEN FAILED; IDDS STATUS: (&00), (&01)

Meaning
The ERMS session could not be opened due to one of the following reasons:
- invalid entry in security partition
- subschema for DRIVE not installed
- dictionary specified by environment variable $DRIVE_DD or default setting not available or locked
- ERMS resource bottleneck or error

(&00): error number
(&01): error text

Response
Refer to ERMS manual for the meaning of the indicated IDDS code. Verify the dictionary definition in the environment variable $DRIVE_DD. In the case of installation errors, notify the ERMS administrator.

DRI0158 INCONSISTENCY IN DD; IDDS STATUS: (&00), (&01)

Meaning
The indicated IDDS code was returned on attempting to access the data dictionary because it was found to be inconsistent. The active transaction was rolled back. Further DD processing is not recommended.
(&00): error number
(&01): error text

Response
Refer to ERMS manual for the exact meaning of the IDDS code. It may be necessary to have the ERMS administrator remove any inconsistencies.

DRI0159 ERMS INSTALLATION ERROR; IDDS STATUS: (&00), (&01)

Meaning
The indicated error occurred during an open session. Possible reasons:
- schema not installed correctly
- subschema not installed correctly

The active transaction was rolled back.
(&00): error number
(&01): error text

Response
Notify ERMS administrator. Check if the error is due to fault installation or invalid command files supplied.

DRI0160 NUMERIC OR 'X' CONTROL CHARACTERS DO NOT MATCH TO FIELD DEFINITION

Meaning
An incorrect number or order of control characters for numeric data types ('Z','*','S','9'), 'P' control characters (for numeric types with point) or an 'X' control character (for the CHARACTER data type) was assigned in a mask. The number of these control characters must correspond to the length of the variable for which the mask was specified.

Response
Check the number or order of the mask control characters using the definition of the variable.

DRI0161 (&00) AND (&01) NOT PERMITTED IN A MASK AT THE SAME TIME

Meaning
(&00),(&01): control characters

DRI0162 (&00) ONLY PERMITTED AS FIRST CONTROL CHARACTER IN THE MASK

Meaning
(&00): control character

Response
Check mask for incorrect '+' or '-'.

DRI0163 CONTROL CHARACTER USED MORE THAN ONCE

Meaning
Each of the control characters 'P', '+', '-' and DATE/TIME may only be used once per mask.

Response
Check the control characters in the mask.

DRI0164 INVALID CONTROL CHARACTER (&00)

Meaning
An invalid character or control character has been entered in a mask.
(&00): control character

Response
Check the mask for invalid characters and control characters.

DRI0165 MAXIMUM LENGTH ((&00)) FOR EDITED MASK EXCEEDED

Meaning
(&00): maximum length permitted for an edited mask

DRI0166 (&00) NOT PERMITTED AFTER (&01)

Meaning
1. The control character 'Z' or '*' is only permitted to the left of '9' in a mask.
2. The control character 'Z' or '*' is only permitted to the right of 'P' in a mask if all the control characters up to 'P' and the insertion control character are 'Z' or '*'.
3. Insertion control characters ('.' or 'B') are not permitted to the right of decimal point character 'P' in a mask.

(&00), (&01): control characters

Response
1 and 2: Replace 'Z' or'*' with a numeric control character.
3: Delete the insertion control characters to the right of 'P'.

DRI0167 REPETITION FACTOR INVALID OR INCORRECT

Meaning
In a MASK clause, either a repetition factor was illegally assigned as a control character (e.g. +(4)), or the entry is incorrect (e.g. 9(0)).

Response
Correct or delete repetition factor.

DRI0168 ENTRY OF NULL VALUE USING NULL CHARACTER NOT PERMITTED

Response
Change the NULL value character using a PARAMETER statement and enter the new NULL character accordingly.

DRI0169 MONINFO TEXT FILE INCORRECT

Meaning
The first of the two field symbols ('@') enclosing the identifier is missing in a line of the MONINFO list layout.

Response
Add missing field symbol ('@') to the list layout.

DRI0170 '(&00)' CHARACTER INVALID; CORRECT ENTRY

Meaning
An invalid character was entered during data entry (e.g. incorrect decimal point character).
(&00): invalid character

DRI0171 INPUT VALUE NOT COMPATIBLE WITH VERBAL INPUT

Meaning
The input value in a DATE or TIME input (e.g. '01' for month) is not compatible with verbal input (e.g. 'FEBRUARY').

Response
Match input value and verbal input.

DRIO172 AMBIGUOUS DATA ENTRY; CORRECT INPUT

Meaning
Too few Q or R control characters have been defined in the mask, meaning the entry cannot be interpreted unambiguously, e.g R(2) and entry 'JU' ('JU' can be interpreted as both 'JUNE' and 'JULY').

DRIO173 MASK CLAUSE NOT PERMITTED

Meaning
The MASK clause was specified for a variable for which it is not permitted.
Possible reason:
- Variable not simple
- Variable of illegal data type

Response
Verify the variable and remove the MASK clause if required.

DRIO174 OBJECT '(&00)' ALREADY EXISTS

Meaning
(&00): The name of the object.

DRIO175 '(&00)' STORED

Meaning
The program has successfully been stored as a member of the library.
(&00): program name

DRIO176 '(&00)' NOT STORED

Meaning
The program has not been stored as a member of the library.
(&00): program name

DRIO177 '(&00)' INCONSISTENT

Meaning
The intermediate code of the program has been changed (e.g. shortened) outside DRIVE.
(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code

Response
Generate new intermediate code using a COMPILE statement.

DRI0178 '(&00)' CONTAINS STATEMENTS NOT PERMITTED IN '(&01)' MODE

Meaning
The program contains statements or clauses that are not permitted in UTM mode (e.g. SYSTEM statement).
(&00): program name with no suffix: source program
program name with suffix CODE: intermediate code
(&01): UTM
(&01): UTM asynchronous operation
Remote access to BS2000 databases is not permitted in UTM asynchronous operation.

DRI0179 COMPILER (&00) NOT AVAILABLE

Meaning
The specified compiler is not linked to DRIVE-phase. COMPILE OPTION OBJECT=ON cannot be executed.
(&00): version of compiler

DRI0180 CONVERSION ERROR IN (&00)TH OPERAND OF THE '(&01)' OPERATION

Meaning
Before the operation (&01) could be executed in an expression, a runtime error occurred during conversion of the (&00)th operand of (&01).

Response
Supply the (&01)th operand correctly or remove (&01) operation from the expression together with its operands.

DRI0181 CALC OVERFLOW ((&00)) IN '(&01)' OPERATION

Meaning
A calculation overflow occurred during execution of the operation (&01) as part of the calculation of an expression.
(&00): (MACHINE ERROR): error in machine calculation

DRI0182 DIVISION ERROR IN '(&00)' OPERATION

Meaning
An attempt was made to divide by 0 (not the NULL value) in the (&00) operation of an expression, meaning the expression could not be calculated.

DRI0183 EXPRESSION VALUE TOO LONG FOR ASSIGNMENT TO '(&00)'

Meaning
The result of the expression is too long for the variable.
(&00): variable name

DRI0184 '(&00)' DURING ASSIGNMENT TO '(&01)'

Meaning
Error (&00) occurred during assignment of variables.
(&00): CONVERSION ERROR
CHECK ERROR
(&01): variable name

DRI0185 ILLEGAL FLOAT POINT MASK/REPRESENTATION

Meaning
- An assigned mask contains the control character 'E', but cannot be recognized as a valid float point mask
- 'e' in the data value of a screen input or NUMERIC function cannot be interpreted as float point value.

Response
Verify and correct mask/input/NUMERIC.

DRI0186 SELECT FINDS MORE THAN ONE RECORD

Meaning
The SELECT resulted in more than one record whereas DRIVE supports single SELECT only.

Response
Modify the SELECT so as to obtain a single record; e.g. by using the WHERE clause.

DRI0187 NO FURTHER CURSOR DECLARATIONS POSSIBLE

Meaning
The user cannot declare more than 63 cursors for informix.

Response
Close and release cursors which are no longer required.

DRI0188 '(&00)' AND '(&01)' NOT COMPATIBLE IN ONE PROGRAM SYSTEM

Meaning
Two programs that have been compiled for the indicated database systems must not refer to each other within the same program system.
(&00), (&01): database systems

DRI0189 INFORMIX DATABASE MUST NOT BE CHANGED DURING UTM OPERATION

DRI0190 TOO FEW LINES AVAILABLE FOR '(&00)' OUTPUT

Meaning
The number of output lines for TTITLE and BTITLE defined in the DECLARE FORM statement is greater than the number of implicitly or explicitly defined lines for the output area (screen or list).
(&00): TTITLE
BTITLE

Response
Increase the LINES entry, if present, or modify the TTITLE or BTITLE entry.

DRI0191 MANDATORY FIELD NOT SUPPLIED

Meaning
The indicated field (EDIT STATE = MUST ERROR) must be supplied.

DRI0192 EXPRESSION VALUE TOO LARGE

Meaning
- Maximum length of the expression in the SYSTEM statement: 254
- Maximum length of the follow-up TAC in the STOP statement: 8.

DRI0193 '(&00)' NOT POSSIBLE WITH PROGRAM IN EDT WORK FILE 0

Meaning
Possible causes:
- PTRACE requires the compiler listing. When a program in the EDT work file 0 is started, the name of the program and thus of the compiler listing is unknown.
- UREF requires a program name in order to store the where-used information in the data dictionary.

(&00): PTRACE
UREF

DRI0194 EXPRESSION VALUE TOO LARGE

Meaning
The value of the expression cannot be displayed as it is too large.

DRI0195 '(&00)' CANNOT BE PROCESSED WITH SQL

Meaning
The following characteristics are not permitted:
- scale < 0
- scale > precision
- precision > 15

(&00): name of the column

Response
Update the data base accordingly.

DRI0196 NO TAC OR INCORRECT TAC SPECIFIED IN UPIC FILE

Meaning
The UPIC file is invalid; the TAC name in the UPIC file is not referenced by the BS2000 application.

Response
Verify UPIC file and add TAC if necessary

DRI0197 UPIC FILE OR TNS ENTRY MISSING OR INVALID

Meaning
1. Upic file (side-info file) does not exist in the current directory or is not correct.
2. There is no TNS entry matching the specified USER.

Response
1. Check upic file or include in the current directory.
2. Provide appropriate TNS entry.

DRI0198 NETWORK CONNECTION ABORTED

Meaning
The network connection to the server is down.
Possible causes are:
- invalid TAC
- PEND ER in UTM conversation
- end of UTM application
- connection cleared down by UTM administration
- connection cleared down by transport system

Response
As appropriate,
- check specified TAC
- end and restart UTM application
- notify network administrator

DRI0199 NOT ALL SELECTED OBJECTS PRESENT

Meaning
This error can occur with the 'Lookup' function of DRIVE-SPU. The list displayed in the object class window is not the current list as the corresponding objects were selected in the object class window.

Response
Update list in object class window if desired.

DRI0200 NAME IN '(&00)' TOO LONG

Meaning
Only names with a maximum length of 32 characters can be stored in the data dictionary.
(&00): DATA DICTIONARY
For SINIX: The resource file names in the usage information contained in Window programs must not exceed 54 characters.

Response
Use shorter library names.
For SINIX: Shorten resource file names, where appropriate.

DRI0201 AMBIGUOUS LIBRARY NAME FOR THIS PROGRAM IN '(&00)'

Meaning
According to the data dictionary, the current DRIVE program is held in a different library to the one currently specified.
(&00): DATA DICTIONARY

Response
Make the source program unique with regard to the library.

DRI0202 DATA DICTIONARY PARAMETER FILE INVALID OR NOT FOUND

Meaning
The parameter file assigned with the link name TOMPAR is either empty, not found or contains incorrect entries, or the required entries are missing.

DRI0203 AMBIGUOUS NAME ASSIGNED TO '(&00)' ENTITY IN '(&01)'

Meaning
The where-used information cannot be stored in the data dictionary as there are ambiguous names.
(&00): entity type
(&01): DATA DICTIONARY

Response
Check names in the data dictionary; delete any entities not required.

DRI0204 ELEMENT TOO LONG

Meaning
- The logical screen of the specified form is not large enough to fully receive a data element used in the FILL statement.
- Data element used in SEND MESSAGE statement is too long

Response
- Adapt LINES or COLUMNS entries, if present.
- Shorten the data element concerned.

DRI0205 '(&00)' ERROR IN '(&01)' MODULE/ENTRY

Meaning
An error occurred in the linking loader system during DRIVE initialisation.
(&00): macro (TABLE/ITABL/LINK)
(&01): module or entry name

Response
If the error is in the LINK macro, check whether the DRIVE library was assigned correctly before DRIVE intialisation. If not, inform administrator.

DRI0206 '(&00)' INPUT FIELD NOT PERMITTED IN OVERFLOW SCREEN

Meaning
No input fields are allowed in an overflow screen for a DRIVE form.
(&00): variable name

DRI0207 STATUS OF DB TRANSACTION UNKNOWN AFTER 'COMMIT WORK'

Meaning
Network connection terminated.

Response
For initial call : Verify TAC
For follow-up call : Check status of UTM application (terminated ?)
Notify network administrator.

DRI0208 'UPIC' CONNECTION ENDED ABNORMALLY

Meaning
The connection to the UPIC process or to UPIC was terminated abnormally.
Possible reason are:
- end of UTM application
- connection cleared down by UTM administration
- connection cleared down by transport system

Response
As appropriate,
- terminate UTM application
- notify network administrator

DRIO209 LICENCE/KEY INFORMATION MISSING

Meaning
Signon to UPIC failed

Response
Check UTM installation

DRIO210 TRANSACTION ROLLED BACK BY DATA STORAGE SYSTEM

Response
- INTERACTIVE mode: Repeat SQL statements entered since last COMMIT WORK.
- PROGRAM mode: Restart program (necessary because the first transaction in the DRIVE program was rolled back).

DRIO211 RESTART NOT POSSIBLE IN FIRST TRANSACTION

DRIO212 CONTINUE PROGRAM ? RESPONSE: (Y=YES; N=NO)

Meaning
The transaction was rolled back by the data storage system.
Remote access: The status of the transaction is unknown due to network problems.

Response
Y: The program is rolled back to the status of the last COMMIT WORK and processing is resumed with the first statement after COMMIT WORK.
In the case of remote access:
the program is aborted if the network connection cannot be recovered;
the status of the transaction in the database is unknown.
N: The program is aborted.
In the case of remote access:
the status of the transaction in the database is unknown.

DRIO213 RESTART NOT POSSIBLE; CONVERSATION ABORTED

Meaning
The transaction and thus the conversation were rolled back before the first COMMIT WORK in the UTM conversation.

Response
Start a new conversation.

DRIO214 ABORTED 'UTM' CONVERSATION CANNOT BE RESTARTED

Meaning
Another error occurred in an external restart (e.g. after a system crash). The conversation has been terminated.

Response
Start a new conversation.

DRIO215 TRANSACTION ROLLED BACK DUE TO SQL CODE '(&00)'

Meaning
The transaction was rolled back by the data storage system. SQL objects have the same status as at the last synchronization point.
(&00): SQL code returned by the DB system

DRIO216 REDEFINED OR REDEFINING VARIABLE NOT ALLOWED

Response
Use variable without these properties.

DRIO217 EVENT NOT PERMITTED FOR WINDOW OBJECT

Meaning
The combination of this type of event and this type of window object is illegal.

Response
Modify event or change object.

DRIO218 ORIGINAL LENGTH OF 'TITLE' DATA CHANGED

Meaning
Before displaying a DRIVE form following an explicit or implicit DISPLAY, DRIVE updates the TITLE data . The new length of the TITLE data differs from the original length (e.g. if assignments have been made to TITLE variables using string functions).

Response
Check assignments after the first FILL statement.

DRIO219 CONNECTION TO BS2000 UTM APPLICATION FAILED; UTMRCS=(&00),(&01)

Meaning
The APRO call could not be successfully completed. Refer to UTM manual 'Applications Programming', chapter on KDCS calls, 'APRO' for reason.
(&00): UTM return code
(&01): internal UTM error code

Response
Notify system administrator and check UTM generation.

DRIO220 DRIVE CACHE: RETURN CODE '(&00)' FOR MACRO '(&01)'

Meaning
(&00): return code
(&01): BS2000 macro name

Response
Inform administrator.

DRIO221 DRIVE CACHE CANNOT BE CLOSED

Response
If the error occurs more than once, check whether the HALT TSN (enter procedure DRI.ENT.DRICACHE) is still running at DRIVE termination. If not, determine the cause. If so, inform administrator.

DRIO222 DRIVE CACHE ALREADY EXISTS IN ID WITH DIFFERENT PARAMETERS

Meaning
The DRIVE cache specified exists with a different length to the one given. The length must always be the same when accessing the same cache.

Response
Possible responses:
- Start another DRIVE UTM application under a different ID.
- Change cache length using an ACQUIRE MEMORY statement.

DRIO223 'HALT TSN' FOR DRIVE CACHE CANNOT BE ACTIVATED

Meaning
The ENTER procedure DRI.ENT.DRICACHE for the DRIVE cache was not initiated within 75 seconds.

Response
- Check ENTER procedure and PRO.DRICACHE program.
- Check whether the BS2000 batch queue is overloaded, preventing the ENTER procedure from being started.

DRIO224 NAME SPECIFICATION IN USER LABEL SYNTACTICALLY INCORRECT

Meaning
The library or member name in the user label has been specified incorrectly.

Response
Consult DRIVE manuals.

DRIO225 ERROR DURING USER LABEL PROCESSING

Meaning
The DRIVE program started during processing of the user label contains analysis or runtime errors. The corresponding error list is written to SYSLST.

Response
Correct DRIVE program and start a new UTM application.

DRIO226 PROGRAM ERROR; DRIVE TERMINATED DUE TO 'TEST=ALL'

Meaning
The most recently started program contains analysis or runtime errors. As the parameter TEST=ALL has been set, DRIVE was terminated immediately. The corresponding error list is written to SYSLST.

DRIO227 CONSTANT NOT PERMITTED

Meaning
The object name refers to a constant, which is not useful in this context.

Response
Specify variable.

DRIO228 NO ACCESS TO '(&00)'

Meaning
No access permission has been granted for the indicated element (read/write/ execute or search depending on the file/directory or attempted operation). The permission can also refer to pathname components. An installation error may be the reason why access to a DRIVE system resource file failed.
(&00) : path name.

Response
Change access permissions.
In the failure to access a DRIVE system resource file is due to an installation error, repeat installation

DRIO229 ENTRY INVALID FOR THE DB SYSTEM '(&00)'

Meaning
The statement or part of it must not be used when processing SESAM/UDS databases.
(&00): DB system (SESAM/UDS)

Response
Correct or remove incorrect statement part, or select INFORMIX db system.

DRIO230 VIEW DEFINED WITH 'QUERY' CANNOT BE USED (SQL CODE=(&00))

Meaning
The view defined with QUERY does not comply with the UDS or SESAM conventions (e.g. upper/lowercase, naming conventions).
(&00): SQL CODE returned by the DB system
SQL CODE = 0: DRIVE has already determined the error.
SQL CODE < 0: the DB system has determined the error.

Response
Redefine the view using QUERY, observing the syntactic and semantic rules of the DB system concerned.

DRIO231 ASSOCIATED WINDOW NOT YET GENERATED

Meaning
This window 4GL statement is not valid until the window to which the object referred to belongs has been generated, i.e. following the first ADD/NEW/NEXT statement for the window.

Response
Generate window before running this statement.

DRIO232 SPECIFIED ATTRIBUTE VALUE NOT PERMITTED

Meaning
The value specified in the SET ATTRIBUTE statement is
- outside the permitted value range.
- the NULL value
- too long

Response
Correct data source for attribute value.

DRIO233 SPECIFIED ITEM NAME IS NOT PERMITTED

Meaning
The specified item name is
- the NULL value
- too long (>255)
- in the item list more than once

Response
Specify non-null value for item name, shorten it, or remove duplicates from item list.

DRIO234 ITEM NAME ALREADY EXISTS

Meaning
All item names must be unique within a choice list.

Response
Correct data source for item name.

DRIO235 'POSITION' ENTRY IN 'ADD ITEM' CLAUSE INVALID

Meaning
The position entry must either refer to an existing list item or must be '0'.

Response
Verify whether the POSITION value is between 0 and the number of list items.

DRIO236 SPECIFIED ITEM NAME DOES NOT EXIST

Meaning
Only names of items existing in the list may be specified in DELETE ITEM statements.

Response
Correct data source for the item name.

DRIO237 PATHNAME OF THE REFERENCED WINDOW OBJECT TOO LONG

Meaning
This object cannot be referenced directly in a window 4GL statement.

Response
The individual object names or the complete pathname must be shortened to the maximum length (32768) in the dialog builder.

DRIO238 TOO MANY ENTRIES IN THE CURRENT CHOICE LIST

Meaning
The maximum number of list entries (32767) would be exceeded by making the intended additional entries.

Response
Delete superfluous entries from choice list before making new entries.

DRIO239 STATEMENT REQUIRES "WITH DUPLICATES" CLAUSE

Meaning
An ALTER CHOICE LIST/ALTER COMBO BOX statement containing a "WITH DUPLICATES" clause has already been executed for the current choice list/combo box. Execution without duplicates requires new generation of the window (by ADD/NEW/NEXT WINDOW).

Response
Add "WITH DUPLICATES" clause to statement.

DRIO240 INVALID LIST DEFINITION IN RESOURCE FILE

Meaning
At least one of the entries which are predefined for the choice list/combo box in the resource file exceeds the maximum length (255).

Response
Shorten list entries ("ITEMS" attribute in the attribute editor of the DialogBuilder).

DRIO241 SEVERAL ENTRIES SELECTED FOR SINGLE-CHOICE LIST

Meaning
The list for a single-choice list may only contain one entry. In order to select more than one entry at the same time, use a checklist.

Response
Specify one entry in the "select item clause" in the invalid "ALTER CHOICE LIST" statement, or change the selection mode for the choice list using the attribute editor of the DialogBuilder from single-choice list to checklist ("SELECTION_MODE = MULTIPLE").

DRIO249 RECIPIENT OF USER EVENT '(&00)' NOT AVAILABLE

Meaning
The target window is not on the screen. Possible reason:
- the window has not yet been output,
- the window was closed by means of the CLOSE WINDOW statement or explicitly via the window menu button.

(&00): event.

Response
Check program sources; Intercept user event error by means of WHENEVER if appropriate.

DRIO250 OUTPUT TOO LONG; TOO MANY INTERNAL CONTROL CHARACTERS REQUIRED

Meaning
The output contains too many fields with different display attributes (bright/normal/flashing/overwritable/protected). These attributes are handled internally with control characters (depending on the individual terminal) that make the overall output too long.

Response
Shorten output or use less fields with different display attributes.

DRIO251 '(&00)' CANNOT BE GENERATED

Meaning
(&00): dictionary: The file directory cannot be generated due to lack of memory space;
(&00): window/dialog box: Inconsistency in resource file; window/dialog box does not exist in the resource file.

Response
In the case of memory shortage, delete superfluous files and repeat statement. In the case of inconsistency, check resource file if the window/dialog box to be output exists in the specified resource file.

DRIO252 NO ROOT OBJECT IN '(&00)'

Meaning
The resource file was illegally modified. It cannot be opened.
(&00): resource file

Response
Generate resource file using the dialog builder only.

DRIO253 ERROR IN LINKING CALLBACKS TO RESOURCE FILE

Meaning
The resource file was illegally modified. It cannot be opened.

Response
Remove callback entries from resource file and recompile DRIVE program.

DRIO254 NO TOP LEVEL OBJECT IN RESOURCE FILE '(&00)'

Meaning
There is no top level object (window/dialog box) in the specified resource file.
(&00): resource file

Response
Check name of top level object and resource file.

DRIO255 ERROR ON OPENING RESOURCE FILE '(&00)'

Meaning
The resource file cannot be opened. Possible reason:
- resource file does not exist
- no access permission

(&00): resource file

Response
Verify existence and access permission of the resource file.

DRIO256 XLIB ERROR IN WINDOW SYSTEM

Meaning
'xlib' reports an error if a system call failed; e.g. because the connection to the server has broken down.

DRIO257 ERROR ON WINDOW SYSTEM 'MOTIF'

Response
The message of the window system is stored in the 'inttrace.idx' or 'inttrace.dat' file.

DRIO258 OBJECT '(&00)' CONTAINS TOO MANY CHILD OBJECTS

Meaning
The specified top level object refers to too many child objects. This results in an internal table overflow.
(&00): top level object.

Response
Shorten resource file.

DRIO259 OBJECT '(&00)' INKONSISTENT

Meaning
Possible reasons:
a) Incorrect object class type for window object
b) Invalid data type for window object
c) Top level object no window or dialog box or message box
(&00): object name

Response
For reasons, refer to the resource file.

DRIO260 ERROR ON WRITING 'DRIVE' DATA TO '(&00)'

Meaning
Write access to the resource file is not permitted.
(&00): resource file

Response
Change access permission.

DRIO261 INCONSISTENCY IN RESOURCE FILE '(&00)'

Meaning
User manipulation on the resource file between the compilation and execution of the DRIVE program resulted in an inconsistency.
(&00): error code :

1,2,3:	Window object not found in SET ATTRIBUTE
4,5,6,7:	input field not found
8,9,10,11,12,13:	choice list not found
14:	Window object not found in GET ATTRIBUTE

Response
The resource file must not contain inconsistencies after updates between compilation and execution.

DRIO262 MAXIMUM NUMBER OF 'FORMANT' SESSIONS EXCEEDED

Response
Notify system administrator.

DRIO263 'FORMANT' RETURN CODE (&00) FOR FORM '(&01)'

Meaning
The form cannot be loaded because the Formant form file is inconsistent, not available, or cannot be accessed.
(&00): Formant return code (see Formant manual)
(&01): form name

Response
Check Formant form file and Formant addressing aid.

DRIO264 'FORMANT' RETURN CODE (&00) FOR FIELD '(&01)' OF FORM '(&02)'

Meaning
The specified field name of the specified form was not found, i.e. the specified field name of the Formant addressing aid is not in the Formant form file.
(&00): Formant return code (see Formant manual)
(&01): field name in form
(&02): form name

Response
Check Formant form file and Formant addressing aid.

DRIO265 INCORRECT 'FORMANT' VERSION INSTALLED

Meaning
An incorrect Formant version has been installed.

Response
Notify system administrator.

DRIO266 OBJECT (&00) NOT IN '(&01)'

Meaning
The specified resource file does not contain the specified top level object.
(&00): top level object
(&01): resource file

Response
Check names of the resource file and top level object.

DRIO267 CONNECTION TO X SERVER FAILED

Meaning
The environment variable DISPLAY is illegal or not defined.

Response
Enter correct DISPLAY environment variable.

DRI0280 PROGRAM '(&00)'

Meaning
Information in SPE indicating that the described program is still executing.
(&00): program name

DRI0281 '(&00)' BEING COMPILED

Meaning
Information in SPE that the described program is still being compiled.
(&00): program name

DRI0282 '(&00)' '(&01)' BEING DELETED

Meaning
Information in SPE indicating that the described program is still being deleted, including all associated objects.
(&00): program name
(&01): symbol for activity that cannot be cancelled

DRI0283 EDITOR LOADED WITH '(&00)'

Meaning
Information in SPE indicating that an editor was called together with the described file. This file may not have been saved yet.
(&00): file name

DRI0284 JOB '(&00)' ENTERED IN QUEUE

Meaning
No DRIVE kernel process is currently available for the described job

DRI0285 '(&00)' BEING PRINTED

Meaning
Information in SPE indicating that a print job has been started for the described file.
(&00): file name

DRI0300 MAXIMUM OBJECT SIZE FOR COMPILED 'DRIVE' PROCEDURE EXCEEDED

Meaning
The statement part of the generated object code exceeds 400 slices, each of 4KB.

Response
Check if the option NULLVALUE = OFF or CHECK = OFF is applicable, otherwise split the DRIVE procedure into several parts.

DRI0301 NULL SPECIFICATION NOT PERMITTED

Meaning
The procedure contains a NULL value although NULL processing has been prohibited in the compiler options.

Response
Modify source or compiler options.

DRI0302 NO CALL POSSIBLE

Meaning
The procedure can neither be called as start procedure nor using DO/ENTER/CALL due to conflicting entries in the compiler options. This is the case if a PERMIT screen is requested although the procedure contains parameters for example.

Response
Modify source or compiler options.

DRI0303 TOO MUCH SPACE REQUIRED FOR INTERNAL PERMANENT VARIABLES

Meaning
Internal permanent variables are generated
- for SUBPROCEDURE,
- for CYCLE FOR with a not constant STEP or END value.
The space required exceeds 32 KB.

Response
Decrease the number of the statements above.

DRI0304 TOO MANY TEMPORARIES REQUIRED (&OO)

Meaning
For the compilation of a statement within a DRIVE procedure too many temporaries are required.
The insert text contains the name of the procedure and the linenumber of the statement in the expanded source listing.

Response
Split the statement into several parts.

DRI0305 ERROR DURING GENERATION OR OPEN OF TEMPORARY FILE FOR OBJECT CODE

Meaning
The generated object code is written first into a temporary file which is generated internally. During generation or open of this temporary file an error has occurred. The DMS return code is displayed upon the terminal.

Response
depending on the DMS return code

DRI0306 ERROR DURING CLOSE OF TEMPORARY FILE FOR OBJECT CODE

Meaning
The generated object code is written first into a temporary file which is generated internally. During close of this temporary file an error has occurred. The DMS return code is displayed upon the terminal.

Response
depending on the DMS return code

DRI0307 ERROR DURING WRITE INTO TEMPORARY FILE FOR OBJECT CODE

Meaning
The generated object code is written first into a temporary file which is generated internally. During write into this temporary file an error has occurred. The DMS return code is displayed upon the terminal.

Response
depending on the DMS return code

DRI0308 ACCESS ERROR ON LMS (&00)

Meaning
While moving the generated object code out of the temporary file into the destination library an error has occurred.
The insert text contains the LMS return codes.

Response
depending on the LMS return code

DRI0309 INTERNAL ERROR DURING COMPILATION (&00)

Meaning
During the compilation of a DRIVE procedure an internal error has occurred. The insert text contains the name of the procedure, the linenumber of the statement in the expanded source listing and an internal error number.

Response
Inform your system administration and provide the expanded source listing together with the information in the insert text.
Occasionally further diagnostic information is required.

DRI0310 CONVERSATION '(&00)' TERMINATED. PLEASE ENTER TRANSACTION CODE

Meaning
The indicated conversation in DRIVE/WINDOWS object mode has been terminated.
(&00): Conversation name

Response
Enter transaction code

DRIO311 INTERNAL ERROR IN 'DRIVE' OBJECT '(&00)','(&01)'

Meaning
The running of the DRIVE-object has been aborted due to internal inconsistences
(&00): eventtype
(&01): systemcode

Response
Inform administrator

DRIO312 ILLEGAL NULL VALUE IN 'DRIVE' OBJECT

Meaning
NULL value occurred in an object defined with NULLVALUE=OFF. This may happen in case of data base queries, screen input or parameter transfer in case of CALL, DO, ENTER.

Response
Compile procedure with OPTION NULLVALUE=ON.

DRIO390 FILE (&00) CAN'T BE ERASED.

Meaning
No access rights to modify the directory or file doesn't exist.
(&00): file name

DRIO391 ERROR OCCURED WHEN FILE (&00) WAS WRITTEN.

Meaning
Not enough space on file system.
(&00): file name

DRIO392 FILE (&00) CAN'T BE OPENED OR CREATED.

Meaning
No access rights to modify the directory or existing file can't be opened.
(&00): file name

DRIO393 C-COMPILATION TERMINATED WITH ERRORS.

Meaning
Compilation was aborted by interrupt or system limits (number of processes, memory space) reached or generated C source with errors.

DRI0394 OPENING ERROR ON LIBRARY '(&00)'

Meaning
There is no library according with the stated name, the access right to the library is missing or the library is locked.
The following also applies to SINIX operating system:
Shell-variable LD_LIBRARY_PATH does not contain the pathname of the library.

Response
Generate a correct library, gain the access right and make shure, that the library is not locked at access time (e.g. by a compilation process).
In addition the following applies to the SINIX operating system:
Add the library's pathname to the shell-variable LD_LIBRARY_PATH.

DRI0399 (&00).

Meaning
There is no shared object (shared library) at the stated place.
(&00): library name

DRI0401 INVALID NAME

Meaning
An invalid entry has been made in an input field of the DRIVE SPE.

Response
In the DRIVE SPE, the help button of the window may be used to obtain information on permitted entries and valid names.

DRI0402 OBJECT DELETION ERROR

Meaning
An error occurred when objects (sources, intermediate codes, list elements, resource files, user labels) were being deleted.

DRI0403 SELECT EXACTLY ONE OBJECT

Meaning
The reqired function can only be executed if exectly one object has been selected.

DRI0404 SELECT AT LEAST ONE OBJECT

Meaning
The required function can only be executed if at least one object has been selected.

DRI0405 CLICK AT LEAST ONE TOGGLE BUTTON

DRI0410 AT STATEMENT TOO LONG

Meaning
The AT statement built from the selections in the dialog box is too long.

Response
Shorten entries in the entry fields.

DRI0411 INVALID LIB-SPEC

Meaning
A path name was specified instead of the 'lib-spec'. The member (file) was not enclosed in '(' and ')'.

Response
Either specify the member name (file name), or 'lib-spec' with the member enclosed in '(' and ')'. A path name is not permitted.

DRI0450 TRANSFER AREA INVALID FOR DISTRIBUTED PROCESSING

Meaning
The transfer area submitted to DRIVE as input from a C/COBOL partner for distributed processing is inconsistent or invalid.

Response
The transfer area to be submitted to DRIVE from a C/COBOL partner for processing must be checked for inconsistencies.

DRI0451 MESSAGE LENGTH (&00) TOO SMALL

Meaning
The storage length of the message submitted by the partner conversation is long enough. The header of the transfer information could not be read.
(&00): length of message received

Response
Check message length on sending.

DRI0452 FIELD '(&00)' IN THE TRANSFER INFORMATION HEADER INVALID

Meaning
The indicated field in the header has been supplied with an invalid value by the partner conversation.
(&00): invalid field

Response
Check entry in header field.

DRI0453 NO 'RETURN' PARAMETER VALUES RECEIVED FROM (&00)

Meaning
The receiving partner called with CALL has not returned any RETURN parameter values although the USING clause does contain RETURN parameters.
A switch was made to old-style operation using new-style CALL. The last old-style program executed has no USING list but RETURN parameters are specified in the USING list in the new-style CALL.

Response
Check interface to receiving program (unit).
Match USING list of new-style CALL and old-style program.

DRI0454 ILLEGAL RETURN PARAMETER VALUES FROM RECEIVING PARTNER

Meaning
The receiving partner called with CALL returns RETURN parameter values although the USING clause does not contain any RETURN parameters.

Response
Check interface to receiving program (unit).

DRI0455 CALLED USER TAC '(&00)' TERMINATED WITH PARAMETER ERROR

Meaning
The called user program unit reports errors on reading the header information in the transfer area.
(&00): UTM transaction code

Response
Check distribution information for the concerned user program unit.

DRI0456 CALLED USER TAC '(&00)' TERMINATED WITH ERROR

Meaning
The called user program unit reports errors on program execution.
(&00): UTM transaction code

Response
Check execution of called user program unit and, if required, submitted USING data.

DRI0457 CALLED RECEIVING DRIVE CONVERSATION TERMINATED WITH PARAMETER ERROR

Meaning
The called receiving DRIVE conversation reports errors on reading the header information in the transfer area.

Response
Check distribution information for the receiving DRIVE program concerned.

DRI0458 CALLED RECEIVING DRIVE CONVERSATION TERMINATED WITH ERROR

Meaning
The called receiving DRIVE program reports errors on program execution.

Response
Check execution of called DRIVE program and, if required, submitted USING data.

DRI0459 LIBRARY NAME '(&00)'

Meaning
Library of an executed DRIVE program. Information in addition to parameter error or execution error message, or status information in the case of distributed processing.
(&00): library name

DRI0460 MEMBER NAME '(&00)'

Meaning
Member name of an executed DRIVE program. Information in addition to parameter error or execution error message, or status information in the case of distributed processing.
(&00): member name

DRI0461 APPLICATION NAME '(&00)'

Meaning
Name of a receiving application in which a DRIVE program or user program unit was executed. Information in addition to parameter error or execution error message, or status information in the case of distributed processing.
(&00): application name

DRI0462 ERRORS ON DISTRIBUTED PROCESSING

Meaning
Distributed execution of one or more receiving DRIVE programs or user program units was terminated with errors.

Response
Check distribution information, USING data or program execution.

DRI0463 CONVERSATION RULE FOR DISTRIBUTED PROCESSING VIOLATED

Meaning
Open receiving conversations still exist in the case of:
- a 'DO PROCEDURE' statement
- a 'STOP' or 'COMMIT WORK WITH STOP' statement
- an 'END PROCEDURE' statement on the highest DRIVE program level of the highest submitting conversation in a distributed submitter/receiver hierarchy.

Response
Check flow of the DRIVE program. If one of the above statements exists, the receiving partners must have been terminated.

DRI0464 TRANSACTION RULE FOR DISTRIBUTED PROCESSING VIOLATED

Meaning
One or more DRIVE programs or user program units have not yet executed 'COMMIT WORK' or requested transaction end.

Response
Check flow of DRIVE program. In the case 'COMMIT WORK', the receiving partners must have terminated the transaction.

DRI0465 BOTTOM-UP STRATEGY FOR DISTRIBUTED PROCESSING VIOLATED

Meaning
A receiving DRIVE program was called with transaction end request.

Response
Terminate processing step in program unit using 'PEND KP'.

DRI0466 RECEIVING PARTNERS WITH TRANSACTION STATUS 'P' STILL OPEN

Meaning
Receiving partners which have requested transaction end are still open when a submitting DRIVE program terminates with errors.

Response
Include the 'COMMIT WORK' statement after CALL or END DISPATCH in order to terminate the transactions in the receiving environments.

DRI0467 CALLED USER-RECEIVER CONVERSATION FOR 'PEND' REQUEST INVALID

Meaning
An error occurred in the program unit on calling the user-receiver conversation containing a request for termination, or the values supplied to the header of the transfer area by the user program unit contained errors.

Response
Check interfaces to, or execution of, the user program units.

DRI0468 USER-RECEIVER CONVERSATION NOT TERMINATED DESPITE 'PEND' REQUEST

Meaning
When calling the user-receiver conversation, the request to terminate was ignored.

Response
Check interfaces to, or execution of, the user program units.

DRI0469 RECEIVING TRANSACTION NOT ROLLED BACK DESPITE 'ROLLBACK' REQUEST

Meaning
When calling the user-receiver conversation, the rollback request for the local transaction was ignored.

Response
Check interfaces to, or execution of, the user program units.

DRI0470 CONVERSATION STATUS '(&00)', TRANSACTION STATUS '(&01)' FOR DRIVE CONVERSATION

Meaning
Receiving DRIVE partner submits status information following distributed conversation restart.
(&00): conversation status of receiving partner
(&01): transaction status of receiving partner

Response
Check execution of receiving DRIVE program.

DRI0471 CONVERSATION STATUS '(&00)', TRANSACTION STATUS '(&01)' FOR USER TAC '(&02)'

Meaning
User-receiver conversation submits status information following distributed conversation restart.
(&00): conversation status of receiving partner
(&01): transaction status of receiving partner
(&02): UTM transaction code

Response
Check execution of user-receiver program unit.

DRI0472 CONVERSATION '(&00)' AND TRANSACTION STATUS '(&01)' FOR RECEIVING CONVERSATION

Meaning
Receiving conversation submits status information following distributed conversation restart. The receiving conversation has been referenced for the first time in the restarted distributed transaction. No further information is available from the receiving partner.
(&00): conversation status of receiving partner
(&01): transaction status of receiving partner

Response
Check receiving conversations in distributed application.

DRI0473 STATUS INFO RECEIVED FROM RECEIVING PARTNER AFTER CONVERSATION RESTART

Meaning
Receiving conversation submits status information following distributed conversation restart.

Response
Refer to DRIVE messages containing additional information.

DRI0474 STATEMENT ONLY PERMITTED ON DISPATCH BLOCK PROGRAM LEVEL

Meaning
A DRIVE program called within a DISPATCH block using a local CALL PROCEDURE must not itself contain a REMOTE CALL.

Response
Transfer REMOTE CALL to DISPATCH block compilation unit.

DRI0475 DATATYPE ILLEGAL FOR REMOTE PROCESSING

Meaning
Only the following datatypes can be transferred in USING parameters to DRIVE programs called with CALL that are to be executed remotely:
- literals
- simple variables
- DRIVE expressions
- vectors, matrixes

Response
Check interface to, or execution of, user program units.

DRI0476 OTHER SERVER ALREADY ADDRESSED

Meaning
A server has already been addressed when a UTM server was accessed remotely. It is only possible to address one server at this point.

Response
Check program flow. Tterminate current UTM server conversation before remote access. Remote accesses to SESAM or UDS databases under BS2000 and remote CALLs within a DRIVE/WINDOWS session are not possible at the same time.

DRI0477 SERVER STATUS DOES NOT MATCH AT MOST RECENT SYNC POINT

Meaning
When a ROLLBACK WORK WITH RESET is performed in the client, a different server has been addressed as when the most recent COMMIT WORK was performed. It is not possible to recover the server status of the COMMIT WORK. Reset is therefore not possible.

Response
Check program flow. Avoid ROLLBACK WORK WITH RESET in client if possible.

DRI0478 RESTART NOT POSSIBLE IN SERVER SESSION

Meaning
In a server session (without distributed transaction processing), a ROLLBACK WORK WITH RESET was performed or a "ta_cancelled" was reported by the database existing locally in the server. DRIVE/WINDOWS cannot reset the server program in this situation as UTM does not support conversation restart for client server connections without distributed transaction processing.

Response
Check flow of server program, as well as its accesses to local data resources.

DRI0479 RESTART NOT SUPPORTED

Meaning
The current DRIVE version does not support restart.

Response
Check program source; the ROLLBACK WORK WITH RESET statement is not permitted.

DRI0480 ERROR OCCURRED IN OLD-STYLE PROGRAM

Meaning
A new-style CALL started an old-style program; an error occurred during its execution.

Response
Switch explicitly to old-style operation, analyse and test the program.

DRI0481 ILLEGAL (&00) IN A CALLED OLD-STYLE PROGRAM

Meaning
A new-style CALL started an old-style program containing the specified illegal statement.
(&00): DO, STOP

DRI0488 EXISTING DYNAMIC TEMPORARY VIEWS RELEASED

Meaning
Before it is possible to exit from a program called with DO, there must be no more temporary views which have been defined in program mode. In other words a dynamic DROP TEMPORARY VIEWS statement must have been processed before the END PROCEDURE, STOP or a follow-up DO statement are encountered.
However, if temporary program views still exist, DRIVE deletes them from the database system and outputs this message.

Response
Enter DROP TEMPORARY VIEWS in interactive mode and then enter suitable DROP TEMPORARY VIEWS statements in the programs containing temporary views which are affected.

DRI0489 ITEM-SPECIFITION REQUIRED WITH PRECISELY ONE ITEM

Meaning
It is necessary to specify ‘ITEM’ when the SET SCREEN ATTRIBUTE PRESELECT statement is used for single selection fields. However, only one item may be specified.

Response
If there is no item specification, enter one.
If more than one item is specified, remove items.

DRI0490 SESAMSQL COMPILATION NOT POSSIBLE WITH TRANSACTION OPEN

Meaning
IT has been attempted to compile a program containing SQL statements other than COMMIT/ROLLBACK even though a transaction is open.

Response
Compile the program separately and execute the CALL and CODE member or close the transaction before CALL is encountered.

DRI0491 MAXIMUM NUMBER OF PERMITTED DECLARATIONS EXCEEDED

Meaning
The DRIVE program contains too many declarations of the same object type, e.g. more than 20 DECLARE FILE statements.

Response
Watch limit for number of declarations.

DRI0492 FORM NO 'FHS-DE' FORM

Meaning
This statement requires FHS-DE forms.

Response
Change form or modify form attributes using IFG.

DRI0493 CHANGE OF FORM TYPE NOT PERMITTED

Meaning
All partial forms must be of the same type. It is not permitted to mix FHS-DE forms and non-DE forms.

Response
Change form or modify form attributes using IFG.

DRI0494 ONLY POSITIVE VALUES PERMITTED FOR (&00)

Meaning
A negative value or 0 was specified for ITEM or LINES.
(&00): ITEM or LINES entry

Response
Specify positive value for ITEM or LINES.

DRI0495 (&00) EXCEEDS '(&01)'

Meaning
(&00): LINES: The LINES value in the SET SCREEN ATTRIBUTE LINES statement exceeds the number of lines defined for the list.
ITEM:
- The ITEM value in the SET SCREEN ATTRIBUTE PRESELECT ON/OFF ITEM statement exceeds the number of lines defined for the list.
- The ITEM value in the SET SCREEN ATTRIBUTE LOCK/PRESELECT ON/OFF ITEM exceeds the number of selection fields defined for the single selection field.

(&01): List elements in the case of LINES
List elements or selection fields in the case of ITEM

Response
Check LINES or ITEM specification.

DRI0496 NEXT WITHOUT PRECEDING FIRST

Meaning
A GET NEXT MODIFIED INDEX was entered for a form for which not previous GET FIRST MODIFIED INDEX has been entered.

Response
- Check if form is correct.
- Execute GET FIRST MODIFIED INDEX first.

DRI0497 ENTER NOT MORE THAN 4 CHARACTERS FOR SCROLL

DRI0498 INVALID CHARACTER IN SCROLL ENTRY

Meaning
The SCROLL specification contains characters other than '<', '>', '+', '-' or blank.

Response
Enter no characters other than '<', '>', '+', '-' or blank for SCROLL.

DRI0499 ACCESS TO FILE NOT POSSIBLE

Meaning
Incompatible OPEN mode prohibits access to file.

Response
Open file in appropriate mode.

DRI0500 PLEASE EXIT DRIVE BEFORE YOU EXIT WINDOWS

Meaning
A DRIVE application is still active. You must close this application before you can exit Windows.

DRI0501 (&00)(&01)

Meaning
An error occurred when defining a report according to the message text.
(&00): report error number, see corresponding DRIVE error number
(&01): message text

Response
Correct definition.

DRI0502 RESOURCE NOT DEFINED

Response
Specify the name of the resource file either using the OPTION or the DECLARE WINDOW statement.

DRI0503 'DRI#.' MUST NOT BE USED AS PREFIX IN FILE NAMES

Meaning
The 'dri#.' prefix is reserved for names internally assigned by DRIVE.

Response
Change file name.

DRI0504 NUMBER OF ATTRIBUTES NOT EQUAL TO NUMBER OF VARIABLES

Meaning
Number of DRIVE-variables not equal to number of WINDOW-/INFORMIX-attributes

Response
Match list of specified attributes to list of specified variables.

DRI0505 OBJECT MUST BE OF TYPE '(&00)'

Meaning
The statement cannot be executed with the specified object.
(&00): - choice list
- input field
- input field/group with input field(s)

DRI0506 ATTRIBUTE NOT PERMITTED HERE

Meaning
For DECLARE WINDOW: the REVERSE ON specification is invalid together with BACKGROUND or FOREGROUND COLOUR.

DRI0507 (&00) ALREADY EXISTS

Meaning
(&00): user label;
A user label already exists for the specified transaction code and USER name.
(&00): USER;
The USER name specified in the PARAMETER statement has already been assigned.

Response
Change name of transaction code or user. Change USER name.
Delete file DRIUSERNAME under /tmp after abnormal termination of DRIVE.

DRI0508 CONSTRAINT IN DD VIOLATED, IDDS-STATUS: (&00), (&01)

Meaning
A cardinality constraint was violated. Possible reason:
- data inconsistent
- installation error
- internal execution error

(&00): error number
(&01): error text

Response
Determine exact cause of the error using the IDDS error number. Check data constellation in the data dictionary. Have ERMS administrator check whether an invalid constraint was defined in the supplied command files.

DRIO509 ENTITY '(&00)' '(&01)' CANNOT BE CREATED

Meaning
The entity already exists in the DD in a partition to which DRIVE has no access.
For entity type CAL, RLS : only name of the called program.
For entity type RES : name shortened to 32 characters.
(&00): entity type
(&01): entity name

Response
Determine the exact error location using the entity type and name. Correct data in data dictionary.

DRIO510 END OF THE RESULT TABLE REACHED OR RESULT TABLE EMPTY

Meaning
INFORMIX query returns no or no further hits.

DRIO511 CHANGE OF DB SYSTEM NOT POSSIBLE WITH TRANSACTION OPEN

Meaning
A change of DB system cannot be effected unless the transaction has been completed.

DRIO512 CALLING 'DRIVE-SPU' NOT PERMITTED

Meaning
It is not permitted to invoke DRIVE-SPU from within DRIVE-SPU.

DRIO513 DRIVE PROCESSES INCOMPATIBLE : '(&00)', '(&01)'

Meaning
Different DRIVE processes were installed which are not compatible.
(&00): version of the sending process
(&01): version of the receiving process

Response
Have system administrator check DRIVE installation and repeat run if required.

DRIO514 ERROR WITH '(&00)' (&01) (&02)

Meaning
An error occurred on running a system macro.
(&00): name of system macro
(&01): error number
(&02): error text

Response
Notify system administrator

DRI0515 START OF PROCESS NOT POSSIBLE : (&00) (&01)

Meaning
Either the number of processes permitted for the system or for the user was exceeded or a memory bottleneck occurred.
(&00): error number
(&01): error text

Response
Consult system administrator or start fewer processes when running DRIVE-SPU.

DRI0516 PROCESS '(&00)' CANNOT BE STARTED DUE TO : (&01)

Meaning
No DRIVE process has been installed for the indicated DRIVE function.
(&00): process name
(&01): cause of error

Response
If the process is required, have system administrator check the DRIVE installation.

DRI0517 INCREMENT VALUE IS 0 OR NULL

Meaning
The increment value of a CYCLE FOR loop must not be 0 or NULL at execution time.

Response
Make valid assignment to increment value.

DRI0518 '(&00)' IS NOT A (&01)

Meaning
(&00): name of file or directory
(&01): file or directory

Response
Delete existing file or change directory and repeat statement; change names.

DRI0519 INPUT FOR (&00) INVALID

Meaning
(&00): directory: No valid directory was specified in the PARAMETER statement.
file: The specified file name does not comply with the SINIX conventions.

Response
Specify valid directory and repeat statement.
Specify valid file name and repeat statement.

DRIO520 DD STATUS MUST NOT BE CHANGED

Meaning
An attempt was made to set the parameter value for the DD to OFF while the object class window for user labels was open.

Response
Close object class window for user labels.

DRIO521 CONTROL VARIABLE USED ON MORE THAN ONE 'CYCLE FOR' LEVEL

Meaning
The control variable for a CYCLE FOR loop must not be used on any other CYCLE FOR level and must not be modified.

Response
Define separate control variable for each CYCLE FOR level.

DRIO522 NUMBER OF PARAMETERS IN SENDING AND RECEIVING SCRIPT NOT EQUAL

Meaning
The number of parameters in the sending script and receiving script must match.

DRIO523 (&OO). INCOMPATIBLE PARAMETER IN SENDING AND RECEIVING SCRIPT

Meaning
The data type of the sending script is not compatible with the receiving script.
(&00): Position of incompatible parameter.

DRIO524 RETURN SPECIFICATION IN (&OO). PARAMETER DOES NOT MATCH

Meaning
Adapt RETURN specification, i.e. the RETURN specifications for the indicated parameters in the sending script must match those in the receiving script.
(&00): Position of mismatching parameter

DRIO525 OPTION '(&OO)' WITH '(&O1)' NOT COMPATIBLE

Meaning
A statement or statement part is not compatible with the assigned DB system.

DRIO526 CURRENT OR FORMAL PARAMETERS ARE NOT COMPATIBLE

Meaning
The data type of the formal parameter cannot be assigned to the current parameter.

Response
Adapt the formal parameter to the current interface definition.

DRIO527 REMOTE ACCESS NOT POSSIBLE IN FIRST DIALOG STEP; ENTER 'DUE'

Meaning
Remote access is not possible before first screen output.

Response
Restart program.

DRIO528 PATHNAME OF PROGRAM TOO LONG

Meaning
The pathname assigned at installation is too long.

Response
At DRIVE installation, make sure that pathname is shorter than 254 characters.

DRIO529 INSUFFICIENT BUFFER LENGTH

Meaning
if the dialog builder fills the DRIVE buffer, the buffer must not be dynamically extensible.

Response
Extend buffer and restart DRIVE.

DRIO530 EVENT-OBJECT COMBINATION NOT UNIQUE

Meaning
A script contains duplicate ON conditions.

Response
Use unique ON conditions only.

DRIO531 NAME OF DIRECTORY TOO LONG

Meaning
The name specified using the $DRIVE_DD environment variable exceeds 10 characters.

Response
Use names of up to 10 characters.

DRIO532 INVALID EXPONENTIAL MASK

Meaning
The exponential mask for a numeric data type has been incorrectly defined. The following applies:
en, where n > 8 and n < 25

Response
Define mask accordingly.

DRIO533 ERROR ON INTERNAL CHANGE/CLOSE OF INFORMIX DATABASE

Meaning
The current database is still active. See further message output for details.

DRIO534 RUNTIME ERROR '(&00)' WHEN CALLING '(&01)'

Meaning
A PASCAL runtime error occurred which was caused by an interface or coding error in the subroutine or subprogram of another programming language.
(&00): runtime error
(&01): module name

Response
- refer to describtion of PASCAL runtime errors in the PASCAL User Guide under "Runtime errors and error handling".
- Remove runtime error by remedying error in external subprogram or subroutine.

DRIO535 SCRIPT DECLARATION FOR '(&00)' MISSING

Meaning
The PROC statement contains a SCRIPT clause which has not been declared.
(&00): window name.

Response
Remove SCRIPT clause from PROC statement or include SCRIPT declaration.

DRIO536 (&00)(&01)(&02)

Meaning
An error occurred when issuing an SQL statement to the INFORMIX database system The message text has the following format:
<INFORMIX error message number>[<(C-ISAM error code)>]
<INFORMIX error message text, including up to one insert>.

Response
Refer to the "INFORMIX Error Messages" manual for an explanation and "Corrective action" for each INFORMIX error message.

DRIO537 '(&00)' ENVIRONMENT VARIABLE INVALID OR NOT SUPPLIED

Meaning
The environment variables "$SQLEXEC" and "$INFORMIXDIR" for INFORMIX are invalid or have not been supplied.
(&00): INFORMIX

Response
Correct environment variable

DRIO538 (&00) NOT PERMITTED IN (&01) PROGRAM BLOCK

Meaning
The described specification is not permitted in the described program block.

Response
Correct or omit statement.

DRI0539 PROCESS '(&00)' NO LONGER EXISTS

Meaning
The described process has already been terminated. The current statement was not executed. This applies to one of the following process types:
(&00): dri_mo: monitor process
dri_op: interface process
dri_dd: DD process
dri_dz: DD process for user labels
The SPE is terminated if the monitor process has been terminated. If the dri_dz process has been terminated, the class window for user labels is closed.

Response
Do not try to repeat your last statement. You can continue to work with DRIVE however. Determine the reason for process failure.

DRI0540 PLEASE ACKNOWLEDGE

Response
Acknowledge message box with OK.

DRI0541 '(&00)' COMPILED WITHOUT ERROR

Meaning
The described program has been compiled without error.
(&00): program name

DRI0542 '(&00)' DELETED WITHOUT ERROR

Meaning
The described program has been deleted.
(&00): program name with associated objects, i.e. list, intermediate codes, etc., if any

DRI0543 'DATABASE' STATEMENT MISSING FROM UTM START FILE

Meaning
Informix programs can only run as UTM applications if the same database is specified in the start file.

Response
Change UTM start file.

DRI0544 '(&00)' PRECOMPILED WITHOUT ERROR

Meaning
The indicated program has been successfully precompiled using OPTION PRECOMPILE=ONLY.
(&00): program name.
The result file with the suffix '.i' has been created.

DRI0545 START FILE DOES NOT EXIST

Meaning
A start file is mandatory
- when DRIVE is running in batch mode
- when DRIVE is running as RTS version

Response
Generate start file

DRI0546 ERROR IN START FILE

Meaning
The start file contains error(s).

Response
Correct start file

DRI0547 'DRIVE' TERMINATES WITH 'EXIT'

Meaning
The program to be run under DRIVE RTS contains an EXIT statement.

Response
If exit not required, eliminate EXIT statement.

DRI0548 'DRIVE' TERMINATES WITH 'BREAK'

Meaning
The program to be run under DRIVE RTS contains a BREAK statement.

Response
If break not required, eliminate BREAK statement.

DRI0549 STATEMENT ABORTED (EXIT=DRIVE TERMINATES)

Meaning
Error(s) in program analysis or execution.

Response
Acknowledge EXIT prompt with <ENTER> to allow DRIVE to terminate normally.

DRI0550 STATEMENT ILLEGAL UNDER DRIVE RTS

Meaning
If the program is to run under DRIVE RTS, the start file may only contain the DO statement. The only response permitted to the prompt is EXIT.

Response
Eliminate invalid statement; acknowledge EXIT prompt.

DRI0551 DRIVE FORM FILE COPY ERROR

Meaning
Copying the DRIVE form file 'D@<usern>' to the save file or copying the save file 'D@<usern>.sav' to the corresponding DRIVE form file failed. Possible reasons: memory bottleneck, insufficient access authorization

Response
Check access permission for DRIVE form files as well as their save copies ('read + write' required), or make available sufficient memory.

DRI0552 SAME EXECUTION ERROR OCCURS TWICE

Meaning
An error occurred during execution of a DISPLAY LIST statement. An error list is to be written in response to the error, during which the same error occurs again.

Response
Remove cause of first error and start program again.

DRI0553 PLEASE ENTER DEBUG STATEMENT

Meaning
DRIVE is in debugging mode and expects you to enter a debug statement.

Response
Enter debug statement.

DRI0554 DEBUG STATEMENT EXECUTED

Meaning
DRIVE has executed the last debug statement without error and expects the next debug statement to be entered.

Response
Enter next debug statement.

DRI0555 LINE NUMBER (&00) INVALID

Meaning
An invalid line number was specified in an AT statement.
(&00): line number

Response
Determine valid line number and correct AT statement.

DRIO556 LINE NUMBER RANGE '(&00)–(&01)' INVALID

Meaning
An invalid line number range was specified in an AT statement.
(&00): start of range
(&01): end of range

Response
Determine valid line number range and correct AT statement.

DRIO557 ALPHA OUTPUTS NOT POSSIBLE WITHOUT FORMANT

Meaning
DRIVE was started with the '-w' option. This implies graphics output, but no alpha outputs.
Note: The alpha output to be effected may be due to an error in the start file.

Response
Do not execute DRIVE programs with alpha outputs when starting DRIVE with '-w'.

DRIO558 FILE: '(&00)' ERROR: (&01)

Meaning
An error occured when processing the indicated file.
(&00): file name
(&01): error message of the operating system.

Response
Analyze indicated error.

DRIO559 '(&00)' NOT FOUND: (&01)

Meaning
The indicated file does not exist.
(&00): file name
(&01): error message of the operating system

Response
Make specified file avialable.

DRIO560 '(&00)' LOCKED: (&01)

Meaning
The indicated file cannot be accessed.
(&00): file name
(&01): error message of the operating system.

Response
Provide access to indicated file.

DRIO561 PLEASE SUPPLY USING PARAMETERS

Meaning
The initial breakpoint has been reached and not all USING parameters have been supplied.

Response
Supply USING parameters using SET or abort debugging session using BREAK DEBUG.

DRIO562 (&00) USING PARAMETERS NOT YET SUPPLIED

Meaning
At the initial breakpoint, the indicated number of USING parameters are missing.
(&00): number

Response
Supply missing USING parameters using SET statements or abort debugging session using BREAK DEBUG.

DRIO563 USING PARAMETERS COMPLETE

Meaning
At the initial breakpoint, all USING parameters have been supplied.

Response
Enter desired debug statement.

DRIO564 STATEMENT INVALID FOR PARAMETER PROMPTING

Meaning
A statement other than SET or BREAK DEBUG has been entered in response to parameter prompting.

Response
Terminate parameter prompting.

DRIO565 'SET' STATEMENT ONLY PERMITTED FOR USING PARAMETERS OF 'DO/CALL'

Meaning
At the initial breakpoint, not all USING parameters have been supplied, a SET statement was entered for another variable however.

Response
Supply USING parameters or enter BREAK DEBUG.

DRIO566 'SET' STATEMENT NOT PERMITTED FOR VARIABLE VALUE ASSIGNMENT

Meaning
At the initial breakpoint, not all USING parameters have been supplied, a SET statement with variable value assignment was entered however.

Response
Supply USING parameters or enter BREAK DEBUG.

DRI0567 APPLICATION LINKED TO ILLEGAL FILE

Meaning
When starting the DRIVE kernel process from the file manager, the DRIVE kernel process may only be linked to a start file (suffix '.dri') or a code element (suffix '.drx').

Response
Correct illegal linkage.

DRI0568 DEBUGGING OUTPUT TOO LARGE

Meaning
In a trace, a statement is to be output which has too many lines (in the listing element). The excessive lines at the end are suppressed.

Response
Press <ENTER> key

DRI0569 TOO MANY ERROR MESSAGES

Meaning
More errors have occurred than can be output. Output of the excessive last errors is therefore suppressed.

Response
Enter next debug statement.

DRI0570 NO DEBUG STATEMENT FOUND

Meaning
At a breakpoint, only the <ENTER> key was pressed. No debug statement was entered.

Response
Enter valid debug statement.

DRI0571 NUMBER OF ERROR EXCEEDS LIMIT

Meaning
The number of errors found during analysis of a DRIVE program exceeds the entries possible in the error table. Additional errors are not listed.

Response
Correct reported errors and recompile program.

DRI0572 EXTENSION '(&00)' NOT PERMITTED

Meaning
The only permitted extensions for DRIVE programs are DRP and DRX.
(&00): extension

Response
Change name of DRIVE program.

DRIO573 BRANCH TO FINAL BREAKPOINT

Meaning
The system has branched to the final breakpoint after the BREAK statement has been entered, or the BREAK key activated, in debugging mode.

Response
Enter BREAK DEBUG, DISPLAY FORM or DISPLAY LIST.

DRIO574 FINAL BREAKPOINT ALREADY REACHED

Meaning
An illegal statement was entered at the final breakpoint. Permitted are: BREAK DEBUG, DISPLAY FORM and DISPLAY LIST.

Response
Enter valid debug statement.

DRIO575 FINAL BREAKPOINT REACHED

Meaning
Processing has stopped after the END PROCEDURE statement on the highest program level (program called with DEBUG) has been executed.

Response
Enter debug statement.

DRIO576 INITIAL BREAKPOINT REACHED

Meaning
After entering the DEBUG statement in interactive mode, the system branches to the initial breakpoint. Execution has processed the PROCEDURE statement in the program called with DEBUG and stopped after this statement.

Response
Enter debug statement.

DRIO577 INITIAL BREAKPOINT: EXECUTION OF DEBUG STATEMENTS STARTS

Meaning
The debugger was started from the SPE. The debug statements entered were executed. This is followed by a debug run with or without trace, depending on whether a TRACE statement was entered or not in the SPE.

DRI0578 CURRENT BREAKPOINT: LINE (&00) IN PROCEDURE '(&01)'

Meaning
The indicated breakpoint has been reached, i. e. program execution has reached a point just before the statement in the specified line of the specified procedure. Any debug operations of the type DISPLAY or SET which have been included at this statement have already been executed.
(&00): line number
(&01): procedure name (incl. library specification)

Response
Enter debug statement.

DRI0579 ERROR ON EXECUTING STATEMENT '(&00)'

Meaning
An execution error occurred when executing a program statement or a debug operation included at this point. The associated line number and program name are output in message number DRI0578. Also other relevant error messages are output prior to message DRI0579 (as in the error list in program mode).
(&00): DRIVE statement

Response
Try to remedy the error using a SET statement. If this fails, the debug run has to be aborted with BREAK DEBUG.

DRI0580 ENTRIES TOO LONG

Meaning
The record exceeds the input list. The excessive characters are ignored.

Response
Extend input list.

DRI0581 FILE OUTPUT TOO LONG

Meaning
Length of 32000 exceeded when outputting to a file.

Response
Distribute output across several output statements.

DRI0582 NULL VALUE NOT DEFINED

Meaning
A null value is to be written to a file whereas no null value has been declared.

Response
Specify a character for the representation of the null value when defining a file.

DRI0583 FILE '(&00)' NOT YET OPEN

Meaning
The indicated file has not yet been opened by the program.
(&00): file name

Response
Open file before calling the program.

DRI0584 FILE '(&00)' ALREADY OPEN

Meaning
The indicated file has already been opened by the program.
(&00): file name.

Response
Close indicated file before calling the program.

DRI0585 TOO MANY FILES OPEN

Meaning
More than 20 'isam-shareupd/input' files open in BS2000 UTM mode.

Response
Reduce number of open files.

DRI0586 END OF FILE

Meaning
The end of file has been reached

DRI0587 INPUTS TO SHORT

Meaning
The record is shorter than the input list.

DRI0588 NAME '(&00)' TOO LONG

Meaning
The name of the library, classs list, or element is too long.
(&00): too long element

DRI0589 TOO MANY BOXES TO BE REMOVED

Meaning
More boxes than are currently on the screen are to be REPLACEd or REMOVEd.

Response
Decrease the value of 'number' in the REPLACE or REMOVE statement.

DRI0590 OPERATION '(&00)' ALREADY INCLUDED AT STATEMENT IN LINE (&01)

Meaning
The specified debug operation has already been included at the specified line number. It cannot be included more than once.
(&00): debug operation
(&01): line number

Response
Enter next debug statement.

DRI0591 NO OPERATION FOUND

Meaning
Following a REMOVE statement, no debug operation was found.

Response
Enter next debug statement.

DRI0592 ASSIGNMENT TO INDIVIDUAL COMPONENT NOT PERMITTED

Meaning
During parameter prompting in debugging mode, making assignments to individual components of a USING parameter is not permitted.

Response
Assignments to structured variables in debugging mode using parameter prompting have to be made via aggregates.

DRI0593 TRACE RUN COMPLETED: LINE (&00) OF PROCEDURE '(&01)'

Meaning
The specified number of steps was executed in a trace operation on the screen. Following execution of the last statement, the system passes to a breakpoint, i. e. program execution has reached a point just after the statement in the specified line of the specified procedure. A debug operation of the type COUNT that has been included at this statement has already been executed.
(&00): line number
(&01): procedure name (incl. library specification)

Response
Enter debug statement

DRIO594 STATEMENT ONLY PERMITTED FOR VARIABLE CURSOR

Meaning
The statement (e.g. DROP) is only permitted if the cursor has been declared as variable cursor (i.e. without FOR SELECT ... clause).

Response
- Correct cursor declaration
- Remove DROP or execute with EXEC.

DRIO595 CHANGE OF DIALOG STEP DURING 'REPORT' PROCESSING UNDER UTM

Meaning
A screen output is to be performed under UTM and at least one REPORT is vet to be filled.

Response
Check program execution. Terminate REPORT processing prior to screen output using CLOSE REPORT.

DRIO596 '(&00)' CAN ONLY BE EXECUTED ON THE HIGHEST PROGRAM LEVEL

Meaning
A program can only be executed either
- with the DO statement, or
- in the receiving environment, on the program level directly called by the submitting partner.

A program can only be executed with DO if it contains e. g. a STOP statement
(&00): program name without suffix: source program
program name with suffix CODE: intermediate code

DRIO597 STATEMENT NOT PERMITTED FOR PREFETCH CURSOR

Meaning
The statement (e.g. FETCH PRIOR) is not permitted if the cursor has been declared as PREFETCH cursor.

Response
Correct cursor declaration.

DRIO598 RUNNING PROGRAM ABORTED

Meaning
The user terminated the running program by activating the 'Cancel' menu item in the SPU for example.

DRI7001 No memory space is available.

DRI7002 System error!

DRI7003 Internal error!

DRI7004 Report Services is not available.

DRI7005 No access to message file.

DRI7006 The given node Id does not reference a suitable report object.

DRI7007 The given pointer does not reference an opened report object.

DRI7008 Actual input parameters are missing.

DRI7009 Incorrect length of actual parameter record.

DRI7010 No memory space is available.

DRI7011 System error (function: (&00) errno: (&01)).

DRI7012 Internal error in line (&00).

DRI7013 An LMS error has occurred (LMS: (&00) DMS: (&01)).

DRI7014 An internal PMC error has occurred.

DRI7015 An error with NLS constants has occurred.

DRI7016 The version of the report object is not supported.

DRI7017 One of the given arguments has an inappropriate value.

DRI7018 The delimiters in the users profile are not unique.

DRI7019 The user function is not defined.

DRI7020 NOT YET IMPLEMENTED!

DRI7021 The function is not allowed in the current open mode.

DRI7022 The report object to be created already exists.

DRI7023 The report object to be opened is locked.

DRI7024 No access to report object or data file.

DRI7025 The meta type (&00) is not allowed in this context.

DRI7026 The meta type (&00) is already defined.

DRI7027 The property (&00) is not allowed in this context.

DRI7028 The property (&00) is already set.

DRI7029 (&00) is invalid for property (&01).

DRI7030 Property sequence error for (&00).

DRI7031 No name is defined for a superior structure.

DRI7032 The group field cannot be found in the order clause.

DRI7033 The to be searched node or property cannot be found.

DRI7034 The report object does not contain a valid record part.

DRI7035 The report object already contains a layout part.

DRI7036 The provided buffer is too short to take up the property value.

DRI7037 The symbol (&00) is not valid.

DRI7038 The symbol (&00) is ambiguous.

DRI7039 An array index must be a constant value or a local variable.

DRI7040 An array index is out of range.

DRI7041 A mandatory node as son of (&00) is missing.

DRI7042 Property (&00) for node (&01) is missing.

DRI7043 The node (&00) has no properties defined.

DRI7044 The number of detail parts and record descriptions is different.

DRI7045 The referenced report object contains no layout part.

DRI7046 A mandatory node representing a record part is missing.

DRI7047 Double (&00) with same hierarchy is defined.

DRI7048 The redefinition exceeds the length of the redefined field.

DRI7049 An error has occurred while validating an expression.

DRI7050 Type conflict in expression.

DRI7051 Wrong group number (&00) in group header and/or group trailer.

DRI7052 A Parameter for the user function is not defined.

DRI7053 Type of an array index not allowed.

DRI7054 Control block (&00) only allowed once.

DRI7055 This feature is not allowed in the page base.

DRI7056 Input data exceeds the maximum record length.

DRI7057 The format description string is incorrect.

DRI7058 For this field NULL_ALLOWED is not defined.

DRI7059 The given length of the data record is wrong.

DRI7060 The record kind is undefined.

DRI7061 The length of the given field is not correct.

DRI7062 The data type of the given field is incorrect.

DRI7063 Error while calling an INFORMIX conversion function.

DRI7064 Error while calling an INFORMIX arithmetic function.

DRI7065 Not enough space for printing in detail area.

DRI7066 The result of the condition is NULL.

DRI7067 Not enough header lines for page header are defined.

DRI7068 Not enough trailer lines for page trailer are defined.

DRI7069 No tabulators are defined for this control block.

DRI7070 Type incompatibility in assign statement.

DRI7071 The data type of an user function argument is wrong.

DRI7072 The parameter of the user function is not declared.

DRI7073 Division by zero is not allowed.

DRI7074 The numeric string constant represents no number.

DRI7075 The value is out of range in ASSIGN statement.

DRI7076 Error in user function: (&00).

DRI7077 Length of image data is wrong.

DRI7078 Image data is not in hexadecimal form.

DRI7079 No device profile is available.

DRI7080 An RDI file with the given path name cannot be opened.

DRI7081 The spool cannot be activated.

DRI7082 The device profile is not defined completely.

DRI7083 The name for the RDI converter is missing in the device profile.

DRI7084 Conversion error in the device profile (line (&00)).

DRI7085 Invalid character in the device profile (line (&00)).

DRI7086 Incomplete line in the device profile (line (&00)).

DRI7087 Incorrect line in the device profile (line (&00)).

DRI7088 Incorrect value in the device profile (line (&00)).

DRI7089 Value to long in the device profile (line (&00)).

DRI7090 Wrong initial string in the device profile (line (&00)).

DRI7091 Incorrect meta word class in the device profile (line (&00)).

DRI7092 Missing command line in the device profile (line (&00)).

DRI7093 Missing command line in the device profile (line (&00)).

DRI7094 Missing command line in the device profile (line (&00)).

DRI7095 Wrong control word sequence in the device profile (line (&00)).

DRI7096 Missing "valid value"-line in the device profile (line (&00)).

DRI7097 Wrong "valid value"-list in the device profile (line (&00)).

DRI7098 Missing define-connected-line in the device profile (line (&00)).

DRI7099 Wrong connected-value-line in the device profile (line (&00)).

DRI7100 Wrong connected-value-line in the device profile (line (&00)).

DRI7101 Wrong increment value in the device profile (line (&00)).

DRI7102 Metaword in device profile is not defined (line (&00)).

DRI7103 Only "CHARTYPE" in device profile allowed (line (&00)).

DRI7104 Wrong parametertype in the device profile (line (&00)).

DRI7105 Defaultvalue in the device profile is not defined (line (&00)).

DRI7106 Substitutevalue in the device profile is not allow (line (&00)).

DRI7107 An error has been returned by the spool system.

DRI7108 Source and target are equal.

DRI7109 Incorrect parameter indicators in the device profile (line (&00)).

DRI7110 Too many parameter indicators in the device profile (line (&00)).

DRI7111 Only "NUMTYPE" in device profile allowed (line (&00)).

DRI7112 Using false device profile or RDI-Converter.

DRI7113 The RDI input file (&00) cannot be opened.

DRI7114 Bad magic number in RDI input stream.

DRI7115 Usage: (&00) ... (see manual)!

DRI7116 No prolog file is available.

DRI7117 Source file has no bounding box.

DRI7118 No user specific and no system specific user profile can be found.

DRI7119 Syntax error in user profile (command "(&00)).

DRI7120 Functionality not available in this release ((&00)).

DRI7121 Data structure skipped in standard layout.

DRI7122 Too many actual parameters are given.

DRI7123 Wrong data type for repeat value of a constant string.

DRI7124 The file defined with a source statement does not exist.

DRI7125 Format description on page (&00) line (&01) is too short.

DRI7126 Printing position page (&00) line (&01) not allowed.

DRI7127 No tabulator on page (&00) line (&01) left.

DRI7128 Rotation not supported by device.

DRI7129 An implicit sort order has been introduced.

DRI7130 No text – never used.

DRI8000 MONDAY/TUESDAY/WEDNESDAY/THURSDAY/FRIDAY/SATURDAY/SUNDAY/

Meaning
Day names that can be used as day strings in masks. Maximum string length: 40 bytes

DRI8001 JANUARY/FEBRUARY/MARCH/APRIL/MAY/JUNE/JULY/AUGUST/SEPTEMBER/OCTOBER/NOVEMBER/DECEMBER/

Meaning
Month names used as month strings in masks. Maximum length per string: 40 bytes

DRI8002 CURRENT CONVERSATION LIST FOR USER '(&00)' WITH CONVERSATION STATUS '(&01)'

Meaning
List header for UTM print outputs.

DRI8003 LIST OF ALL CONVERSATIONS FOR USER '(&00)' WITH CONVERSATION STATUS '(&01)'

Meaning
List header for UTM print outputs.

DRI8004 CONVERSATION DATE: (&00) CONVERSATION TIME: (&01)

DRI8005 LIST PRINTOUT TERMINATED AT: (&00)/(&01)

Meaning
(&00): date
(&01): time

DRI8010 (NEXT/PRIOR/FIRST/LAST/BREAK=SCROLL;+/-/+-/++/--=PAGING IN RECORD)

Meaning

SCROLL = positioning within the cursor table

- NEXT: read the next cursor record
- PRIOR: read the preceding cursor record
- FIRST: read the first cursor record
- LAST: read the last cursor record
- BREAK: read no more cursor records

Paging in the record:

- \+ : display next screen
- \- : display preceding screen
- +- : redisplay current screen
- ++ : display end of record
- -- : display beginning of record

DRI8011 (NEXT/BREAK=SCROLL;+/-/+-/++/--=PAGING IN RECORD)

Meaning

SCROLL = positioning with the cursor table

- NEXT: read the next cursor record
- BREAK: read no more cursor records

Paging in the record:

- \+ : display next screen
- \- : display preceding screen
- +- : redisplay current screen
- ++ : display end of record
- -- : display beginning of record

DRI8012 (BREAK=ABORT;+/-/+-/++/--=PAGING IN RECORD)

Meaning

Paging in the record:

- \+ : display next screen
- \- : display preceding screen
- +- : redisplay current screen
- ++ : display end of record
- -- : display beginning of record

DRI8013 (NEXT/PRIOR/FIRST/LAST/CURRENT/ABSOLUTE/RELATIVE/BREAK=SCROLL)

Meaning
SCROLL = Navigate within cursor set.
NEXT: Read next cursor row.
PRIOR: Read prior cursor row.
FIRST: Read first cursor row.
LAST: Read last cursor row.
CURRENT: Read current cursor row again.
RELATIVE: Relative positioning with row number which is subsequently requested.
ABSOLUTE: Absolute positioning with row number which is subsequently requested.
BREAK: Stop reading cursor rows.

DRI8014 (ROWNO)

Meaning
Continuation of number 8013 giving the input for the RELATIVE or ABSOLUTE entry

Response
Enter row number for FETCH RELATIVE/ABSOLUTE.

DRI8020 PAGE/LINE/SOURCE/NEST/BY COMMAND/BY DEFAULT/IN SOURCE CODE/COMPILATION OPTIONS/ NUMBER OF ERRORS/REFERENCE LIST/PROGRAM NAME/MEMBER/IDP/STARTUP/PROGRAM/

Meaning
Entries for the compiler listing. Maximum string length: 40 bytes

DRI8021 LITERAL/ATTRIBUTE SPECIFICATION/PREDICATE/USER TEXT/

Meaning
Maximum string length: 40 bytes

DRI8022 NO/DATE/TIME/USER/FROM/ERROR LIST/ERROR INFORMATION/CALLED BY/DYNAMIC CHAINED PROGRAM CALL/END OF ERROR LIST/DISTANCE/LIBRARY TABLE/

Meaning
Maximum string length: 40 bytes

DRI8023 END OF REFERENCE LIST/OF/OF TYPE/AT/PREDEFINED IN DD/BASE TABLE/BASE TABLES/ COLUMN/FILE/VARIABLE/FORM/CALL MOD/LIBRARY/PROGRAM/

Meaning
Entries for UREF/XREF Maximum string length: 40 bytes

DRI8024 EDT FILE O/PROGRAM IN EDT FILE O/SYSTEM LIBRARY/TASK LIBRARY/DATA DICTIONARY/ VECTOR/MATRIX/PROGRAM ERROR/

Meaning
Maximum string length: 40 bytes

DRI8025 FORM INPUT/FORM OUTPUT/STATEMENTS/CONSTANTS/VALUES/INTERMEDIATE CODE/NUMBER/NAME/SIZE/TOTAL/

Meaning
Entries for size of tables. Maximum string length: 40 bytes

DRI8026 DIRECTORY/OBJECT/USERLABEL/COPY-ELEMENT/CONSTANT/WINDOW-OBJECT/IN/DEBUG-LIST/

DRI8027 DIALOG/ALPHA/GRAPHIC/UTM-ENTER/MASK/CHECK/WINDOW-ATTRIBUTE/

DRI8028 PROCESS CANNOT BE STARTED/EDITOR CANNOT BE STARTED/CHANGE OF DIRECTORY NOT POSSIBLE/DIRECTORY CANNOT BE GENERATED/PATHNAME TOO LONG/PATHNAME INCORRECT/TILDE CANNOT BE INTERPRETED/

Meaning
Inserts included in the message issued by the editors.

Response
if inserts are to be modified, the maximum string length per insert is 40 bytes.

DRI8029 COMPUTING ERROR/IN ARGUMENT/BATCH/SERVER/CLIENT/BEGIN/END/

Meaning
Inserts included in the message issued by the component processing expressions.

Response
If inserts are to be modified, the maximum string length is 40 characters.

DRI8030 VERSION/DIRECTIVE/CALL KEY/TOTAL LENGTH/LENGTH/LIBRARY NAME LENGTH/MEMBER NAME LENGTH/LEVEL/COUNTER/

Meaning
Inserts for messages.

DRI8100 FORWARD

Meaning
Output made by DRIVE system program.

DRI8101 BACKWARD

Meaning
Output made by DRIVE system program.

DRI8102 CANCEL

Meaning
Output made by DRIVE system program.

DRI8103 SELECT PARAMETER STATEMENT

Meaning
Output made by DRIVE system program.

DRI8104 TRACE OPTION

Meaning
Output made by DRIVE system program.

DRI8105 DEFINE DYNAMIC PARAMETERS

Meaning
Output made by DRIVE system program.

DRI8106 ASSIGN K- OR F-KEYS

Meaning
Output made by DRIVE system program.

DRI8107 DEFINE STATIC PARAMETERS

Meaning
Output made by DRIVE system program.

DRI8108 LOCK STATEMENTS

Meaning
Output made by DRIVE system program.

DRI8109 PROCESSING

Meaning
Output made by DRIVE system program.

DRI8110 NO ACTION

Meaning
Output made by DRIVE system program.

DRI8111 C

Meaning
Output made by DRIVE system program.

DRI8112 P

Meaning
Output made by DRIVE system program.

DRI8200 /********* DECLARATION OF WINDOWS PROCESSED *********/

DRI8201 /************** EVENT-DRIVEN PROCESSING **************/

DRI8202 /**************** INITIALIZATION BLOCK ****************/

DRI8203 /********************* PROGRAM BODY *********************/

DRI8204 /*********** ERROR ROUTINE FOR WINDOW ERROR ************/

DRI8205 <Nothing found>

Meaning
No object matching the choice list in a class window was found. The list is currently locked. The functions of the 'Object' menu are also locked, except the 'New' and 'Close' functions.

Response
Continue processing with one of the available functions, i.e. close the window, change the directory, create a new user label or source.

DRI8206 Abort running processes? (Yes/No)

Meaning
Closing the class window would at the same time abort any processes that are still running.

DRI8207 Yes

Meaning
Entry option in response to previous message

DRI8208 No

Meaning
Entry option in response to previous message.

DRI8209 <No child directory>

Meaning
The current directory does not contain a child directory. A change of directory is only possible to a parent directory.

DRI8210 <No active activities>

Meaning
No background activities are currently running, e.g. compilations, program executions, etc.

DRI8211 Path name/Library specification

Meaning
Label for dialog box.

DRI8212 New object name:

Meaning
Label for dialog box.

DRI8213 Old object name:

Meaning
Label for dialog box.

DRI8214 Select at least two objects

Meaning
When linking a load module, at least two objects must be selected from the object list.

DRI9000 STATEMENT EXECUTED SUCCESSFULLY

DRI9010 RECORD LOCKED BY FOREIGN TRANSACTION

DRI9021 INVALID USER ID

DRI9022 ACCESS RIGHTS VIOLATED

DRI9100 END OF RESULT TABLE REACHED OR RESULT TABLE EMPTY

DRI9116 SCHEMA NAME '(&00)' INCORRECT

DRI9118 SCHEMA NAME '(&00)' AMBIGUOUS

DRI9119 SCHEMA NAME '(&00)' TOO LONG

DRI9121 '(&00)' TABLE NOT SIMPLE

DRI9123 '(&00)' TABLE NOT A BASE TABLE

DRI9126 TABLE NAME '(&00)' INCORRECT

DRI9127 '(&00)' TABLE NOT DEFINED

DRI9128 TABLE NAME '(&00)' AMBIGUOUS

DRI9129 TABLE NAME '(&00)' TOO LONG

DRI9131 COLUMN SPECIFIED INCORRECTLY

DRI9135 ERROR IN COLUMN SPECIFICATION

DRI9136 COLUMN NAME '(&00)' INCORRECT

DRI9137 COLUMN NOT DEFINED

DRI9138 COLUMN NAME AMBIGUOUS

DRI9139 COLUMN NAME TOO LONG

DRI9141 CURSOR IS CLOSED

DRI9142 CURSOR IS OPEN

DRI9143 CURSOR NOT POSITIONED

DRI9144 CURSOR CANNOT BE POSITIONED

DRI9146 CURSOR NAME '(&00)' INCORRECT

DRI9147 CURSOR NOT DEFINED

DRI9148 CURSOR NAME '(&00)' AMBIGUOUS

DRI9149 CURSOR NAME '(&00)' TOO LONG

DRI9156 CORRELATION NAME '(&OO)' INCORRECT

DRI9158 CORRELATION NAME '(&OO)' AMBIGUOUS

DRI9159 CORRELATION NAME '(&OO)' TOO LONG

DRI9210 NULL VALUE CONSTRAINT VIOLATED

DRI9220 UNIQUENESS CONSTRAINT VIOLATED

DRI9230 REFERENTIAL CONSTRAINT VIOLATED

DRI9300 INDICATOR VARIABLE NOT PERMITTED

DRI9310 INDICATOR VARIABLE NOT SPECIFIED

DRI9320 QUERY RETURNS MORE THAN ONE HIT

DRI9330 VALUE LIST INCOMPLETE OR INCORRECT

DRI9335 VALUE INCORRECT

DRI9336 PRIMARY KEY SPECIFIED IN 'SET' CLAUSE

DRI9337 PRIMARY KEY NOT FULLY SPECIFIED

DRI9338 SET FUNCTION NOT PERMITTED

DRI9339 TOO MANY ELEMENTS IN AGGREGATE

DRI9340 VALUE OVERFLOW / UNDERFLOW

DRI9345 INCORRECT DATA TYPE

DRI9350 INCOMPATIBLE DATA TYPE

DRI9360 INCOMPATIBILITY DURING CONVERSION

DRI9365 INCORRECT COLUMN IN SET FUNCTION

DRI9370 INCORRECT PATTERN IN 'LIKE' CLAUSE

DRI9371 ONLY ONE ELEMENT IN 'IN' CLAUSE

DRI9372 DEFAULT VALUE NOT PERMITTED

DRI9380 ERROR IN THE CONDITION

DRI9384 MORE THAN 2 TABLES ADDRESSED

DRI9385 MORE THAN 6 SORT CRITERIA

DRI9390 NO SYNONYMS SPECIFIED FOR COLUMNS IN THE VIEW

DRI9400 ERROR IN 'ORDER BY' CLAUSE

DRI9420 ERROR IN 'GROUP BY' CLAUSE

DRI9440 ERROR IN 'INTO' CLAUSE

DRI9450 RECORD ALREADY DELETED BY THIS TRANSACTION

DRI9500 INCORRECT OBJECT SPECIFICATION IN 'SHOW' STATEMENT
DRI9600 SEQUENCE OF STATEMENTS INCORRECT
DRI9630 POSITIONING SPECIFICATION NOT PERMITTED
DRI9700 SHORT-TERM ACCESS LOCK
DRI9701 ABORTION DUE TO 'CANCEL' OR 'INTR' CALL
DRI9702 ABORTION DUE TO SORT ERROR
DRI9710 SHORT-TERM DB SYSTEM OVERLOAD
DRI9720 DB SYSTEM I/O ERROR
DRI9730 TASK DEADLOCK
DRI9740 CONVERSATION UNKNOWN DUE TO ADMINISTR. INTERVENTION OR BOTTLENECK
DRI9745 CONVERSATION UNKNOWN DUE TO DBMS STARTUP
DRI9750 INSERT STATEMENT NOT PERMITTED FOR SPECIFIED BASE TABLE
DRI9760 UPDATE STATEMENT NOT PERMITTED FOR SPECIFIED BASE TABLE
DRI9770 BASE TABLE ACCESS NOT PERMITTED
DRI9775 DB OPEN ERROR
DRI9780 SCHEMA ACCESS NOT PERMITTED
DRI9785 COLUMN ACCESS NOT PERMITTED
DRI9790 TWO SCHEMAS FROM A DATA BASE ADDRESSED IN A SINGLE TA
DRI9800 ACCESS BRIEFLY NOT PERMITTED
DRI9810 UPDATE ACCESS NOT PERMITTED
DRI9820 DATA BASE SYSTEM TERMINATED NORMALLY BY ADMINISTRATOR
DRI9830 DBH NOT YET OR NO LONGER AVAILABLE
DRI9840 SYSTEM COMPONENT FOR DATA BASE SYSTEM NOT AVAILABLE
DRI9850 NEW TA NOT CURRENTLY PERMITTED DUE TO ADMINISTRATOR INTERVENTION
DRI9860 ERROR IN CONFIGURATION FILE
DRI9900 PROGRAMMING ERROR IN DATA BASE SYSTEM
DRI9910 DATA BASE INTEGRITY VIOLATED OR PROGRAMMING ERROR
DRI9920 DATA BASE SYSTEM LIMITS REACHED
DRI9930 ERROR IN STATEMENT REPRESENTATION
DRI9940 STATEMENT/STATEMENT CLAUSE NOT IMPLEMENTED
DRI9990 OPERATION INCORRECT

6.1 SQL return codes

SQL codes are taken over from the database systems SESAM V1 and UDS and are output by DRIVE/WINDOWS as error messages in the form DRI9xxx. Consequently, SQL code 121 results in the DRIVE message DRI9121.

If an error with SQL code SQL 920 (error in statement representation) or 990 (error in operation) occurs, DRIVE/WINDOWS is aborted.

For the SQL return codes, refer to the SQL language description for the relevant database system.

SESAM V2

For the SQL codes of the SESAM V2 database system, please refer to the User Guide SESAM/SQL-Server: Messages [24]

7 Appendix

Naming conventions

Letters, digits and the underscore character (_) may be entered in names. Uppercase and lowercase letters are not distinguished.

Each name must begin with a letter and may only end in an underscore if it occurs in a UDS database.

Names must conform to the system where they are used:

– INFORMIX names must conform to INFORMIX conventions,

– SESAM names must conform to SESAM conventions,

– UDS names must conform to UDS conventions,

– IFG form names must conform to IFG conventions,

– FHS form names must conform to FHS conventions,

– BS2000 file names must conform to BS2000 conventions,

– SINIX file names must conform to SINIX conventions,

– Microsoft Windows filenames must conform to Microsoft Windows conventions,

– Module names must conform to the conventions of the library system.

Within a DRIVE session, the names of namable objects of the same type must be unique. For example, two procedures with the same name must not be active at the same time.

column

column is used to identify columns of a table (max. 31 characters). It must be unique within a table. Identical names may be used in different tables, in which case the table name must also be specified for unique identification: *table.column*

correlation

correlation is used together with a *query-expression* to define a name for a base table or a view. This correlation is valid only for the duration of the SELECT. The name must not exceed 18 characters in length.

Correlations can especially be used to define shorter or more meaningful names for a table.

cursor

cursor is used to identify a cursor. It must not exceed 18 characters in length.

If *cursor* is assigned by DRIVE/WINDOWS, it may receive the values DRIVE0000000001 through DRIVE999999999999. All cursor names must be unique within a compilation unit.

file-name

In a BS2000 system:

In the PARAMETER DYNAMIC LOGFILE statement this is the filename extension for the dialog log file (max. 20 characters).

In the LIST ... [INTO FILE] statement this is the filename extension for the central print file (max. 20 characters).

In a SINIX system:

Relative or absolute pathname of a file (max. 54 characters).

A relative path name refers to the directory from which DRIVE/WINDOWS was started.

In a Microsoft Windows system:

Relative or absolute pathname of a file (max. 54 characters).

A relative path name refers to the directory from which DRIVE/WINDOWS was started

flib-name

Name of the format library. The name must not exceed 54 characters in length.

form-name

Name of the DRIVE form. It must not exceed 31 characters in length.

library

In a BS2000 system:

Name of a DRIVE library (max. 54 characters). If the name contains special characters, it must be enclosed in double quotes ("). *library* can also be the link name of a DRIVE library (in accordance with BS2000 conventions). DRIVE/WINDOWS interprets *library* first as a link name then as a library name.

The DRIVE library *library* can be preset using the PARAMETER DYNAMIC LIBRARY statement.

In a SINIX system:

Relative or absolute path name (max. 54 characters) of a directory acting as the DRIVE library.

A relative path name refers to the directory from which DRIVE/WINDOWS was started.

library and *member-name* specifications are combined with the *class-name* specification from the PARAMETER DYNAMIC CLASS statement or with the default setting to form the file name *library/class-name/member-name* .

The DRIVE library *library* can be preset using the PARAMETER DYNAMIC LIBRARY statement.

In a Microsoft Windows system:

Relative or absolute path name (max. 54 characters) of a directory acting as the DRIVE library.

A relative path name refers to the directory from which DRIVE/WINDOWS was started.

A drive can be specified with absolute path names.

library and *member-name* specifications are combined with the *class-name* specification from the PARAMETER DYNAMIC CLASS statement or with the default setting to form the file name *library/class-name/member-name* .

The DRIVE library *library* can be preset using the PARAMETER DYNAMIC LIBRARY statement.

The DRIVE library *library* can be preset using the PARAMETER DYNAMIC LIBRARY statement

list-name

Name of the list form. It must not exceed 31 characters in length.

member-name

In a BS2000 system:

Name of a member of a DRIVE library (max. 31 characters).

In a SINIX system:

Name of a file which identifies a member of a DRIVE library (max. 8 characters). In a Microsoft Windows system:

Name of a file which identifies a member of a DRIVE library (max. 8 characters). The filename extension can be up to 4 characters where the first character is a period.

password

The password for the dialog logfile in the PARAMETER DYNAMIC LOGPASSWORD statement (max. 4 characters).

The user ID for a schema of a UDS or SESAM database in the PERMIT ... PASSWORD statement (UDS: max. 48 characters, SESAM max. 3 characters).

path-name

In a SINIX system:

Relative or absolute path name of a file (max. 254 characters).

A relative path name refers to the directory from which DRIVE/WINDOWS was started.

In a Microsoft Windows system:

Relative or absolute path name of a file (max. 254 characters).

A relative path name refers to the directory from which DRIVE/WINDOWS was started.

A drive can be specified with absolute path names.

prog-name

prog-name is used to identify the name of a program. The name must not exceed 31 characters in length. It must not be identical to the member name under which the source program is stored.

Names for external subprograms must comply with the rules of the language in which the program is written. The name must not contain any reserved words from the language in which the subprogram is written nor any DRIVE keywords.

In a SINIX system, the name of the subprogram must correspond exactly to its call name in the skeleton program, e.g. uppercase and lowercase must be taken into account.

The names of programs which access UDS databases must be in uppercase, because DRIVE/WINDOWS assigns a prefix for the program to cursor and view names, and UDS only permits names in uppercase.

schema-name

Name of a schema for a UDS or SESAM database (UDS: max. 30 characters, SESAM: max. 18 characters).

screen-form

Name of an FHS partial form (max. length: 7 characters).

string

String (max. 256 characters), of an alphanumeric data type. *string* must be enclosed in single quotes (').

subprog-name

subprog-name is used to identify an internal subprogram (max. length 31 characters).

table

table is used to identify a base table or view. The name of a base table must not exceed 18 characters in length in a SESAM database, or 30 characters in a UDS database. The name of a view must not exceed 18 characters in length.

table must be specified to uniquely identify columns with identical names from different tables that appear in the same statement.

table identifies a base table or view. If *table* is qualified with [*schema-name*.], it is defining a base table. In UDS, this base table is defined in the SQLU schema name. In SESAM, the schema-name and the table name of the base table must be identical.

If *table* identifies a base table in UDS, it must be specified in the form [*schema-name*.] *base-table*. If *table* identifies a view, *schema-name* must not be specified.

user-group

Name of a user group. It must comply with UDS conventions and may not exceed 8 characters in length.

user-label

A leader program can be started for a DRIVE user with a user label. The user label is composed of the transaction code (TAC) and the UTM user ID (USER). The user label has a maximum of 31 characters.

user-name

Name stored in the system variable &USER. It must not exceed 8 characters in length.

In TIAM applications, *user-name* may be assigned only so long as no SQL statement has been entered. Thereafter, the TSN is entered for *user-name*. USER may be explicitly specified in TIAM applications only.

In UTM applications, &USER is initialized with the name specified in KDCSIGN.

var-name

Name of a simple variable. The variable must be prefixed by"&" and can have up to 32 characters (including the "&").

variable

variable is used to identify a simple variable or a component of a structured variable. The components can be specified with qualified names ("." or "*"). *variable* must be prefixed by "&" and can have up to 32 characters (including the "&").

view

Name of a view (max. 18 characters). All view names must be unique within a compilation unit and distinct from any base table names.

Keywords

The following table lists the reserved keywords and, where applicable, their abbreviations. The words must not be used as names in statements.

Keyword	Abbreviation
$PI	
ABS	
ABSOLUTE	
ACCELERATOR	
ACQUIRE	
ACTION	
ACTIVATE	ACT
ADD	
ALARM	
ALIGNMENT	
ALL	
ALTER	
AND	
ANGLE	
ANSI	
ANY	
APPLICATION	APPL
AS	
ASCENDING	ASC
AT	
ATTRIBUTE	ATTR
AUTHORIZATION	
AVG	
BACKGROUND	
BASE	
BEFORE	
BEGIN	
BETWEEN	

Keyword	Abbreviation
BIN	
BLANK	
BOLD	
BLUE	
BORDER	
BOTH	
BOTTOM	
BOX	
BREAK	
BSSYSTEM	
BS2000	
BTITLE	
BTYPE	
BUFFERED	
BUTTON	
BWIDTH	
BY	
C	
CANCEL	
CAPITAL	
CASE	
CATALOG	
CENTER	
CHARACTER	CHAR
CHARLENGTH	CHARLN
CHARTYPE	
CHECK	
CHOICE	
CLASS	
CLEAR	
CLICK	

Keyword	Abbreviation
CLIPPED	
CLOSE	
CLUSTER	
CM	
COBOL	
CODE	
COLOUR	COLOR
COLUMN	
COLUMNS	
COMBO	
COMMIT	
COMMITTED	
COMPILE	
COMPRESS	
COMTRACE	
CONCAT	
CONNECT	
CONSISTENCY	CONSIS
CONSTANT	
CONSTRAINT	
CONTINUE	CON
COPY	
COPYSOURCE	
COS	
COUNT	
CREATE	CRE
CROSS	
CURRENT	
CURSOR	
CURSORS	
CYAN	

Keyword	Abbreviation
CYCLE	
DATA	
DATABASE	
DATE	
DATETIME	
DAY	
DAYS	
DBA	
DBSERVER	
DBSYSTEM	
DBTRACE	
DBUTRACE	
DCSYSTEM	
DEACTIVATE	
DEBUG	
DECFLOAT	
DECIMAL	DEC
DECIMALSIGN	DECSIGN
DECLARE	DCL
DEFAULT	DEF
DELETE	DEL
DELSTRING	DELSTR
DENSITY	
DESCENDING	DESC
DESELECT	
DETAIL	
DEVICE	
DEVICETABLE	DEVTAB
DIAGNOSIS	DIAG
DIALOG	
DICTIONARY	DD

Keyword	Abbreviation
DIRTY	
DISPATCH	
DISPLAY	
DISTANCE	
DISTINCT	DIST
DISTRIBUTION	DIS
DMSTRACE	
DO	
DOUBLE	
DROP	
DUPLICATES	
DYNAMIC	DYN
EDITABLE	
EDITOR	
EDT	
ELEMENT	
ELSE	
EMPTY	
END	
ENTER	
ERROR	
ERRORATTRIBUTE	ERRATTR
ESCAPE	
EXCLUSIVE	
EXECUTE	EXEC
EXISTS	
EXIT	
EXP	
EXPANSION	
EXPERT	
EXPLAIN	

Keyword	Abbreviation
EXTEND	
EXTENDED	
EXTENT	
EXTERNAL	
FETCH	F
FILE	
FILL	
FILLER	
FILTER	
FIRST	
FIRSTPAGE	
FLOAT	
FLUSH	
FOCUS	
FONT	
FOR	
FOREGROUND	
FOREIGN	
FORM	
FORMAT	
FORMLIB	
FRACTION	
FRACTIONS	
FREE	
FROM	
FULL	
FUNCTION	
GET	
GLOBAL	
GRANT	
GRAPHICEDITOR	

Keyword	Abbreviation
GREEN	
GROUP	
GROUPNUM	
HARDCOPY	HC
HAVING	
HEADER	
HEIGHT	
HELP	
HIGHINTENSITY	HINT
HOLD	
HORIZONTAL	
HOUR	
HOURS	
ICON	
IF	
IMAGE	
IN	
INCH	
INDEX	
INDICATOR	IND
INFORMIX	
INIT	
INNER	
INOUT	
INPUT	
INSERT	INS
INTEGER	INT
INTERVAL	IV
INTO	
INTTRACE	
INVALID	

Keyword	Abbreviation
INVERSE	
INVISIBLE	INVIS
IOTRACE	
IS	
ISAM	
ISOLATION	
ITALIC	
ITEM	
ITEMS	
JOIN	
KEY	
KFKEY	
LANDSCAPE	
LAST	
LASTPAGE	
LAYOUT	
LEASY	
LEFT	
LENGTH	
LETTERS	
LEVEL	
LG	
LIBRARY	LIB
LIKE	
LINE	
LINES	
LIST	
LISTING	
LISTTYPE	
LN	
LOCATE	

Keyword	Abbreviation
LOCK	
LOG	
LOGFILE	
LOGPASSWORD	LOGPSW
LOWERSTRING	
LTERM	
LTYPE	
LWIDTH	
MAGENTA	
MANAGE	
MANDATORY	
MARGIN	
MASK	
MATCHES	
MAX	
MEMORY	MEM
MEMTRACE	
MESSAGE	MSG
MIN	
MINIMUM	
MINUTE	
MINUTES	
MNEMONIC	
MODE	
MODIFY	
MODULE	
MODULO	
MONEY	
MONINFO	
MONTH	
MONTHS	

Keyword	Abbreviation
MOVE	
MSGSTRING	MSGSTR
MUST	
NAMES	
NATIONAL	
NEED	
NEW	
NEWLINE	NL
NEWPAGE	NP
NEXT	
NOCHECK	
NOCOLOUR	NOCOLOR
NOCURSOR	NOCURS
NOINIT	
NOINVERS	
NORMALINPUT	NORMIN
NORMALINTENSITY	
NORMSQL	
NOT	
NOUNDERLINE	NOUL
NULL	
NULLVALUE	
NUMBER	
NUMERIC	NUM
NUMFLOAT	
NUMTYPE	
OBJECT	
OBJECTNAME	
OF	
OFF	
OK	

Keyword	Abbreviation
OLDSTYLE	
ON	
ONLY	
OPEN	
OPTION	
OR	
ORDER	
OUT	
OUTER	
OUTIN	
OUTPUT	
OVERLAY	
PAGE	
PAPER	
PARAMETER	PAR
PARENT	
PASSWORD	PSW
PATTERN	
PERMANENT	PERM
PERMISSION	
PERMIT	
PIXMAP	
PORTRAIT	
POSITION	
POSITIONED	
POTMUST	
PRAGMA	
PRECISION	
PRECOMOPT	
PRECOMPILE	
PREFETCH	

Keyword	Abbreviation
PRESELECT	
PRESSED	
PRIMARY	
PRINT	
PRIOR	
PRIVILEGES	
PROCEDURE	PROC
PROMPT	
PROPORTIONAL	
PUBLIC	
PROTECTED	PROT
PUBLIC	
PUT	
READ	
REAL	
RECORD	
RED	
REDEFINES	REDEF
REFERENCES	
RELATIVE	
REMOTE	
REMOVE	
RENAME	
REPEAT	R
REPEATABLE	
REPLACE	
REPORT	
RESET	
RESOLUTION	
RESOURCE	
RESOURCELIB	

Keyword	Abbreviation
REST	
RESTART	
RESTORE	
RESULT	
RETURN	RET
REVERSE	
REVOKE	
RIGHT	
ROLLBACK	
ROLLFORWARD	
ROTATION	
ROUND	
ROW	
ROWCOL	
ROWID	
ROWS	
SAM	
SAVE	
SCHEMA	
SCREEN	
SCREENERROR	SCREENERR
SCRIPT	
SCROLL	
SEARCHED	
SECOND	
SECONDS	
SELECT	S
SELECTABLE	
SELECTED	
SEND	
SEQUENCE	SEQ

Keyword	Abbreviation
SERIAL	
SERIALIZABLE	
SESAM	
SESAMSQL	
SESSION	
SET	
SETVALUE	
SHARE	
SHIFTLEFTSTRING	SLSTR
SHOW	
SIDEINFO	
SIGN	
SIN	
SINIX	
SITENAME	
SIZE	
SMALLINT	SMINT
SOME	
SORT	
SOURCE	
SPACE	
SPACING	
SPECIAL	
SQLCODE	
SQR	
SQRT	
STABILITY	
STANDARD	STD
START	
STATIC	
STATISTICS	

Keyword	Abbreviation
STATUS	
STOP	
STORE	
SUBPROCEDURE	SUBPROC
SUBSCRIPT	
SUBSTRING	SUBSTR
SUM	
SUPERSCRIPT	
SYNONYM	
SYSTEM	
TABLE	
TABLES	
TABULATOR	TAB
TAC	
TAN	
TASKTYPE	
TEMPORARY	TEMP
TERMINAL	
TERMINATE	TERM
TEST	
TEXT	
THEN	
TIAM	
TIME	
TIMESTAMP	
TITLE	
TO	
TODAY	
TOGGLE	
TOP	
TRACE	T

Keyword	Abbreviation
TRAILER	
TRANSACTION	TA
TRSTRING	
TRUNC	
TTITLE	
TYPE	
UDS	
UNCHANGED	
UMCOMMITTED	
UNDERLINE	UL
UNION	
UNIQUE	
UNITS	
UNPROTECTED	UNPR
UNSAVE	
UPARROW	
UPDATE	UPD
UPDSTRING	UPDSTR
UPPERSTRING	
UREF	
USE	
USER	
USEREVENT	USEV
USERGROUP	
USERLABEL	
USERMSGFILE	
USERNAME	
USING	
UTM	
UTMRC	
VALID	

Keyword	Abbreviation
VALUE	
VALUES	
VARCHAR	
VARIABLE	VAR
VARYING	
VERSIONMIX	
VERTICAL	
VIEW	
VIEWS	
VISIBLE	VIS
WAIT	
WEEKDAY	
WHENEVER	
WHERE	
WHILE	
WHITE	
WIDTH	
WINDOW	
WITH	
WITHOUT	
WORK	
WRITE	
XDEC	
XREF	
YEAR	
YEARS	
YELLOW	

Related publications

[1] **DRIVE/WINDOWS V1.1** (BS2000)
Programming System
User Guide

Target group
Application programmers
Contents
– Introduction to the programming system DRIVE/WINDOWS
– Explanation of the functions available in interactive mode
– Installation
– DRIVE/WINDOWS generation and administration

[2] **DRIVE/WINDOWS** (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** (BS2000)
System 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** (SINIX)
Directory of DRIVE SQL Statements for SESAM V1.x
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 V1.x.

[5] **DRIVE/WINDOWS** (SINIX)
Directory of DRIVE SQL Statements for SESAM V2.x
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.

[6] **DRIVE/WINDOWS** (SINIX)
Directory of DRIVE SQL Statements for UDS
Reference Manual

Target group
Application programmers
Contents
A concise description of the syntax and scope of functions of all the DRIVE SQL statements for UDS.

[7] **DRIVE/WINDOWS V2.0** (MS-Windows)
Software Production Environment (SPE)
User Guide

Target group
Application programmers.
Contents
The manual describes the functions of the software production environment (desktop), how to prepare DRIVE/WINDOWS for use, remote access to BS2000 and SINIX databases and client/server applications.

[8] **DRIVE/WINDOWS V2.0** (MS-Windows)
Programming Language
Reference Manual

Target group
Application programmers.
Contents
The manual describes the creation of programs, including window and client/server applications.

[9] **DRIVE/WINDOWS V2.0** (MS-Windows)
System Directory
Reference Manual

Target group
Application programmers.
Contents
The manual describes the syntax and functions of all statements, messages and keywords of DRIVE/WINDOWS.

[10] **DRIVE/WINDOWS** (SINIX)
Software Production Environment (SPE)
User Guide

Target group
Application programmers
Contents
The functions available in the software production environment (desktop) and in expert mode. Setting up DRIVE/WINDOWS, including remote access to BS2000 databases and generating applications for BS2000.

[11] **DRIVE/WINDOWS** (SINIX)
Programming Language
Reference Manual

Target group
Application programmers
Contents
The creation of programs, including graphical and alpha screen forms, as well as list forms using DRIVE and the report generator.

[12] **DRIVE/WINDOWS** (SINIX)
System Directory
Reference Manual

Target group
Application programmers
Contents
The syntax and scope of functions of all DRIVE statements, as well as all DRIVE messages and keywords.

[13] **DRIVE/WINDOWS** (SINIX)
Directory of DRIVE SQL Statements for INFORMIX
Reference Manual

Target group
Application programmers
Contents
A concise description of the syntax and scope of functions of all the DRIVE SQL statements for INFORMIX.

[14] **DRIVE V5.1** (BS2000)
Part 1: User's Guide

Target group
– Users in non-dp departments
– Applications programmers
Contents
– General overview of the DRIVE system in old style
– Description of the DRICE components
– Introduction to DRIVE application using worked examples
– DRIVE generation and administration in UTM operation

[15] **DRIVE V5.1** (BS2000)
Part 2: System Directory

Target group
– Users in non-dp departments
– Applications programmers
Contents
– Syntax and scope of functions of all DRIVE statements in old style
– DRIVE messages and keywords

[16] **DRIVE/WINDOWS-COMP** (BS2000)
User Guide

Contents
The differences concerning the DRIVE V6.0 language, and the compilation process. Generating and starting TIAM and UTM applications with compiled DRIVE objects, with special consideration of mixed version operation.

[17] **SQL for SESAM/SQL**
Language Reference Manual

Target group
Programmers who want to access SESAM databases using SQL statements.
Contents
SQL statements available for accessing SESAM databases.

[18] **SESAM/SQL-Server** (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.

[19] **SESAM/SQL-Server** (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.

[20] **SESAM/SQL-Server** (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.

[21] **SESAM/SQL-Server** (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.

[22] **SESAM/SQL-Server** (BS2000/OSD)
Migrating SESAM Databases and Applications to SESAM/SQL-Server
User Guide

Target group
Users of SESAM/SQL-Server.
Contents
This manual gives an overview of the new concepts and functions. Its primary subject is, however, the difference between the previous and the new SESAM/SQL version(s). It contains all the information a user may require to migrate to SESAM/SQL-Server V2.0.

[23] **SESAM/SQL-Server** (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

[24] **SESAM/SQL-Server** (BS2000/OSD)
Messages
User Guide

Target group
All users of SESAM/SQL.
Contents
All SESAM/SQL messages, sorted by message number.

[25] **SQL for UDS/SQL**
Language Reference Manual

Target group
Programmers who want to access UDS databases using SQL statements.
Contents
SQL statements available for accessing UDS databases.

[26] **UDS/SQL** (BS2000)
Administration and Operation
User Guide

Target group
Database administrators
Contents
All features comprising the management and operation of the database, such as database saving, processing, restructuring, as well as outputting database information and checking the consistency of the database.
Applications
Database operation by the database administrator

[27] **UDS/SQL** (BS2000)
Creation and Restructuring
User Guide

Target group
Database administrators
Contents
– Overview of the files required by UDS
– UDS utility routines required for database creation
– Utility routines required for restructuring
Applications
Database creation by the database administrator

[28] **IFG for FHS** (TRANSDATA)
User Guide

Target group
Terminal users, application engineers and programmers
Contents
The Interactive Format Generator (IFG) is a system that permits simple, user-friendly generation and management of formats at a terminal. In conjunction with FHS, these formats can be used on the host computer. This user guide describes how formats are generated, modified and managed, plus also the new functions of IFG V8.1.

[29] **FHS** (TRANSDATA)
User Guide

Target group
Programmers
Contents
Program interfaces of FHS for TIAM, DCAM and UTM applications. Generation, application and management of formats.

[30] **UTM** (TRANSDATA, BS2000)
Generating and Administering Applications
User Guide

Target group
– System administrators
– UTM administrators
Contents
– Creation, generation and operation of UTM applications
– Working with UTM messages and error codes
Applications
BS2000 transaction processing

[31] **UTM** (TRANSDATA)
Programming Applications
User's Guide

Target group
Programmers of UTM applications
Contents
– Language-independent description of the KDCS program interface
– Structure of UTM programs
– KDCS calls
– Testing UTM applications
– All the information required by programmers of UTM applications
Applications
BS2000 transaction processing

[32] **UTM**(SINIX)
Formatting System

Target group
UTM(SINIX) users who wish to use formats, C programmers and COBOL programmers
Contents
How to use the FORMANT format handler in UTM(SINIX) program units, create formats, convert formats from BS2000 to/from SINIX.

[33] **EDT V16.5A** (BS2000/OSD)
Statements
User Guide

Target group
EDT newcomers and EDT users
Contents
Processing of SAM and ISAM files and elements from program libraries and POSIX files.

[34] **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.

[35] **BS2000/OSD-BC**
Commands, Volume1 - 3

Target group
The manual addresses both nonprivileged BS2000/OSD users and system support.
Contents
This manual contains BS2000/OSD commands (basic configuration and selected products) with the functionality for all privileges. The introduction provides information on command input.

[36] **BS2000/OSD-BC V2.0**
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

[37] **BS2000/OSD-BC** V2.0A
DMS Introductory Guide
User Guide

Target group
The manual addresses both nonprivileged users and systems support.
Contents
The manual describes file processing in BS2000, focussing on:
– file and catalog management
– files and data media
– file and data protection
– OPEN, CLOSE and EOV processing
– DMS access methods (SAM, ISAM ...).

[38] BS2000
Introductory Guide for System Users
User's Guide

Target group
BS2000 users
Contents
– Introduction to BS2000
– Description of the most frequent user commands
– Introduction to using the utility routines and software products EDT, SORT, ARCHIVE, TSOSLNK, LMS and PERCON
– Notes for the programmer
Applications
BS2000 interactive mode and batch mode

[39] **FORMANT** (SINIX)
Reference Manual

Target group
– C programmers
– COBOL programmers
– Application designers
Contents
Formant is a mask control program for all SINIX systems. The manual contains:
– Introduction to FORMANT
– Description of FORMANTGEN
– Description of user interface
– Program interfaces in C and COBOL
– Programming examples

[40] **OMNIS** (TRANSDATA, BS2000)
Administration and Programming
User Guide

Target group
– OMNIS administrators
– Programmers
Contents
Introduction to OMNIS administration, the OMNIS utility routines and the application interface for extending the OMNIS functionality
Applications
– Software development
– Application scheduling

[41] **DRIVE/WINDOWS-COMP** (SINIX)
Compiler
User Guide

Target group
Applications programmers and system administrators
Contents
Description of the compilation process using the DRIVE Compiler.

[42] **INFORMIX-NET V4.0** (SINIX)
INFORMIX-STAR V4.0 (SINIX)
User Guide

Target group
– INFORMIX users
– System administrators
Contents
This manual describes how to use the INFORMIX-NET and INFORMIX-STAR products. Using these products, INFORMIX applications can generate and process databases on foreign computers from local computers.

[43] **DRIVE/WINDOWS V1.1**
(SINIX)
Supplement
User Guide

Target group
Application programmers
Contents
The manual contains the functional changes included in DRIVE/WINDOWS (SINIX) V1.1. If this supplement is to be used, the manuals of version 1.0 are also required.

Ordering manuals

The manuals listed above and the corresponding order numbers can be found in the Siemens Nixdorf List of Publications. New publications are described in the Druck-schriften-Neuerscheinungen (New Publications).

You can arrange to have both of these sent to you regularly by having your name placed on the appropriate mailing list. Please apply to your local office, where you can also order the manuals.

Index

B

C

D

E

F

N

P

S

Contents

DRIVE/WINDOWS V2.1 (BS2000/OSD)

Directory of DRIVE Statements

Reference Manual

Target Group

The manual is aimed at programmers who develop DRIVE applications or components of client-server applications using DRIVE/WINDOWS on BS2000 computers.

Contents

The manual describes all DRIVE statements in alphabetical order together with their syntax and a description of their functional scope.

Edition: February 1996

File: DRV_LEX.PDF

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.