

EDT V16.6B

Statements

Comments... Suggestions... Corrections...

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to manuals@fujitsu-siemens.com.

Certified documentation according to DIN EN ISO 9001:2000

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2000.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Copyright © Fujitsu Siemens Computers GmbH 2007.

All rights reserved.

Delivery subject to availability; right of technical modifications reserved.

All hardware and software names used are trademarks of their respective manufacturers.

Contents

1	Preface	11
1.1	Structure of the EDT documentation	12
1.2	Target groups for the EDT manuals	12
1.3	Structure of the “EDT Statements” manual	13
1.4	Changes compared to EDT V16.6A	15
1.5	Notational conventions	17
2	Introduction to EDT	19
2.1	Principle of operation of EDT	20
2.2	Working with EDT	21
2.2.1	The EDT screen	21
2.2.2	Statements in EDT	23
2.3	Updating files	24
2.4	Example of how to process a file	27
3	Using EDT	33
3.1	Calling EDT	33
3.2	Interrupting and terminating EDT	37
3.2.1	EDT command return code	39
3.2.2	Monitoring an EDT session with monitoring job variables	41
3.3	Input and output	42
3.3.1	Entering data (text)	44
3.3.2	Entering statements	44
3.3.3	Indirect specification of operands	45
3.3.4	Symbolic line numbers	46

Contents

3.3.5	Uniqueness of string variables	47
3.4	Work file concept	48
3.5	File processing	49
3.5.1	Processing ISAM files with nonstandard attributes	50
3.5.2	Processing SAM files with nonstandard attributes	52
3.6	Processing POSIX files	54
3.6.1	POSIX in BS2000	54
3.6.2	EDT and POSIX	55
3.6.3	Processing POSIX files	57
3.6.4	Overwriting read-only files	57
3.7	Library processing with EDT	58
3.7.1	Element types supported by EDT	60
3.7.2	Processing library elements using EDT	61
3.8	SDF support for the writing of system procedures	62
3.9	Extended Host Code Support (XHCS)	63
3.9.1	XHCS and EDT	63
3.9.2	XHCS in EDT interactive mode	64
3.9.3	XHCS in EDT procedure mode	66
3.10	Job variables	67
3.11	SDF-P support	68
3.12	Task switches	69
3.13	Data protection	71
3.13.1	Constraints on privileged user IDs	71
3.13.2	Uninterruptible procedures	72
4	EDT operating modes	73
4.1	F mode	73
4.1.1	Work window	75
	Mark column	75
	Line number display	76
	Data window	76
	Statement line	78
	Status display	79
	Processing sequence	81
4.1.2	F keys	83
4.1.3	K keys	84

4.1.4	Statement codes in F mode	85
	Overview of EDT statement codes	86
	* Clear copy buffer	87
	A,B,O Mark line as destination	88
	C Mark for copying	90
	D Delete records	92
	E Insert characters	93
	J Chain two records	95
	K Copy line to statement line	96
	L Convert marked records to lowercase	97
	M Copy and delete marked lines	98
	n/l Insert lines	100
	R Mark for copying (without clearing copy buffer)	103
	S Position work window (horizontal and vertical)	105
	T Syntax test by SDF	107
	U Convert marked records to uppercase	112
	+/- Position work window	113
	+/- Position work window by structure depth	114
	X Modify lines	116
	D Delete record mark	118
	m Set record mark	118
4.1.5	Statement in the data window - split record	119
4.1.6	Statements in the statement line	119
	+/- Position within work file	120
	>/< Position horizontally within work file	122
	# Display last statement	124
	fwkfnr/fwkfv Switch work files	125
	EDIT LONG Display records with more than 80 characters	126
	HEX Switch on hexadecimal code	128
	INDEX Select work window format	130
	SCALE Display column counter	131
	SHIH Display statement buffer	133
	SPLIT Display 2 work windows	134
4.1.7	Description of the record marks in F mode	136
4.1.8	Statements in F mode	137
4.2	L mode	138
4.2.1	Input in L mode	138
4.2.2	Statements in L mode	139

5	EDT procedures	141
5.1	EDT input sources	141
5.2	EDT variables	143
5.3	Creating, calling and executing EDT procedures	145
5.4	@DO procedures	147
5.5	@INPUT procedures	149
5.6	Calling an EDT procedure in a BS2000 system procedure	153
5.7	Unconditional and conditional branches	155
5.8	External and internal loops	156
5.9	Variable EDT procedures - parameters	158
6	EDT statements	161
6.1	Description of the syntax	161
6.2	Overview of the EDT operands	164
6.3	Overview of the EDT statements	180
	EDT management	180
	File processing	185
	Processing of POSIX files	186
	Program library and file processing	187
	Switching or positioning the work file	188
	Line number handling	189
	Creating, inserting and modifying texts	191
	Copying and moving lines	192
	Deleting work files, lines, texts and record marks	193
	Comparing work files	194
	Changing the operating mode	194
	Output of lines and information	194
	Interrupting or terminating EDT	197
	Branching within EDT procedures	197
	Management and execution of EDT procedures	198
	Calling a user program	201
	Erasing, reading, cataloging and outputting job variables	201
	Declaring and reading S variables and list variables	202

6.4	Description of the statements	203
	@ Change current increment value and line number	203
	@+ Increment current line number	206
	@- Decrement current line number	207
	@: Define statement symbol	208
	@AUTOSAVE Automatic saving	209
	@BLOCK Set or reset block mode	211
	@CHECK Check lines	213
	@CLOSE Close and write file or library element	214
	@CODE Convert character codes	216
	@CODENAME Switch explicitly to different CCSN	223
	@COLUMN Insert text or delete blanks at end of line	224
	@COMPARE Compare work files line-by-line	226
	@CONTINUE Define branch destination	238
	@COPY Copy data	240
	@CREATE Create text lines	248
	@DELETE Delete work files, library elements and record marks	253
	@DELIMIT Define text delimiter characters	258
	@DIALOG Switch to F mode screen dialog	259
	@DO Start EDT procedure	262
	@DROP Delete work files	271
	@EDIT Switch edit mode	273
	@ELIM Delete ISAM file	275
	@END Terminate processing of current work file	277
	@ERAJV Delete job variables	279
	@EXEC Start program	280
	@FILE Preset file name	282
	@FSTAT Display catalog information	284
	@GET Read ISAM file	287
	@GETJV Read value of job variable	289
	@GETLIST Read elements of list variable	291
	@GETVAR Read S variable	293
	@GOTO Branch to line number in procedure	294
	@HALT Terminate EDT	295
	@IF Query strings, line numbers, integers and switches	297
	@INPUT Define input mode or start procedure	313
	@LIMITS Display line numbers	321
	@LIST Print contents of work file	322
	@LOAD Load program	325
	@LOG Control logging in batch mode	327
	@LOWER Specify uppercase and lowercase display	328
	@MOVE Move line ranges	329
	@NOTE Place comment in EDT procedure	333
	@ON Process file with search string	334

@OPEN	Open and read file or library element	376
@P-KEYS	Define programmable keys	384
@PAGE	Execute form feed	385
@PAR	Enter default parameters	386
@PARAMS	Define EDT parameters	396
@PREFIX	Insert string as prefix	402
@PRINT	Print or display lines or string variables	404
@PROC	Switch work files	408
@QUOTE	Redefine delimiter for strings	413
@RANGE	Define line range symbol	414
@READ	Read SAM file	415
@RENUMBER	Renumber lines	420
@RESET	Reset EDT and DMS error switches	422
@RETURN	Terminate screen dialog and abort procedures	423
@RUN	Call user program as subroutine	426
@SAVE	Write as ISAM file	427
@SDFTEST	Start SDF syntax check on data lines	430
@SEARCH-OPTION	Set default value for searching with @ON	433
@SEPARATE	Perform line break	434
@SEQUENCE	Generate or check line numbers	437
@SET	Supply values for EDT variables	442
@SET	Specify new current line number and increment	469
@SETF	Position window	471
@SETJV	Catalog job variable and assign value	473
@SETLIST	Extend list variable	474
@SETSW	Set switches	476
@SETVAR	Declare S variable and assign value	478
@SHOW	Display directory	479
@SORT	Sort lines in line range	485
@STAJV	Output information on job variables	487
@STATUS	Show current EDT settings and variable contents	490
@SUFFIX	Append string to lines	494
@SYMBOLS	Define symbols	496
@SYNTAX	Set syntax check and execution mode	498
@SYSTEM	Enter system commands	500
@TABS	Set tabs	502
@TMODE	Display task information	506
@UNLOAD	Unload module	507
@UNSAVE	Delete file	508
@UPDATE	Update records	509
@USE	Define external statement routines	514
@VDT	Control screen output	516
@VTCSET	Control screen output	517
@WRITE	Write file or library element	518

	@XCOPY Read POSIX file	524
	@XOPEN Open and read POSIX file	526
	@XWRITE Save contents of current work file to POSIX file	528
	@ZERO-RECORDS Setting empty line mode	530
7	EDT messages	533
<hr/>		
8	Installation notes	593
<hr/>		
8.1	Product components	594
8.2	EDTSTART start procedure	597
8.3	EDT as a subsystem	597
8.4	Installation notes for the module CODTAB	598
	Related publications	601
<hr/>		
	Index	603
<hr/>		

1 Preface

The EDT (EDITOR) is used to edit files. It can edit SAM and ISAM files, elements of program libraries and POSIX files.

With EDT, the user can

- open, create, close and store files or elements
- update files or elements (by deleting, inserting and modifying data)
- search files or elements for specific data
- compare files or elements with each other
- display or print the contents of files or elements.

For data processing, EDT offers the following facilities:

1. Virtual processing of files and library elements in interactive mode
 - a) creation and processing in the user address space
 - b) writing and storing a file or a library element from the user address space to disk or tape.

The main advantages of processing in the user address space are that

- the file is closed during processing, and
- the number of disk access operations required is minimal.

2. Real processing of files in interactive mode
The files can be processed directly on the disk.
3. Processing of files and library elements using EDT procedures
File processing operations which have to be executed frequently in the same or a similar manner can be programmed as EDT procedures.
4. Processing in batch mode
Although EDT was designed as an interactive program, it can also be used for the virtual or real processing of files and library elements in batch mode.

EDT can

- call another program as a subroutine, or
- be called by another program as a subroutine.

1.1 Structure of the EDT documentation

The complete documentation for EDT comprises three manuals:

- Statements
- Subroutine Interfaces
- Statement Formats (Ready Reference)
- EDT Operands (Reference Card)

The “Statements” manual describes all EDT statements and should be available to every EDT user. It offers a brief introduction to EDT, but is mainly intended as a reference volume for the numerous EDT statements.

The “Subroutine Interfaces” manual describes the EDT subroutine interfaces. It is helpful only in conjunction with the “Statements” manual.

The “Ready Reference” contains summary descriptions of all EDT statements.

1.2 Target groups for the EDT manuals

While the “Statements” manual (which you are reading right now) is directed mainly at EDT novices and end users, its companion volume, “Subroutine Interfaces”, is intended for seasoned EDT users and programmers who wish to employ EDT in their own programs.

The present manual, “EDT Statements”, is aimed at the full range of EDT users from the beginner to the expert (for the latter chapter 6, “EDT statements”, which contains descriptions of all EDT statements, constitutes an indispensable source of reference). EDT users wishing to write their own EDT procedures or to adapt existing ones will find in chapter 5, “EDT procedures”, a valuable introduction to writing procedures with EDT.

In order to call EDT, users should be familiar with the most important BS2000 commands.

1.3 Structure of the “EDT Statements” manual

This manual begins with an introduction to EDT and continues by describing the processing of files and library elements (members) and the creation and application of EDT procedures. It also provides an overview of all EDT statements, accompanied by detailed descriptions and a variety of examples.

Breakdown of the various chapters:

- **Introduction to EDT**

Brief introduction for newcomers to EDT.

- **Using EDT**

Explanation of how to call and terminate EDT, and of the processing of files and library elements.

- **EDT operating modes**

File processing in F mode: screen-oriented operation with EDT, description of the statement codes and statements which can only be used in F mode.
File processing in L mode.

- **EDT procedures**

Application of EDT procedures (creation, invocation, execution).

- **EDT statements**

EDT statements in alphabetical order, accompanied by numerous examples. Overview of EDT operands.

- **EDT messages**

List of all EDT messages and their “Meaning” and “Response” texts.

- **Installation notes**

Notes on installation for the system administrator.

A detailed description of the EDT subroutine interfaces can be found in the companion manual:

EDT (BS2000) V16.6
Subroutine Interfaces
User Guide

Summary descriptions of all EDT statements are contained in:

EDT (BS2000) V16.6
Statement Formats
Ready Reference

README file

Information on functional changes and additions to the current product version described in this manual can be found in the product-specific README file. You will find the README file on your BS2000 computer under the file name `SYSRME.product.version.language`. The user ID under which the README file is cataloged can be obtained from your systems support staff. You can view the README file using the SHOW-FILE command or an editor, and print it out on a standard printer using the following command:

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

or, for SPOOL versions earlier than V3.0A:

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

1.4 Changes compared to EDT V16.6A

New statements

@SHIH Display statement buffer

@ZERO-RECORDS Set empty line mode

Extensions in statements

@SDFTEST Syntax check of files by SDF

- External program name permitted (INTERNAL | EXTERNAL new operands).

@PAR Enter default parameters

- External program name permitted for SDF-PROGRAM operand.
- The new SDF-NAME-TYPE operand controls whether a program name in the @SDFTEST and @PAR SDF-PROGRAM statement is interpreted as an internal or external name (operand value INTERNAL | EXTERNAL).

@STATUS Display current settings and variable contents

Either the internal or external program name is output with @STATUS=SDF, depending on the setting. In addition, the setting of the current name type is also output.

@FSTAT Query catalog information

The length of the file name specification has been extended to 80 characters.

@SHOW Output a table of contents

The length of the file name specification in the FILES operand has been extended to 80 characters.

@ON (format 1) Output lines containing the search string

When the lines containing the search string are output, the search string can be highlighted (**E**mphasize switch).

@ON (format 7) Replace the search string

When replacing the search string, the number of hits and the number of hit lines can be written into integer variables (**V** switch).

@TMODE Output process attributes

The date is output with a four-digit year specification.

Handling data lines with length 0

The @ZERO-RECORDS statement enables you to specify that lines with length 0 should also be taken into account when reading and writing POSIX files, SAM files and library members, and that this should also apply for lines of length 8 when reading and writing ISAM files with standard properties.

Coded character set (CCS)

- It is possible to switch to another CCS name in procedure or batch mode.
- The coded character set name EDF04F is used for the EBCDIC.DF-04-15 character set (Euro character set) and for the EBCDIC.DF.04-NAF.IND character set (French-Arabic alphabet with Indian digits). As EDT cannot determine which of these character sets is meant, from V16.6B EDF04F this is by default interpreted as the Euro character set.

POSIX support

- In a TSOS ID the user can now control whether or not a read-only file should be overwritten.
- The CRTE subsystem no longer needs to be activated when the @XCOPY, @XOPEN and @XWRITE statements are used.

Writing to files with a record length > 256




It is possible to initialize SAM and ISAM files with a fixed record length > 256.

Search hierarchy for the EDTSTART file

The search hierarchy for the EDTSTART file has been extended.

1.5 Notational conventions

The following notational conventions are used in this manual:

“quotes“	Chapter/section titles and words to be emphasized.
▬	Blank.
[number]	Reference to a manual in the "Related publications" section.
	Symbolizes a key on the keyboard.
	Indicates additional information.
	Warning, for example against loss of data.

For a description of the syntax and the EDT operands, see [section “Description of the syntax” on page 161ff](#), and [section “Overview of the EDT operands” on page 164ff](#).

2 Introduction to EDT

This chapter is intended for users who are not yet familiar with EDT.

It deals only with selected functions in order to simplify the introduction to the operating principles and handling of EDT.

EDT is an aid for the systematic creation and editing of texts.

With EDT, the user can

- create files and library elements
- enter, modify, insert and delete data
- write files and library elements to disk and read them from disk
- search a file or a library element for specific data
- display or print the data.

EDT can be used to process files or library elements for a wide range of applications, such as:

- text files or tables (for bookkeeping, inventories, ...)
- source programs
- test data for validating program operation
- data for the productive execution of programs
- work files.

2.1 Principle of operation of EDT

EDT can be used to create and update files (SAM, ISAM, POSIX) and library elements. These files and elements are processed in **3 memory areas**:

- the **work window** displayed on the screen of the terminal,
- the work area of EDT in virtual memory, referred to from now on as the **work file**,
- the **public space** on disk.

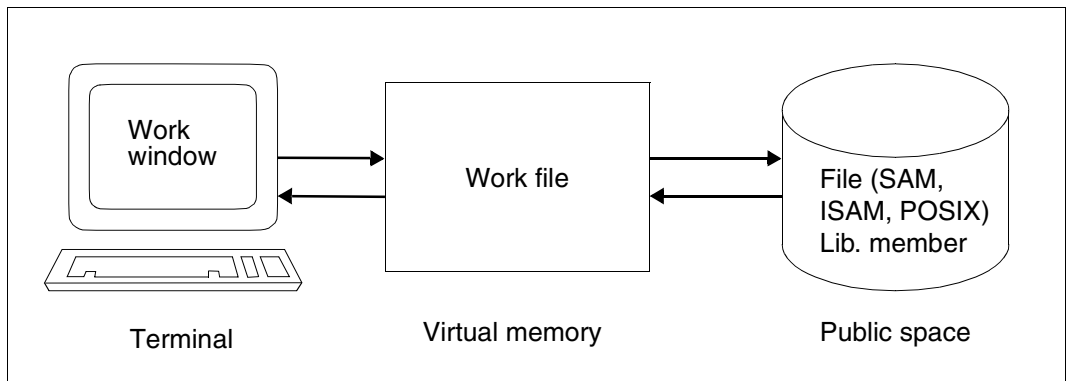


Figure 1: The memory areas of EDT

EDT creates a work area, called the work file, in virtual memory. When EDT is started, this work file is empty.

When the user wishes to create a file or a library element, he/she enters the data in the work window. The contents of this work window are then transferred to the work file by pressing a function key. When data input has been completed, EDT statements are used to write the work file contents to disk in the form of a SAM or ISAM or POSIX file or a library element.

In order to update an existing file or library element, the file or element must first be read into the work file. When this is done, the first 23 lines of this work file are displayed in the work window. The disk file remains unchanged.

EDT statements are used to execute the desired functions, such as inserting, modifying or deleting data, in the work file. The changes are stored and displayed in the work window. The **[DUE]** key transfers the contents of the work window to the work file. Newly entered texts and corrections are stored in the correct positions in this file. Finally, the updated work file is written back to the disk file with the aid of an EDT statement. This overwrites the old contents of the disk file.

Whenever data is transferred from one memory area to another, the information in the source memory area always remains unchanged, i.e. the data is available in both memory areas. If errors occur during processing in the target memory area, it is thus always possible to retrieve the original data and rectify the error.

If, for example, a file which has been transferred from disk to the work file is updated, the user can still access the disk# file at any time if an error occurs. Similarly, an error during the updating of lines on the screen can be rectified by fetching the “old” data again from the work file, thus discarding the changes already made on the screen.

2.2 Working with EDT

2.2.1 The EDT screen

The file editor EDT is called up by means of the system command **START-PROGRAM \$EDT** or **START-EDT** (as of BS2000/OSD V2.0).

The screen displays an empty work window.

```

1.00 .....
2.00 .....
3.00 .....
4.00 .....
5.00 .....
6.00 .....
7.00 .....
8.00 .....
9.00 .....
10.00 .....
11.00 .....
12.00 .....
13.00 .....
14.00 .....
15.00 .....
16.00 .....
17.00 .....
18.00 .....
19.00 .....
20.00 .....
21.00 .....
22.00 .....
                                EDT V16.6A00
.....0000.00:001(0)

```

Figure 2: The work window displayed when EDT is called

The work window is made up of 5 areas:

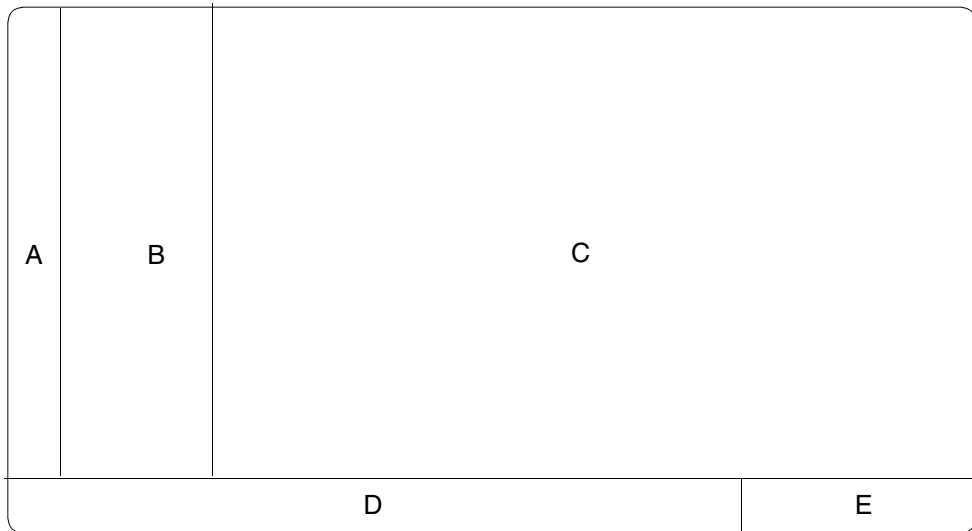


Figure 3: The 5 areas making up the EDT work window

- A **The mark column** (column 1 on the screen)
Lines in the work window can be marked for specific purposes (see below) by entering certain characters in the mark column.
- B **The line number display**
The line numbers of the text in the data window are shown here.
- C **The data window**
This is where the records are entered and/or displayed.
- D **The statement line** (the bottom screen line)
EDT statements must be entered in this statement line.
- E **The status display**
The first part of the status display is the line number of the first line in the data window, followed by a colon. The second part is the current column number. The last part of the status display is the number of the work file, which is enclosed in parentheses.

EDT is a **screen-oriented** editor, i.e. the user can, with the aid of EDT statements, display any part of a file in the data window and process it as desired, e.g. by overwriting the text or deleting or inserting lines. The data is transferred to the work file by means of `[DUE]`.

2.2.2 Statements in EDT

Functions can be implemented by means of:

- statements, which must be entered in the statement line of the work window (see [chapter “EDT statements” on page 161ff](#)).
- statement codes, which must be entered in the mark column of the work window (see [section “Statement codes in F mode” on page 85ff](#)).
- statements in the data window (see [section “Statement codes in F mode” on page 85](#)).

Statements are passed to EDT by means of one of the following keys:

- `DUE` or `DUE1`

The statements or statement codes in the work window are executed.

- `F2`

The statements or statement codes in the work window are executed and the data window is set to overwritable (bright). The data can now be modified as desired without the need for any statements.

Statements may be entered in the statement line (see [chapter “EDT statements” on page 161ff](#)) or in the mark column (see [section “Work window” on page 75ff](#)) in either uppercase or lowercase letters.

2.3 Updating files

Entering data

Records with any desired contents are entered in the data window from the keyboard. First, the cursor is positioned to the beginning of the first line by hitting `[←]`.

The text is then entered in the line. Hitting `[→]` causes the cursor to move to the beginning of the next line, where further text can be entered.

When all desired text lines have been entered, or when all lines in the data window have been used, the text must be sent off by hitting `[DUE]` or `[DUE1]`.

If more text is to be entered, the user must enter + in the statement line and transmit the data by hitting `[DUE]`. The last line of the old data window now appears as the first line in the new data window.

`[→]` must now be pressed twice to move the cursor to the beginning of the second line.

Further text lines can now be entered as described above.

Storing (writing) a newly created work file

After the data has been entered, the work file is written into a file or a library element by means of `@WRITE` or `@SAVE` (see `@WRITE`, `@SAVE`).

Creating (writing) a SAM file on disk or tape:

```
@WRITE 'filename'
```

Creating (writing) an ISAM file on disk or tape:

```
@SAVE 'filename'
```

Creating (writing) a library element:

```
@WRITE LIBRARY = libname (ELEMENT = elemname)
```


Reading a file into the work file

An existing file is read into the work file by means of @READ or @GET (see @READ, @GET).

Reading in a SAM file:

```
@READ 'filename'
```

Reading in an ISAM file:

```
@GET 'filename'
```

Reading in a library element:

```
@COPY LIBRARY = libname (ELEMENT = elemname)
```

The beginning of the file or library element is displayed in the data window.

Correcting characters

The desired section of the file or element is moved into the data window by means of the +, +n, ++, -, -n, -- statements.

- To make the data window overwriteable, you can either press **F2** or mark the line with an X in the mark column.

The data window is displayed with high intensity, indicating that it is overwriteable.

The cursor is then moved by means of the positioning keys to the character in the data window which is to be corrected.

The text can now be corrected by overwriting or by inserting or deleting.

If the line number indicator is switched on, @PAR EDIT FULL=ON can always be used to set the data window to overwriteable. At the same time, statement codes can also be specified in the mark column.

Hitting **→** then positions the cursor at the mark column in the next line or at the beginning of the data line.

- Characters are deleted by pressing **AFG**.
- Characters are inserted by pressing **EFG**.

Characters can then be inserted at the desired position. Note that any characters which are shifted past the right-hand edge of the data window as a result of insertion will be lost.

- Insert mode is switched off using the **[RS]** key.

The corrected data window is sent to the work file by hitting **[DUE]**.

If the corrections are not to be sent to the work file (e.g. because the input was incorrect) the original screen contents can be restored by pressing **[K3]**. This will work only if the incorrect contents of the data window have not yet been copied into the work file using **[DUE]**.

Inserting lines

The file section in which lines are to be inserted is first moved to the data window by means of the +, +n, ++, -, -n or -- statement (see above).

In the line before which the lines are to be inserted, a number between 1 and 9 (indicating the number of lines to be inserted) is entered in the mark column and the **[DUE]** key is hit. The requested number of empty lines now appears in the data window and text may be entered in these lines exactly as for creation of a new file. Finally, the **[DUE]** key must be pressed.

If you wish to enter further text, repeat the procedure (see also statement code I).

Deleting lines

The file section in which lines are to be deleted is moved to the data window by means of the +, +n, ++, -, -n or -- statement.

The letter D is then entered in the mark column of the line(s) to be deleted and the **[DUE]** key is pressed.

The lines marked in this manner are deleted from the work file. The data window in which the lines were deleted is displayed again, with the lines following the deleted lines moved up to occupy the space which has become free.

Storing files and library elements

Updated files or library elements are saved by means of @WRITE or @SAVE, or closed by means of @CLOSE.

2.4 Example of how to process a file

In this example, the SAM file DRUGSTORE, which is stored in public space on disk, is read into virtual memory as a work file. The work file DRUGSTORE is then processed in the data window:

- The entry in the column ORDERED is corrected in one line.
- 4 lines are deleted from the work file, since these articles are no longer stocked.
- 4 lines are inserted into the work file and one line is appended to the work file, since 5 new articles are to be added to the sales range.

The work file DRUGSTORE, as modified in the data window, is then written back to disk: the original contents of the disk file DRUGSTORE are overwritten by the (modified) contents of the work file.

```

/start-program $edt

1.00 .....
2.00 .....
3.00 .....
4.00 .....
5.00 .....
6.00 .....
7.00 .....
8.00 .....
9.00 .....
10.00 .....
11.00 .....
12.00 .....
13.00 .....
14.00 .....
15.00 .....
16.00 .....
17.00 .....
18.00 .....
19.00 .....
20.00 .....
21.00 .....
22.00 .....
                                EDT V16.6A00
@read 'drugstore' .....0000.00:001(0)

```

The SAM file DRUGSTORE is read into work file 0. All changes are made in the virtual memory of EDT and must then be saved to a file by means of a @WRITE or @SAVE statement.

SEQ.NR	ART.NO.	ART.NAME	STOCK	ORDERED
1.00			
2.00	1	0024 SOAP	3000	150.....
3.00	2	0015 DEODORANT	2500	600.....
4.00	3	0048 PARFUME	400	60.....
5.00	4	0003 CREME	987	555.....
6.00	5	0091 SHAVING FOAM	350	30.....
7.00	6	0090 AFTER SHAVE	340	30.....
8.00	7	0092 SHAVING BRUSH	200	30.....
9.00	8	0054 TOOTH PASTE	400	50.....
10.00	9	0055 TOOTH BRUSH	200	30.....
11.00	10	0061 SHOWER GEL	250	40.....
12.00	11	0062 BATH SALTS	150	55.....
13.00	12	0071 BODY LOTION	100	80.....
14.00	13	0073 HAND CREAM	350	30.....
15.00	14	0075 NIGHT CREAM	240	20.....
16.00	15	0076 DAY CREAM	300	-.....
17.00	16	0105 SUN LOTION	160	200.....
18.00	17	0107 SUN OIL	220	200.....
19.00	18	0121 SUN GLASSES	50	50.....
20.00	19	0144 COSMETIC BAG	30	35.....
21.00	20	0056 COMB	40	200.....
22.00	21	0057 HAIR BRUSH	70	150.....
23.00	22	0058 MASSAGE BRUSH	35	40.....
+0001.00:001(0)

The data window is to be moved one data window towards the end of the file and then set to overwritable.

+ is entered in the statement line and sent off by hitting function key **F2**.

24.00	23	0039 SHAMPOO	600	300.....
25.00	24	0010 TISSUES	1500	500.....
26.00	25	0053 MANICURE SET	80	50.....
27.00	26	0201 DIAPERS	2000	500.....
28.00	27	0210 BABY CREAM	1300	100.....
29.00	28	0211 BABY OIL	700	400.....
30.00	29	0220 BABY FOOD	4000	200.....
31.00			
#210021.00:001(0)			

F2 switched the data window to overwritable, which means that the number 50 can now be entered directly in the ORDERED column of line 26.00.

The data window is now to be positioned to line 21. This is done by entering #21 in the statement line and pressing **DUE**.

```

21.00 20      0056  COMB          40      200.....
d 22.00 21      0057  HAIR BRUSH   70      150.....
d 23.00 22      0058  MASSAGE BRUSH 35      40.....
d 24.00 23      0039  SHAMPOO      600     300.....
d 25.00 24      0010  TISSUES      1500    500.....
  26.00 25      0053  MANICURE SET  80      50.....
  27.00 26      0201  DIAPERS      2000    500.....
  28.00 27      0210  BABY CREAM   1300    100.....
  29.00 28      0211  BABY OIL     700     400.....
  30.00 29      0220  BABY FOOD    4000    200.....
  31.00 .....

```

Lines 22.00 through 25.00 are to be deleted. This is done by entering D in the mark column of each of these lines and sending of the data window with **DUE**.

```

21.00 20      0056  COMB          40      200.....
4 26.00 25      0053  MANICURE SET  80      50.....
  27.00 26      0201  DIAPERS      2000    500.....
  28.00 27      0210  BABY CREAM   1300    100.....
  29.00 28      0211  BABY OIL     700     400.....
  30.00 29      0220  BABYFOOD     4000    200.....
  31.00 .....

```

Lines 22.00 through 25.00 have been deleted.

Four new lines are now to be inserted before line 26. 4 is entered in the mark column of line 26 and **DUE** is pressed.

```

21.00 20      0056  COMB          40      200.....
22.00 21      0133  ski glasses  100     20.....
23.00 22      0134  ski wax      500     -.....
24.00 23      0138  gloves       150     -.....
25.00 24      0139  scarf        15      30.....
26.00 25      0053  MANICURE SET  80      50.....
27.00 26      0201  DIAPERS      2000    500.....
28.00 27      0210  BABY CREAM   1300    100.....
29.00 28      0211  BABY OIL     700     400.....
30.00 29      0220  BABY FOOD    4000    200.....
31.00 .....
32.00 .....
33.00 .....
34.00 .....
35.00 .....
36.00 .....
37.00 .....
38.00 .....
39.00 .....
40.00 .....
41.00 .....
42.00 .....
43.00 .....
+ .....0021.00:001(0)

```

New articles are entered in the newly inserted lines 22.00 through 25.00.

The data window is then to be moved one data window towards the end of the file. + is entered in the statement line and sent off by hitting **[DUE]**.

```

30.00 29      0220  BABY FOOD      4000    200.....
31.00 30      0130  nasal spray    250     40.....
32.00 .....

@write 'drugstore'.....0030.00:001(0)

```

Since line 30.00 is the last line in the work file, EDT positions the work file to this line. New articles can now be entered at the end of the file without the need for special statements: line 31 is created automatically.

Finally, the changes are to be written back to file DRUGSTORE on disk. This is done by entering **@WRITE 'DRUGSTORE'** in the statement line and sending it off by means of **[DUE]**.

```

30.00 29      0220  BABY FOOD      4000    200.....
31.00 30      0130  nasal spray    250     40.....
32.00 .....

% EDT0903 FILE 'DRUGSTORE' IS IN THE CATALOG, FCBTYP = SAM
y EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO).....0030.00:001(0)

```

Since file DRUGSTORE already exists, EDT asks whether this file is to be overwritten.

The file is to be overwritten. This is indicated by entering Y in the statement line and pressing the **[DUE]** key.

```

30.00 29      0220  BABY FOOD      4000      200.....
31.00 30      0130  nasal spray    250       40.....
32.00 .....

% EDT0171 FILE ':3:$USID.DRUGSTORE' REPLACED AND WRITTEN
@halt.....0030.00:001(0)

```

EDT confirms that the work file was written to the file DRUGSTORE.

EDT is now to be terminated. This is done by entering @HALT in the statement line and pressing the **[DUE]** key.

If the user forgets to save his/her work files before attempting to terminate EDT, the attempt is rejected and EDT issues the message: % EDT0900 EDITED FILE(S) NOT SAVED! The numbers of the work files with data that has not been saved is then output. The user is then asked: % EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)

Response N: The EDT work window is displayed again and the user can close and write any work files that have not yet been saved.

Response Y: Any virtual files which have not been saved are lost. EDT is terminated.

3 Using EDT

3.1 Calling EDT

Calling EDT as the main program

EDT can be called as the main program by means of the following command:

START-PROGRAM \$EDT	AMODE 31 (if supported by the hardware)
---------------------	--------------------------------------------

or

START-PROGRAM *MODULE(\$EDTCLIB,EDTC)	AMODE 24
---------------------------------------	----------

Starting EDT as of BS2000/OSD V2.0

As of BS2000/OSD V2.0, EDT can be loaded and started by means of the START-EDT command. The START-EDT command allows the user to select a particular version of EDT if two or more versions are installed.

The START-EDT command may only be input in user IDs which have the requisite privileges (see [section "Data protection" on page 71](#)).

The alias for the START-EDT command is EDT.

START-EDT	Alias: EDT
VERSION = *STD / <product-version 6..10> /<product-version 4..8 without-correction-state> / <product-version 3..7 without-manual-release> ,MONJV = *NONE / <full-filename 1..54 without-gen-vers> ,CPU-LIMIT = *JOB-REST / <integer 1..32767> ,PROGRAM-MODE = *ANY / 24	

VERSION =

Version of EDT which is to be started.

VERSION = *STD

The version defined by the command SET-PRODUCT-VERSION is selected. If there is no version defined as standard, the system selects the highest version available.

**VERSION = <product-version 6..10> /
<product-version 4..8 without-correction-state> /
<product-version 3..7 without-manual-release>**

Explicit specification of the version.

MONJV = *NONE / <full-filename 1..54 without-gen-vers>

Name of the job variable which is to monitor the EDT session. The job variable must have been cataloged beforehand (only for users with the Job Variables software product).

During the EDT session, the system sets the job variables to the following values:

Value	Meaning of value assigned
\$R	EDT is running
\$T	EDT was terminated without errors
\$A	EDT was terminated abnormally

MONJV = *NONE

No job variable is to be used for monitoring.

CPU-LIMIT = *JOB-REST / <integer 1..32767>

The CPU time which EDT is allowed to use for execution. If EDT exceeds this time in interactive mode, the system informs the user, and if EDT exceeds this time in batch mode, the system terminates the session.

CPU-LIMIT = *JOB-REST

If the operand CPU-LIMIT=STD was specified in the SET-LOGON-PARAMETERS command, the program is not subject to any time limitation.

If the operand CPU-LIMIT=t was specified in the SET-LOGON-PARAMETERS command, the value defined at system generation will be used as the time limitation for the EDT session.

PROGRAM-MODE =

Specifies the addressing mode in which EDT is to run.

PROGRAM-MODE = *ANY

EDT is loaded into the upper address space and runs in 31-bit mode.

PROGRAM-MODE = 24

EDT is loaded into the lower address space and runs in 24-bit mode. If EDT is loaded into the upper address space as a subsystem, a private copy is loaded into the lower address space dynamically.

EDT is started in F mode.

If task switch 5 is set ([section “Task switches” on page 69](#)), L mode is activated. EDT reads the inputs from SYSDTA with RDATA.

When you call EDT, values can be preset in the following way:

- Initializing string variables by means of S variables
- Executing an EDT start procedure.

Processing is performed in the order specified.

Initializing string variables by means of S variables

When EDT is called, each string variable is initialized with a blank.

If the SDF-P subsystem is installed in the system, S variables can be used for initializing the string variables and for transferring values when EDT is terminated. Please note the following:

- If S variables from the set SYSEDT-S00 through SYSEDT-S20 of TYPE=STRING exist and have been assigned a value, their contents are transferred to the corresponding string variables #S00 through #S20, thus initializing the string variables.
- If the contents of an S variable are longer than 256 characters, no characters are transferred, i.e. the relevant string variable is not initialized. No error message is issued.
- When EDT is terminated using @HALT, the values assigned to string variables #S00 through #S20 are exported to any existing S variables (SYSEDT-S00 through SYSEDT-S20). New S variables (SYSEDT-Sxx) are not declared by EDT itself but can be declared by the user in EDT by means of a @SETVAR statement.

Processing an @INPUT start procedure

After EDT is called, as start input procedure is processed, if one exists.

A search is first made for the link name \$EDTPAR. If it exists, the file linked to it is used as the start procedure. If it does not exist, the search for the start procedure is carried out as follows:

- If a file EDTSTART exists in the current user ID, it is used as the start procedure.
- If it does not exist, the file linked with the logical ID SYSDAT.EDTSRART during installation is used as the start procedure.
- If no file is assigned to SYSDAT.EDTSTART, \$.EDTSTART is used as the start procedure, if this file exists and can be accessed.

Otherwise, no start procedure is executed.

Any EDT caller is allowed to define an individual start procedure with the SET-FILE-LINK command.

Furthermore, linking the file *DUMMY with \$EDTPAR causes no start procedure to be executed:

```
SET-FILE-LINK FILE-NAME = *DUMMY, LINK-NAME = $EDTPAR
```

See the [chapter “EDT procedures” on page 141ff.](#), and the @INPUT statement for more details. If the procedure contains a @HALT statement, processing of the @INPUT procedure is terminated. No error messages are issued.

Example of an EDTSTART start procedure

```
@IF ON=5 RETURN ----- (1)
@PAR GLOBAL INFORMATION=ON, EDIT FULL=ON ----- (2)
@GETJV '$SYSJV.JOBNAME'=1 ----- (3)
@ON 1:1-5 F 'USER1' ----- (4)
@DELETE ----- (5)
@IF .FALSE. RETURN ----- (6)
@SYMBOLS FILLER=' ' ----- (7)
```

- (1) If task switch 5 is set, terminate procedure
- (2) Display an information line in the data window
- (3) The contents of \$SYSJV.JOBNAME are written to line 1 in the work file
- (4) Check whether line 1 (contents of \$SYSJV.JOBNAME) is identical to 'USER1'. No distinction is made between 'USER1' and, for example, 'USER11'.
- (5) The work file is deleted (delete line 1).
- (6) If no hit was found in @ON (line 1 is not identical to 'USER1'), abort the procedure.
- (7) Define the filler character between the end of the record and the end of the screen line as a blank.

Calling EDT as a subroutine

EDT can be called not only as the main program, but also as a subroutine from a user program.

Calling EDT as a subroutine is described in the manual “EDT Subroutine Interfaces” [1].

3.2 Interrupting and terminating EDT

Interrupting an EDT session

In both F mode and L mode, the EDT session can be interrupted by means of @SYSTEM or by hitting **[K2]**. In either case, EDT remains loaded.

The user can return to the interrupted EDT session using the command RESUME-PROGRAM, which causes the EDT session to be resumed at the position where it was interrupted. If, in F mode, the work window in which the EDT session was interrupted is not displayed or is displayed only partially after RESUME-PROGRAM, the original contents of the work window can be restored by hitting **[K3]**.

If the EDT session was interrupted in F mode, the user can return to F mode using the command SEND-MESSAGE TO=PROGRAM. The rest of the statement line is not, however, executed.

If the EDT session was interrupted in L mode, the user can return to L mode using the command SEND-MESSAGE TO=PROGRAM. This causes the STXIT routine of EDT to be executed. This routine closes all open files (except for files opened by means of @OPEN). If there is no active @INPUT or procedure file, the STXIT routine displays the current statement symbol on the screen.

If, at the time of the interruption, EDT had not yet fully processed the lines of a @DO or an @INPUT procedure or the lines of an input block (BLOCK mode) and if the SEND-MESSAGE command is then used to return to the EDT session, the processing which was interrupted will be aborted and the remaining lines will not be executed.

If the command START-PROGRAM or LOAD-PROGRAM is entered, or procedures containing these commands are started, while EDT is interrupted, then EDT is unloaded, regardless of whether it was interrupted in F mode or L mode.



The EDT run cannot be interrupted if EDT was started within a BS2000 system procedure protected against interruption by means of the setting INTERRUPT-ALLOWED=NO (see [section “Data protection” on page 71](#)).

Terminating an EDT session

The @HALT, @RETURN, @EXEC and @LOAD statements and the **[K1]** key all terminate an EDT session. EDT closes all files which are open.

In interactive mode, EDT can be terminated with @END. In L mode, a message is first issued.

@HALT ABNORMAL can be used in interactive mode or in a system procedure to force abnormal termination of the EDT session.

If an attempt is made to terminate EDT while there are still unsaved work files, EDT is not terminated. The numbers of the work files containing unsaved data are output after the following message:

```
% EDT0900 EDITED FILE(S) NOT SAVED!
```

The user then receives the following query:

```
% EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)
```

The user responses have the following effects:

- N The EDT work window is displayed again. The user can now close and save any files which had not been saved before.
- Y Any unsaved virtual files are lost. EDT is terminated.

If the SDF-P subsystem is available in the system and S variables SYSED-T-S00 through SYSED-T-S20 with TYPE=STRING exist, terminating EDT with @HALT or @END will cause the values of string variables #S00 through #S20 to be exported to S variables SYSED-T-S00 through SYSED-T-S20. New S variables SYSED-T-Sxx are not declared by EDT itself, but can be declared in EDT with the statement @SETVAR.

If the event "Program runtime exceeded" occurs (EDT runtime is greater than the value specified for CPU-LIMIT in the START-PROGRAM command), a message is output to SYSOUT and EDT terminates abnormally.

If the interrupt event PROCHK (program check) or ERROR (unrecoverable program error) occurs and the EDT data area is still addressable, message EDT8910 is output, in which the program counter and the interrupt weight are specified. In interactive mode, EDT is terminated with errors or an attempt is made (e.g. in the event of a data error in L mode) to remove the invalid data by deleting the current work file. If this is not possible, TERM is issued with a request for a memory dump.

Regardless of whether EDT is terminated normally using @HALT, @RETURN or, in interactive mode, using @END, or is terminated abnormally by the system or the user with @HALT, information on the cause of termination and on the EDT session is made available for use in controlling system procedures in which EDT is called.

This information is not made available for an EDT session which was aborted with @EXEC or @LOAD.

3.2.1 EDT command return code

EDT provides a command return code that can be used by SDF-P for controlling S procedures. This command return code makes it possible to make a targeted response to particular error situations.

The command return code consists of three parts:

- the main code, which corresponds to a message code by means of which detailed information can be queried with the command HELP-MSG-INFORMATION,
- subcode1 (SC1), which categorizes the error situation that occurred into an error class indicating how serious the error is and
- subcode2 (SC2), which may contain additional information (value other than null).

(SC2)	SC1	Main code	Meaning
0	0	EDT8000	Normal termination of the EDT session. No messages were issued.
2	0	EDT8000	Normal termination of the EDT session. Any messages issued were of message levels* 0, 1 or 2 only. No syntax errors occurred, only information, warnings or messages.
5	0	EDT8000	Normal termination of the EDT session. A message of message level* 4 or 5 was issued. No syntax errors occurred. An error in function or execution occurred.
10	0	EDT8000	Normal termination of the EDT session. An syntax error occurred in a statement (message ¹ 3).
50	64	EDT8100	Abnormal termination by the user (@HALT ABNORMAL).
100	64	EDT8200	Abort due to time overrun (RUNOUT).
100	64	EDT8292	Error in RDATA; program aborted.
100	64	EDT8901	Abort due to data error.
100	64	EDT8902	Abort due to data error (ENTER process).
150	64	EDT8910	Program interruption; abnormal abort with DUMP.
150	64	EDT8001	Abnormal termination of the EDT session; abnormal abort with DUMP.
200	64	EDT8002	Error in dynamic loading of EDT mode.
200	64	EDT8003	Insufficient virtual memory available.
200	64	EDT8005	Error in initializing EDT.
200	64	EDT8006	Installation error in EDT or EDTLIB.
200	64	EDT8300	Internal EDT error.
200	64	EDT8900	No virtual addressing space.

¹ The message level is the thousand's place of the message number

In the event of an error, the components of the return code can be queried with the SDF-P functions SUBCODE1(), SUBCODE2() and MAINCODE().

The return code can also be saved and evaluated following an error-free session with the command SAVE-RETURNCODE. (For more detailed information on command return codes and how to query them, see the manual “SDF-P“ [13]).

Example of how to query return codes

```
/MODIFY-JOB-SWITCHES ON=5
/START-PROGRAM $EDT
@LOG NONE
@...
@DIALOG
@...
@HALT
/SAVE-RETURNCODE
/IF-BLOCK-ERROR
/  WRITE-TEXT 'FEHLER: &SUBCODE1, &SUBCODE2, &MAINCODE'
/ELSE
/  WRITE-TEXT 'EDT NORMAL BEENDET'
/  IF (&SUBCODE2 > 5)
/    WRITE-TEXT 'SYNTAX FEHLER IST AUFGETRETEN'
/    RAISE-ERROR MAINCODE=EDT3002
/  END-IF
/  ...
/END-IF
/HELP-MSG-INFORMATION &MAINCODE
/MODIFY-JOB-SWITCHES OFF=5
```


3.2.2 Monitoring an EDT session with monitoring job variables

EDT execution can be monitored with a BS2000 job variable.

To have the operating system set up the monitoring job variable, use the following commands:

```
START-PROGRAM $EDT,MONJV=jvname or
START-EDT MONJV=jvname (as of BS2000/OSD V2.0)
```

The operating system maps two values in the job variable:

- a three-byte status indicator and
- a four-byte return code indicator.

The following table shows the values EDT may place in the job variable.

Error class	Termination	Status indicator	Return code	Spin-off mechanism
[No message] [Note] [Function error] [Syntax error]	Normal	\$T	0000 1002 1005 1010	no
[Interruption]	abnormal by the user	\$A	2050	yes
[Fatal] [Fatal]+DUMP [Initialization error]	abnormal		2100 2150 3200	

The last three places of the return code are identical in value and meaning to subcode2 (SC2) of the command return code.

3.3 Input and output

Input

Input to EDT can be entered as follows:

- primarily via the screen
- from a SAM or ISAM file
- from a library element
- from a POSIX file or
- from another EDT work file.

EDT makes a distinction between data (text) and statements in the input.

Output

EDT can output all or part of any work file:

- primarily to the screen
- to a SAM or ISAM file on disk
- to a library element
- to a POSIX file
- to another EDT work file or
- to the printer.

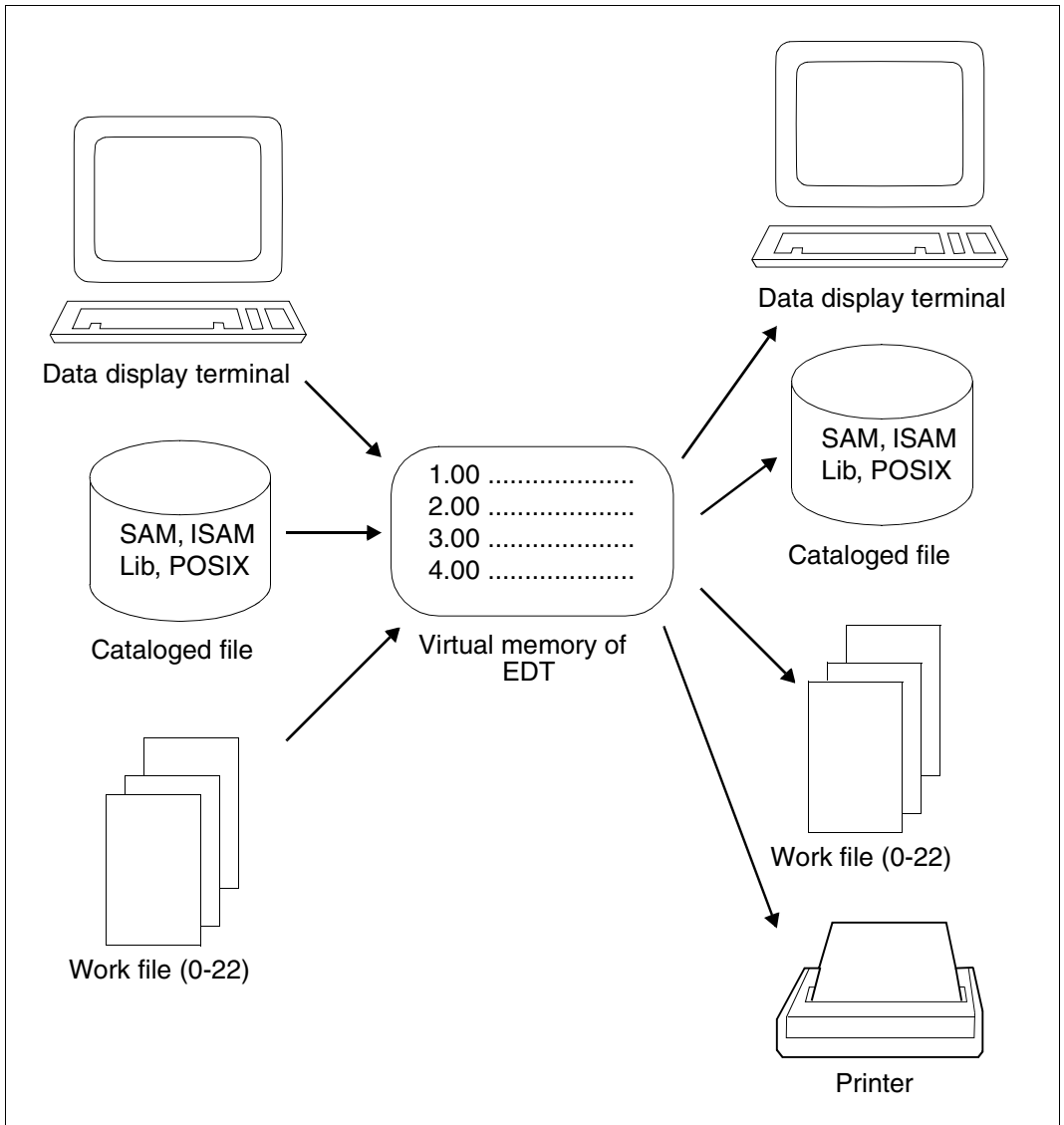


Figure 4: Input and output in EDT

3.3.1 Entering data (text)

Data is transferred to the file in the order in which it is entered when the file is being created. The text is formatted in accordance with any tab characters it may contain.

When a file is updated, EDT overwrites the corresponding records in the file with the updated records.

3.3.2 Entering statements

Statements control the EDT session.

EDT regards the following inputs as statements:

- in F mode, entries in the statement line or the mark column (statement codes) of the work window;
- in L mode, any input in which the first non-blank character is the statement symbol and the next non-blank character after this statement symbol is not the statement symbol.

The default statement symbol in L mode is @. If an input begins with @@ (2 statement symbols), then EDT regards this input as text. It regards the second statement symbol as the first character of this text. EDT removes all characters (the first statement symbol and any blanks) which come before the second statement symbol.

If, in L mode, EDT receives its input from the screen, it is possible, if block mode (see [chapter “EDT statements” on page 161ff](#)) is active, to pass several inputs to EDT in a single input block.

Each of these inputs may be up to 256 characters long.

If " or ' appears within a character string, then "" or "" must be entered.

Example

```
'This is a "random" character string'
```

General format of a statement:

Operation	Operands	F mode/ L mode / @PROC
Operation	{ operands &str-var }	

operation	The operation is the same as the statement name, e.g. @OPEN, @COPY, @WRITE... This must be at the beginning of the statement. In F mode, the EDT statement symbol (default value: @) may be omitted.
operands	The operation is followed by operands, which are separated from the operation by one or more blanks. The operands must be entered in the specified order. Each operand may be preceded or followed by any number of blanks.
str-var	String variable which contains the operands (indirect specification of operands).

The separator (blank) between the operation and the operands, and between the individual operands, must be entered if it is not possible to distinguish between the operation and the operand or between two operands (example: @SYMBOLS='?' is incorrect; @SYMBOL S='?' is correct).

3.3.3 Indirect specification of operands

Once the statement name (operation) has been recognized, the rest of the statement is replaced by the contents of the specified string variable and the operands contained in the variable are analyzed.

If logging has been activated (e.g. @LOG ALL or @LOG COMMANDS), the statement generated by this replacement is output in addition to the original input. If an error occurs, only the values to be used in the replacement are logged.

If the length of the statement name plus the replacement of the string variable is greater than 256, processing of the statement is rejected with the following error message:
% EDT1905 INPUT TOO LONG. CORRECT INPUT

Restrictions

Replacement is not possible in the following statements:

- statements which do not have a statement name (e.g. redefinition of the statement symbol and value assignment for EDT variables without @SET)
- statements in which the statement name cannot be clearly recognized (e.g. setting and modifying the current line number using @In or @+, @-; in the “text” operand, e.g. in an @IF statement).
- @PARAMS statement

In a @DO procedure, replacement of the procedure parameters is performed first, and then any indirect operand specification is resolved.

3.3.4 Symbolic line numbers

In some statements (e.g @ON), the line range can also be specified by means of symbolic line numbers (% , * , \$) or via the current range symbol (&).]

Symbol	Meaning
%	Line number of the first record in the work file
\$	Line number of the last record in the work file
*	Current line number
&	Set line range (cf. @RANGE, default value: 0000.0001-9999.9999)
?	Line number of the first hit line after @ON

When line ranges are specified, symbolic line numbers must be invalidated by means of a '.' (period).

Example

%.–10	Line range from the first line to line 10
*–1L	Line before the current line
10–.\$	Line range from line 10 to the last line
\$–1L	Penultimate line

3.3.5 Uniqueness of string variables

In many statements it is possible to specify string variables as operands. In this case, the variable name must be preceded by a period in order to avoid the risk of confusion with another like-named variable: `.#Sxx`.

Example for @OPEN (Format 2)

```
.  
.   
@CREATE #S01: 'TESTLIB'   
@CREATE #S02: 'ELEM'   
.   
.   
@OPEN L=.#S01(E=.#S02,S) ----- (1)
```

- 1 Specification of a period is necessary here because '#S01' could also be the file name of the library and '#S02' could also refer to the element.

3.4 Work file concept

In L mode, there are 23 virtual files in which the user can process files. These are the work files 0 through 22. Work files are always virtual files. In work file 0, it is also possible to process an ISAM file opened for real processing by means of @OPEN.

In F mode, work files 0 through 9 are available to the user for the direct input of data. In F mode, the user can also split the work window (see @PAR SPLIT) and display two work files at the same time.

Work files 9 and 10 are needed for the execution of some of the EDT statements. For this reason, these two files should only be used as temporary help files.

- Some statements store their results in work file 9 (e.g. @COMPARE, format 2, @FSTAT, @SHOW, @STATUS), overwriting the contents of this work file without issuing a warning.
- Work file 10 is used as an auxiliary file by @COMPARE, format 2, unless the user explicitly specifies another auxiliary file.

3.5 File processing

By default, EDT processes the following ISAM and SAM files:

ISAM files with

- variable record length (RECORD-FORMAT = VARIABLE(...))
- key position 5 (ACCESS-METHOD = ISAM(KEY-POSITION = 5))
- key length 8 (ACCESS-METHOD = ISAM(KEY-LENGTH = 8))
- numeric keys (X'F0'-X'F9', at least one character ≠ X'F0'),
- block size 1 (BUFFER-LENGTH = STD(1)).

EDT interprets the ISAM key as the line number of the record.

SAM files with

- variable record length (RECORD-FORMAT = VARIABLE(...))
- block size 1 (BUFFER-LENGTH = STD(1)).

Systems with the presetting BLKCTRL=DATA also have a default block-size value of BUFFER-LENGTH=STD(2) for NK4 disks.

All records in variable-record-length SAM or ISAM files which exceed 256 characters in length, will be truncated at position 257 when written back.

The files can be processed with @GET, @READ, @SAVE, @WRITE, @ELIM and @INPUT (see [chapter “EDT statements” on page 161ff.](#)). The files can also be processed with @OPEN, @WRITE, @COPY format 2. The access method is determined by the operand TYPE=SAMIISAM.

The name of the file must be enclosed in single quotes ('filename').

Instead of 'filename', the user may specify '/' if one of the following file link names was permanently assigned to the file before EDT was started:

```
/SET-FILE-LINK LINK-NAME = EDTSAM | EDTISAM, FILE-NAME = filename
```

If a file link name has been permanently assigned to a file, EDT cannot process any other SAM file (EDTSAM) or ISAM file (EDTISAM) until this assignment is canceled.

The permanent assignment is canceled by means of

```
/REMOVE-FILE-LINK LINK-NAME = EDTSAM | EDTISAM
```

If '/' is specified as the file name, EDT does not check the catalog information before opening the file and does not release superfluous storage space after closing the file. Nor does EDT ask for confirmation as to whether or not an existing file is to be overwritten. Superfluous storage space is also not released if the file name is specified under a USERID other than TSOS (unless EDT is running under TSOS) or task switch 7 is set.

3.5.1 Processing ISAM files with nonstandard attributes

If files with nonstandard attributes are to be processed, the file attributes must be specified in the SET-FILE-LINK or CREATE-FILE command.

General processing sequence:

- The file link name EDTISAM is assigned to the file. Assigning the name EDTISAM and specifying the nonstandard attributes are described in the individual steps below. If a new file is to be created, all file attributes must be specified.
- Processing via @GET or @SAVE with LINK-NAME = EDTISAM and processing via @READ or @WRITE with LINK-NAME = EDTSAM.
- It is advisable to cancel the file link name assignment by means of REMOVE-FILE-LINK LINK-NAME = EDTISAM or LINK-NAME = EDTSAM after the file has been processed.



ISAM files with nonstandard attributes can also be processed directly with @OPEN format 2 and the operand TYPE=CATALOG. In this case, the attributes are adopted directly from the catalog, and it is not necessary to assign a file link name.

ISAM files with ISAM keys less than 8 bytes long

Assignment: / SET-FILE-LINK LINK-NAME = EDTISAM, FILE-NAME = filename, -
/ ACCESS-METHOD = ISAM(KEY-LENGTH = keylength)

If a key length of less than 8 bytes is specified, then any existing ISAM keys are truncated on the left. If 4 is specified for KEY-LENGTH, for example, then the line number 1234.5678 is interpreted as the ISAM key 5678. This means that the ISAM key may no longer be unique. The user is responsible for ensuring that the ISAM keys are unique.

ISAM file with fixed record length

Assignment: /SET-FILE-LINK LINK-NAME = EDTISAM, FILE-NAME = filename, -
/ RECORD-FORMAT = FIXED(RECORD-SIZE = reclength)

The length of the key (KEY-LENGTH) may lie in the range 1 through 8. If a value other than 1 is specified for KEY-POSITION, @GET OR @SAVE is rejected.

If the record length is greater than 256 characters when it is written to an ISAM file, initially the following message is output

```
% EDT0914 RECORD-SIZE > 256. ONLY 256 CHARACTERS WILL BE WRITTEN
```

and then

```
% EDT0296 OVERWRITE FILE? (Y=YES; N=NO)
```

If Y is input, the file is overwritten. Only 256 characters are written in each case. The rest of the data record is overwritten with undefined information.

If N is input, the previous message EDT4981 is output.

Note

The OVERWRITE operand in the @SAVE statement also causes the file to be overwritten (in batch mode too).

ISAM files with a block size greater than 1

```
Assignment:  /CREATE-FILE FILE-NAME = filename, SUPPORT = PUBLIC-DISK( -
/   SPACE = RELATIVE(PRIMARY-ALLOCATION = pages)
/SET-FILE-LINK LINK-NAME = EDTISAM, FILE-NAME = filename, -
/   BUFFER-LENGTH = STD(SIZE = blocksize)
```

For ISAM files, the block size may be set to a multiple of the standard block size. In this case, a primary allocation of at least twice this block size must be specified by means of CREATE-FILE.

ISAM files which can be processed with the file link name EDTSAM:

- ISAM files with variable record format and a key position \neq 5
- ISAM files with a fixed record length and a key position $>$ 1
- ISAM files with a key length $>$ 8
- ISAM files with a non-numeric ISAM key

```
Assignment:  /SET-FILE-LINK LINK-NAME = EDTSAM, FILE-NAME = filename, -
/   ACCESS-METHOD = ISAM(KEY-LENGTH = keylength, -
/   KEY-POSITION = keypos), -
/   RECORD-FORMAT = format(RECORD-SIZE = reclength)
```

with the *format* FIXED for fixed-length records and the *format* VARIABLE for variable record format.

Processing:

@READ '/' or

@WRITE '/'

In this case, the line numbers are generated on the basis of the current line number and the current increment, and the ISAM key is regarded as part of the record and placed in the work file as such. If the ISAM key is modified, the record order must be the same as the order of the ISAM keys, since @WRITE '/' will otherwise be rejected with an error message.

3.5.2 Processing SAM files with nonstandard attributes

General processing sequence:

- The file link name EDTSAM is assigned to the file. Assigning the name EDTSAM and specifying the nonstandard attributes are described in the individual steps below. If a new file is to be created, all file attributes must be specified.
- Processing with @READ or @WRITE
- It is advisable to cancel the file link name assignment by means of REMOVE-FILE-LINK LINK-NAME = EDTSAM after the file has been processed.



ISAM files with nonstandard attributes can also be processed directly with @OPEN format 2 and the operand TYPE=CATALOG. In this case, the attributes are adopted directly from the catalog, and it is not necessary to assign a file link name.

SAM file with fixed record length

Assignment: /SET-FILE-LINK LINK-NAME = EDTSAM, FILE-NAME = filename, -
/ RECORD-FORMAT = FIXED(RECORD-SIZE = reclength)

If the record length is greater than 256 characters when it is written to an SAM file, initially the following message is output

```
% EDT0914 RECORD-SIZE > 256. ONLY 256 CHARACTERS WILL BE WRITTEN
```

and then

```
% EDT0296 OVERWRITE FILE? (Y=YES; N=NO)
```

If Y is input, the file is overwritten. Only 256 characters are written in each case. The rest of the data record is overwritten with undefined information.

If N is input, the previous message EDT4981 is output.

Note

The OVERWRITE operand in the @WRITE statement also causes the file to be overwritten (in batch mode too).

SAM files with a block size greater than 1

Assignment: /CREATE-FILE FILE-NAME = file name, SUPPORT = PUBLIC-DISK(-
/ SPACE = RELATIVE(PRIMARY-ALLOCATION = pages))
/SET-FILE-LINK LINK-NAME = EDTSAM, FILE-NAME = filename, -
/ BUFFER-LENGTH = STD(SIZE = blocksize)

For SAM files, the block size may be set to a multiple of the standard block size. In this case, a primary allocation of at least twice this block size must be specified by means of CREATE-FILE.

SAM file on magnetic tape

Assignment: /CREATE-FILE FILE-NAME = filename, SUPPORT = TAPE(-
/ VOLUME = vsn, DEVICE-TYP = devtype)
/SET-FILE-LINK LINK-NAME = EDTSAM, FILE-NAME = filename

3.6 Processing POSIX files

The functions for processing POSIX files are supported as of BS2000/OSD V2.0. POSIX and the associated runtime system CRTE must be activated as subsystems.

3.6.1 POSIX in BS2000

The increasing degree of networking of heterogeneous computer systems and of distributed processing within these networks require the standardization and openness of the networked computer systems and their interfaces. These interfaces must comply with the POSIX/XPG4 standards. The BS2000/OSD V2.0 operating system supports these POSIX/XPG4 standards with the software product “POSIX”.

POSIX (**P**ortable **O**pen **S**ystem Interface for **U**NIX) and XPG4 (**X**/Open **P**ortability **G**uide **V**ersion **4**) are a series of UNIX-based standards. POSIX is used to denote both these standards and the BS2000 software product.

The POSIX software product makes BS2000 an open system. Applications complying with the standard can be ported between the BS2000 and other systems that support POSIX, especially UNIX/SINIX.

The POSIX file system is a file system in BS2000 with the structure of a UNIX file system (UFS). It is hierarchical in structure and consists of files (POSIX files) and directories. POSIX users can create and process POSIX files. From within the POSIX file system, POSIX users can access remote UNIX file systems. Conversely, a user on a remote UNIX system can access the local POSIX file system.

POSIX can be accessed by all BS2000 users; even users on a UNIX system can access POSIX on a BS2000 system (via rlogin or emulation). Access control is handled entirely by the BS2000.

For further information on POSIX in BS2000, refer to the manual “POSIX Fundamentals for Users and System Administrators” [15] and “POSIX Commands” [16].

3.6.2 EDT and POSIX

Files stored in the POSIX file system can be read into EDT with the statements @XOPEN and @XCOPY and written back to the POSIX file system with the statements @XWRITE and @CLOSE.

Conventions for naming files

EDT can process file names and path names only up to a maximum length of 256 characters. If the path name is longer, the user must first move to a subdirectory within the POSIX shell using the cd command.

The name of a POSIX file is defined as follows:

xpath::= chars | .str-var (see also operand description).
 A character string not exceeding 256 characters in length
 and specifying the name of a POSIX file (perhaps with directory).

Non-printable characters, blanks and other delimiter characters may be used in a file name only if specified in str-var.

When entering a name containing lowercase letters at a terminal operating in L mode, it is first necessary to activate @LOWER ON or @PAR LOWER=ON.

EDT does not position itself in the POSIX file system. A file name always refers to the current directory, unless it begins with /. In this case, the name refers to the root directory.

Record length

EDT reads data in character by character and recognizes the end of the record by means of the record-end characters X'15' or X'0A'.

Permitted record length: 1 through 256 characters

Character strings exceeding 256 characters in length are truncated at the 256th character, and the error message % EDT1253 (SOME) RECORD(S) TRUNCATED is issued.

Character strings with a length of 0 cannot be represented in the EDT data area and must be handled specially.

Depending on the setting of the AUTOFORM mode (see @BLOCK), EDT proceeds as follows when reading in the data:

- AUTOFORM switched off:
character strings of length 0 are ignored; no record is created.
- AUTOFORM switched on:
each blank receives an end-of-line character X'0D' and is created in the data area.

Analogously, the setting of the AUTOFORM mode is evaluated when writing a data line containing X'0D':

- AUTOFORM switched off:
data lines containing X'0D' are written as such in the POSIX file.
- AUTOFORM switched on:
data lines containing X'0D' are written as a record of length 0 in the POSIX file.

Processing data in ASCII code

EDT must be informed with the operand CODE whether the data is presently in ASCII code or is to be stored in ASCII code in the POSIX file.

A fixed conversion table is used. The table corresponds to the correlation of EDF03IRV to ISO646 international 7-bit code (equivalent to correlation of EDF041 to ISO8859-1).

With @PAR HEX=ON and the presetting @PAR CODE=ISO, the data in ASCII code in the work file can be displayed in hexadecimal format in the data window and modified.

The presetting @INPUT HEX ISO makes hexadecimal input in ASCII code possible in L mode.

3.6.3 Processing POSIX files

POSIX files can be processed with the following statements:

Function	Statement
Create a new POSIX file Read a POSIX file into the work file	@XOPEN
Copy POSIX files into the work file	@XCOPY
Create a new POSIX file by writing an existing work file into a new POSIX file Write a work file back to a POSIX file	@XWRITE
Write a work file back to a POSIX file and close the POSIX file	@CLOSE

The individual statements are described in detail in the [chapter “EDT statements” on page 161ff.](#)

3.6.4 Overwriting read-only files

Attempting to overwrite a read-only file with @XWRITE or to open one for writing with @XOPEN leads to the following message being output:

```
% EDT0244 ALLOW WRITE ACCESS FOR READ ONLY FILE? REPLY (Y=YES; N=NO)
```

If Y is input, the file is overwritten with the @XWRITE statement or the file is opened for writing with the @XOPEN statement so that the file can be overwritten with a subsequent @CLOSE statement.

If N is input, as with IDs that have no TSOS privilege, message EDT5312 is output and the file is not overwritten or opened for writing.

No overwriting is made in batch mode.

3.7 Library processing with EDT

A library is a special file with a substructure which contains library elements (members) and a directory.

An element is a storage unit in which a logically related set of data such as a file, a procedure, an object module or a source program is stored. Each element in a library can be addressed separately.

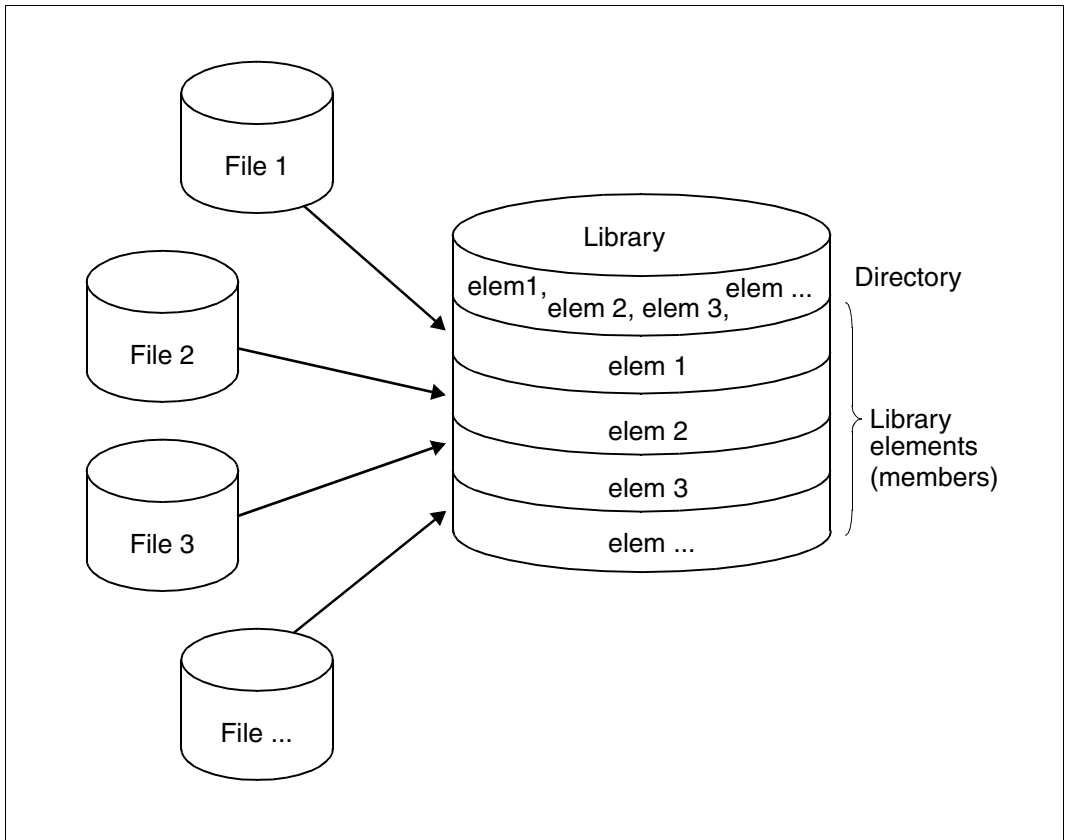


Figure 5: Structure of a library

EDT processes program libraries

Program libraries are PAM files which are processed using the program library access method PLAM. For this reason, they are also called PLAM libraries.

Program libraries offer the following advantages:

- Elements of all types can be stored in the same library.
The characteristics of program libraries permit all the data for a project, from the source programs to the object and load modules, the compilation procedures, the test data, right up to the documentation, to be held in the corresponding elements of a library.
- Elements with the same name may exist if they are distinguished by type or version designation.
- Several users may access a library, for both read and write access, at the same time.
- Storing several files as elements in one library offloads the system catalog, since each library has only one entry. This saves storage space, since a standard allocation of storage space is made only once per library, and the elements only occupy the space that they really require.

Element designation

Elements in program libraries can be addressed individually via their element designation.

The element designation consists of the name, version number and element type, and is specified as follows:

elemname[(vers)][,elemtype]

“elemname” designates the name of the library element, “vers” the version number of the element, and “elemtype” the type of the element. Specification of the version number and the element type is optional.

If no version number is specified in a statement, the element with the highest version number is selected by default. If no element type is specified in a statement, the value specified for @PAR ELEMENT TYPE is used by default (Type S).



For reasons of compatibility, the allocation of element designations is governed by the naming conventions of the software product LMS (see the “LMS” manual [14]). These must be followed, so that library elements which have been created or processed using EDT can also be managed with the software product LMS.

3.7.1 Element types supported by EDT

The element type determines the type of the stored data.

Standard and predefined types are:

Element type	Contents
S	Source programs.
M	Macros.
J	Procedures.
P	Edited data.
D	Text data.
X	Data in any format.
R	Object modules. Supported only by @DELETE and @SHOW.
C	Load modules. Supported only by @DELETE and @SHOW.
H	Compiler result information. Supported only by @DELETE and @SHOW.
L	Link load modules. Supported only by @DELETE and @SHOW.
U	IFG format masks.
F	IFG user profiles.

If, in a statement, no value is specified for “elemtype”, the value specified in @PAR ELEMENT TYPE is assumed by default.

As of EDT V16.5, freely selectable type names (user-defined types) may be used with the statements for processing library elements. No check is made on the base type.

3.7.2 Processing library elements using EDT

EDT can be used to:

- create, modify or read elements (element types S,M,J,P,D, X and corresponding free type names)
- delete elements (all element types)
- output the contents of a library (all element types).

Library elements can be processed using the following statements:

Function	Statement	Element types supported
Create a new library element. Read a library element into the work file	@OPEN, format 2	S, M, P, J, D, X, free type names
Create a new library element by writing an existing work file into a new library element. Write a work file back to a library element.	@WRITE, format 2	
Write a work file back to a library element and close the library element.	@CLOSE	S, M, P, J, D, X, free type names
Copy library elements into a work file.	@COPY, format 2	
Delete library elements.	@DELETE, format 2	S, M, P, J, D, X, free type names
Preset libraries containing the most frequently used elements.	@PAR LIBRARY	S, M, P, J, D, X, free type names
Preset the most frequently used element type.	@PAR ELEMENT TYPE	S, M, P, J, D, X, R, C, H, L, U, F, free type names
Output the directory of a library.	@SHOW	S, M, P, J, D, X, free type names

The various statements are described in detail in [chapter “EDT statements” on page 161ff.](#)



Element type X itself may be a complete library. It can still be processed by EDT but its structure will be destroyed in the process.



Delta elements cannot be processed directly. To process delta elements with LMS (as of LMS V3.0A), enter the following statement in EDT: @USE COM='!(LMSEDT,\$LMSLIB)

For a more detailed description of libraries, see the “LMS” manual [14].

3.8 SDF support for the writing of system procedures

In systems in which SDF V3.0 or higher is installed, a system procedure created in or read into an EDT work file can be checked for syntax errors and, if necessary, corrected without the user's leaving EDT.

EDT can be used to do the following:

- pass the contents of a line or a range of lines to SDF for a syntax check and, depending on the setting of the SDF options, correct faulty or missing operands of commands or statements in SDF's correction dialog (@SDFTEST and T statement code). If the correction dialog is aborted or if none is possible, the faulty line is displayed and can be overwritten at the top of the window, and an error message is issued. If the syntax was correct or has now been corrected, the commands and statements are transferred to the EDT work file.
- preset the internal name of a program (@PAR SDF-PROGRAM). If the user specifies a program name to which an SDF syntax file is assigned, the statements of the program are also subjected to a syntax check.
- display on the screen or write to a file information on SDF syntax files and the SDF options and internal program names that have been set (@STATUS=SDF).

When checking syntax, EDT distinguishes between three types of lines:

1. lines beginning with one (and only one) / in column 1
These lines are checked for command syntax in accordance with the SDF syntax file hierarchy. Their admissability in regard to privileges or system environment (e.g. batch process or procedure) is determined by the current user and the current environment.
2. lines beginning with //.
These lines are passed to SDF, where they are subjected to a statement check. The program name is preset by means of the statement @PAR SDF-PROGRAM or is known through a preceding @SDFTEST PROGRAM=name statement. The program name must be known in a current SDF syntax file.
3. lines of pure data
These lines are not checked.

SDF does not detect faulty operands in ISP commands.

3.9 Extended Host Code Support (XHCS)

Computer systems and data display terminals work with a single set of letters, numbers and characters which are used to create words and other elementary components of a language. This is referred to as the character set.

By extending these character sets, country-specific character representations, such as umlauts (in German) or accents (in French) can be provided within a character set.

A coded character set (CCS) is the unique representation of the characters in a character set in binary form. The contents of a coded character set and its specific regulations, such as the sort sequence and the conversion rules, are laid down in international standards.

Example: The character “ä” is represented in the coded character set EBCDIC.DF.03 (German reference version) by the byte X'FB' and in EBCDIC.DF.04-1 by the byte X'43'.

Each coded character set (or code) is identified by its unique name (Coded Character Set Name: CCSN).

Example: The code EBCDIC.DF.03 (international reference version) has the name “EDF03IRV”.

The coded character set name EDF04F is used for two different charactersets:

- EBCDIC.DF.04-15 (Euro character set, as per ISO8859-15)
- EBCDIC.DF.04-NAF.IND (French-Arabic alphabet with Indian digits)

EDT cannot determine which of the two character sets is meant when EDF04F is set.

As of V16.6B, EDF04F is interpreted as the Euro character set.

(An optional object correction can be provided for customers who wish interpretation as the French-Arabic character set).

You will find a list of existing codes in the “XHCS” manual [11].

3.9.1 XHCS and EDT

EDT uses conversion tables to determine non-displayable characters and for lowercase/uppercase conversion if LOWER ON is specified.

If the XHCS subsystem (eXtended Host Code Support) is available, EDT uses these conversion tables in place of the permanently defined EBCDIC.DF.03 tables.

If the XHCS subsystem is not available, the standard EDT conversion tables based on EBCDIC.DF.03 are used.

When EDT is initialized, the user standard code is set if it has been activated by means of the /MODIFY-TERMINAL-OPTIONS command. In batch or procedure mode, the CCSN of the procedure file read in with RDATA is used. In interactive mode, lowercase/uppercase conversion is effected by VTSU (MODE=LINE).

EDT can also be used to process data containing binary values or packed numbers. If conversion is performed from one CCS to another CCS, such data may be corrupted. This is why EDT does not perform any conversion. A homogeneous code environment must exist within an active EDT application, i.e. only a single CCS can be used at any one time.

If, for example, file A with the code XC1 is being processed, all work files are set to this code. Reading in (merging) other files is only permitted if the code used in these files is the same as the code used in file A. If the code used is different, @COPY, @GET, @INPUT and @READ statements are rejected. If the user wishes to read in file B with the code XC2, processing of file A must first be terminated. In other words, the work file must be cleared by means of @DELETE or (if it has been opened with @OPEN) closed using @CLOSE.

Printer output from EDT (@LIST statement and logging to SYSOUT in batch mode) is output, as before, as hexadecimal character strings without a code attribute.

Code attributes are ignored when EDT string variables (#S00-#S20) are processed.

Job variables and S variables are read and written as hexadecimal character strings without taking any code attribute into account.

Extension for the Arabic and Farsi languages

EDT supports the Arabic and Farsi languages. The major difference between the writing of these languages and that of European languages is the direction of writing (from right to left). The modified and extended functions offered by EDT to support Arabic and Farsi are described in the following manuals:

“Additional Information for Arabic“ [3]

“Additional Information for Farsi“ [4]

3.9.2 XHCS in EDT interactive mode

When EDT is called, the user standard code is set if it has been activated by means of the /MODIFY-TERMINAL-OPTIONS command. The CCSN is stored in the EDT data area and is valid for all work files.

This global CCSN remains valid until an explicit or implicit switch is made to a different CCSN. In 7-bit mode (current CCSN=EDF03IRV) the generated files or library elements are assigned the code attribute “blank” (CCSN=uuuuuuuuu)

EDT requests the table of displayable characters and the lowercase/uppercase conversion table from the XHCS subsystem. Prior to output to the data display terminal, the video buffer is converted using these tables (taking LOWER ON/OFF into account).

In the case of input/output to the data display terminal (WROUT, WRTRD, RDATA), the CCSN is included as an operand via VTSU (in the VTSUCB).

When files or library elements are generated using @WRITE or @SAVE, the CCSN is entered in the catalog or in the library, as appropriate, as the code attribute. Any existing CCSN is overwritten. In 7-bit mode (current CCSN=EDF03IRV) the standard EDT conversion tables are used.

Switching character sets

It is possible to switch from the character set selected in EDT to a different character set either explicitly or implicitly. In order for this to be done without corrupting data, the following conditions must be met:

- The EDT work files must not contain data which uses a different CCSN (i.e. all EDT work files are empty).
- The CCSN must be included in the list of CCSNs valid for the data display terminal (i.e. the data display terminal can display the character set).
- The coded character set (CCS) must not be an ISO code and must not be a 7-bit code other than EDF03IRV.

If these conditions have been met, the table of displayable characters and the lowercase/uppercase conversion table are requested from the XHCS subsystem. If these conditions have not been fulfilled, the switchover is rejected with a message, and the currently set CCSN remains valid.

An implicit character set switchover occurs when a file or library element which uses a different CCSN is read in. If the file or library element contains a CCSN with the value "blank", the CCSN EDF03IRV is assumed by default. When switching to the 7-bit coded character set (CCS) EDF03IRV, the standard EDT conversion tables are used.

An explicit character set switchover is achieved by means of the EDT statement @CODENAME.

3.9.3 XHCS in EDT procedure mode

In procedure and batch mode, EDT is started from a BS2000 procedure. This involves reading the statements with RDATA.

If the CCSN of the file or library element read in with RDATA is “blank”, the CCSN EDTF03IRV is assumed. If not, the specified CCSN is set for EDT.

Switching character sets

It is possible to switch to another CCS name in procedures or in batch operation, either explicitly with the @CODENAME statement or implicitly by reading in a file with another CCS name. The following requirement must be satisfied:

No EDT work file contains data with another CCSN (i.e. all EDT work files are empty).

Note

The CCS of the procedure or the batch file is used as the current CCS in EDT. It is therefore possible that after switching over, conflicts will occur between the data in the work file and (possibly differently coded) data from the procedure file (for example, characters in an @ON statement). EDT cannot detect such conflicts, so the user has the responsibility of avoiding them.

3.10 Job variables

In systems in which the subsystem “job variable support” has been installed, job variables (JV) can be used.

In EDT the user can:

- delete job variable entries from the catalog (@ERAJV),
- have the values of a job variable
 - displayed on the screen
 - written to a work file
 - assigned to a character string (@GETJV)
- enter job variables in the catalog (@SETJV)
- allocate values to job variables (@SETJV) and
- have information on job variables
 - output on the screen
 - written to a work file (@STAJV).

An EDT session can be monitored using a monitor job variable (see [section “Monitoring an EDT session with monitoring job variables” on page 41](#)).

For further information on job variables, see the “Job Variables” manual [12].

3.11 SDF-P support

In systems where the SDF-P subsystem has been installed, S variables can be used.

In EDT, the user can:

- display on the screen or
- assign to a character string (@GETVAR)
 - the contents of STRING and INTEGER-type S variables
- declare S variables (@SETVAR)
- assign values to S variables (@SETVAR) or
- read in (@GETLIST)
- extend (@SETLIST)
- re-write (@SETLIST] or
- delete (@SETLIST MODE=OVERWRITE)
 - the contents of composite LIST-type S variables (list variables). The elements of the list variables must be of the STRING type.

As of BS2000/OSD-BC V1.0, regardless of whether EDT is terminated normally using @HALT, @RETURN or, in interactive mode, using @END or is terminated abnormally, EDT provides a command return code that can be used by SDF-P for controlling S procedures (see [section “EDT command return code” on page 39](#)).

For more detailed information on S variables, see the “SDF-P” manual [13].

3.12 Task switches

There are four task switches which EDT evaluates for runtime control. These switches can be set or reset by means of the system command MODIFY-JOB-SWITCHES before EDT is started. During an EDT session, the switches can be set by means of the @SETSW statement.

Task switch 4

Interactive or batch mode:

If task switch 4 is set before EDT is loaded, message BLS0500 (which normally appears after EDT has been loaded) is suppressed, as is the message % EDT8000 EDT NORMAL END when EDT is terminated. The following messages are likewise suppressed: % EDT0900 EDITED FILE(S) NOT SAVED! and % EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)

Batch mode:

If task switch 4 is set before EDT is loaded, @LOG NONE is set, i.e. nothing is logged during the EDT session.

Task switch 5

If task switch 5 is set before EDT is loaded, then EDT is set to L mode. It reads its inputs from SYSDTA with RDATA. The same effect (reading from SYSDTA with RDATA) can be achieved by entering @EDIT ONLY on the screen. Instead of the current line number in interactive mode, EDT displays *.

If task switch 5 is set, the compatible syntax check of L mode is preset (see @SYNTAX SECURITY=LOW).

Changing the setting of this task switch during the EDT session has no effect.

@EDIT without operands switches to L mode, and @EDIT FULL SCREEN switches to F mode.

Task switch 6

Normally, EDT prints only 132 characters per line, but if task switch 6 is set it prints up to 160 characters per line. In both cases (132 or 160), any output which exceeds the line capacity is printed in the following lines.

If task switch 6 is to be set, this must be done before EDT is loaded.

Task switch 7

This switch may be set before or during the EDT session and prevents EDT from automatically releasing previously allocated storage space which is no longer needed. Normally, EDT releases unoccupied storage space via FILE macro (see [section “File processing” on page 49ff](#)).

3.13 Data protection

There are two ways of protecting your system against unauthorized access via EDT:

- EDT can only be started if the user ID has the required privilege
- use of uninterruptible BS2000 system procedures which check which EDT statements are called

3.13.1 Constraints on privileged user IDs

The START-EDT command can be entered in all user IDs with the privilege TSOS and/or STANDARD-PROCESSING. If a given user ID has only one or more of the following privileges, EDT will be started but any, security-relevant statements will be rejected.

Privilege	Meaning	System ID
HARDWARE-MAINTAINACE	Hardware online maintenance	\$SERVICE
SECURITY-ADMINISTRATION	Security administration	\$SYSPRIV
SAT-FILE-MANAGEMENT	Management of SAT files	\$SYSAUDIT
SAT-FILE-EVALUATION	Evaluation of SAT files	\$SYSAUDIT

Table 1: User IDs with special privileges

The following statements are security-relevant for user IDs with these privileges:

Statement	Meaning
@EXEC	Start program
@LOAD	Load program
@RUN	Run user program as subroutine
@SYSTEM	Issue system commands
@UNLOAD	Unload program
@USE	Define external statement routines

Table 2: Security-relevant statements

If used with the specified user IDs, these statements are rejected with the error message %EDT4976: STATEMENT INHIBITED FOR USER

3.13.2 Uninterruptible procedures

If BS2000 system procedures are protected against interruption by the caller by means of INTERRUPT-ALLOWED=NO, the following condition applies to EDT as of SDF-P V2.0:

- It is not possible to switch to system mode by means of `K2`

If EDT procedures are aborted by means of `K2`, EDT issues message EDT0913 to inquire whether any actions are to be performed.

In interactive mode and in the case of input from a file (read with RDATA from SYSDTA, processing of a start procedure), the security-relevant statements @SYSTEM, @EXEC, @LOAD, @RUN, UNLOAD and @USE are rejected.

Exception: SYSDTA=SYSCMD applies.

4 EDT operating modes

EDT offers two operating modes for processing data:

- In FULL SCREEN mode (F mode), the whole screen is available in 10 work files (0-9) for the input of data and statements.
- In LINE mode (L mode), there are 23 work files (0-22) but only one screen line is ever available for the input of data and statements.
Statements must be entered with a preceding @ in order to distinguish them from records.

4.1 F mode

In FULL SCREEN mode (F mode), EDT provides screen-oriented file processing for SAM and ISAM files and for elements of program libraries and for POSIX files. A total of 10 work files (0-9) are available for this purpose.

Screen-oriented processing means that, in the file section displayed on the screen:

- data may be overwritten in any desired order,
- text can be deleted and inserted without having to worry about the record structure.

In addition to carrying out changes directly on the screen, the user can control file processing by means of:

- statements entered in the statement line
- statement codes entered in the mark column
- statements entered in the data window
- record marks
- function keys

The formatted screen output is called the work window. This displays the data of the work file, which was entered by inputs on the screen or by reading the contents of SAM or ISAM files or library elements or POSIX files into this work file.

It is possible to switch from F mode to L mode (see @EDIT).

Special features of the 3270 Data Display Terminal

EDT is designed for use with Siemens 8160 and 9750 Data Display Terminals (including upwardly compatible devices) and their characteristics.

The device characteristics of the 3270 Data Display Terminal are very different from those of the 8160/9750, with the result that modification of EDT for use with the 3270 must be less than perfect from an ergonomic point of view.

When used in conjunction with the 3270 Data Display Terminal, the different device characteristics mean that the full range of EDT functions cannot be used and that there are a number of minor irregularities at the screen interface.

This is due primarily to the following device characteristics:

- The device control characters (ASZ, FBZ) occupy a visible byte (blank) on the screen, and this reduces the possible number of characters in a line.
- There are no null characters (X'00') on the screen display. Blanks (X'40') are displayed instead. The null characters displayed as blanks are not passed to the computer.
- The character X'44' (Japanese currency symbol) is used as a substitute symbol for codings which cannot be displayed.
- The keys field mark and DUP are irrelevant for EDT and are rejected in F mode with a question mark (?). In L mode, the character X'44' is passed to the computer.

4.1.1 Work window

The work window divides the screen into fields with different functions. The following diagram shows the structure of the work window.

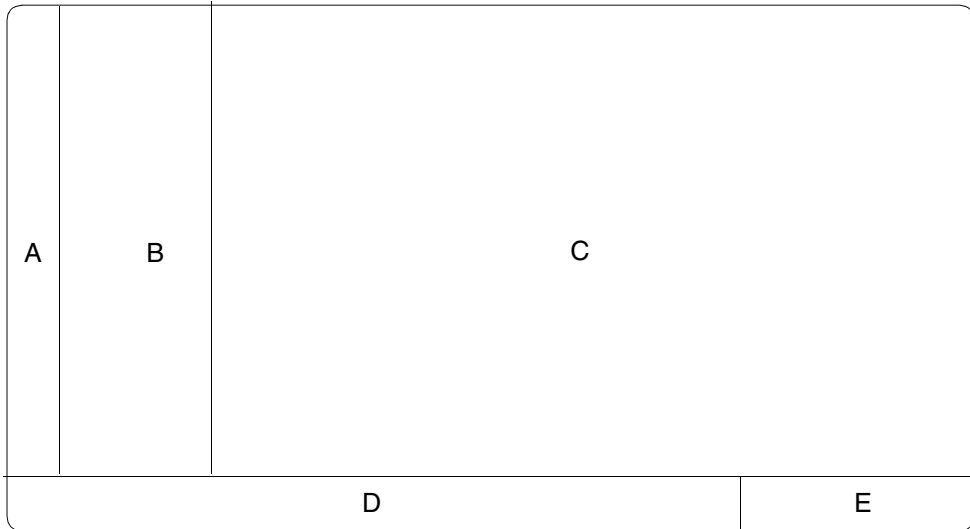


Figure 6: Standard work window format with line number display active

- A = Mark column
- B = Line number display
- C = Data window
- D = Statement line
- E = Status display

Mark column

Functions can be initiated by entering single-character statement codes in the mark column.

When records are displayed in the data window, the default setting is that the mark column is overwritable and the data window is protected against overwriting. The data window lines are set to overwritable only when a statement code is entered in the mark column or when data is sent off using **F2**. After this, it is no longer possible to enter statement codes in the overwritable lines.

The statement @PAR EDIT FULL=ON can be used, provided the line number display is active (@PAR INDEX=ON), to set the data window and the mark column to overwritable. It is possible to mark a line and at the same time modify data in this line (see @PAR EDIT FULL).

Invalid entries in the mark column can be deleted by overwriting with blanks or null characters.

Line number display

When EDT is called, the line number display is active by default. It can be suppressed by means of @PAR INDEX=OFF.

With the exception of the first column, which is also the mark column, the line number display cannot be overwritten.

The line number is displayed as 6 digits, 4 digits before and 2 digits after the decimal point.

The entire line number, with 4 digits after the decimal point, can be viewed only in L mode.

Data window

The current work file is displayed in the data window. A work file consists of records and these records are displayed in the lines of the data window. A record may be longer than a data window line; in this case, only part of the record is visible in the data window. The data window displays only part of the work file, but it can be positioned anywhere within the file.

Records which are longer than a data window line can be displayed fully in EDIT LONG mode (see @PAR EDIT LONG).

If the file contains fewer records than the data window has lines, the remaining lines are filled with the filler character (default: NIL character) and set to overwritable.

When EDT is called, the empty work file 0 is displayed on the screen.

By default, the records in the data window are not overwritable. To change this, the records must be marked in the mark column or the entire screen must be set to overwritable by means of **F2**. In EDIT FULL mode, which is set by means of @PAR EDIT FULL = ON, all records in the data window can always be overwritten. Statement codes can also be specified simultaneously in the mark column (see @PAR EDIT FULL).

F mode requires no end-of-record character in order to detect the end of the record. The end of the record is the last character in the record which is not null or a filler. Null or filler characters at the end of a record are ignored.

Regardless of whether the screen is sent off using **F2**, **DUE** or **DUE1**, all inputs in the data window are transferred to the work file.

Filler characters

The filler character can be defined via the statement @SYMBOLS FILLER = and is entered between the end of the record and the end of the screen line. Unless otherwise specified, the filler character is a null character (see @SYMBOLS).

Treatment of filler characters in the data window

When a record is input (by typing into an empty file, adding records to the end of the file, entering text in lines which have been inserted by marking lines), null characters before or between other characters are converted into blanks.

When existing records are modified, filler characters within a record are passed to the file as blanks.

In EDIT LONG mode or in HEX mode, filler characters within a line are passed to the file. Filler characters at the end of a line are ignored.

Screen lines consisting only of filler characters other than '␣' are not added to the file. A record can therefore be deleted by overwriting the part that is visible in the data window with filler characters.

Screen lines consisting only of the filler characters '␣' are created as records consisting of 2 blanks.

Treatment of null characters in the data window

If a line contains only null characters, no record is created for this line in the file.

During input of a record (by entering text in an empty file, appending records to a file, or entering text in lines which have been inserted by marking existing lines) null characters before or between other characters are converted into blanks.

When existing records are modified and the filler character is not the null character, null characters within a record are treated as editable characters, i.e. X'00' is displayed as a null character on the screen and is placed unchanged in the file when the screen is sent off. In EDIT LONG or HEX mode, null characters are also left unchanged in any new lines created in the file.

Null characters at the end of a line are ignored. If a record is longer than a screen line, any null characters at the end of the screen line cause the remainder of the text to be moved up to the position after the last character which is not null, thus shortening the record.

When deleting a complete record, the use of `LZE` and `LZF` is subject to the following restrictions:

- `LZE` deletes all characters in the record as of the specified position.
- `LZF` deletes only the rest of the line; any characters in the subsequent record are moved up.

A complete record starting in a column position other than 1 must be deleted explicitly by means of `@DELETE` or with the statement code D.

`@SYMBOLS FILLER = '␣'` is used to set the form of display valid up to EDT V16.2.

Nondisplayable characters in the text

If a file contains characters which cannot be displayed on the screen, they are displayed as smudge characters.

If such a record is modified, the original character, not the smudge character, is placed in the file. If the position of the smudge character is changed by inserting or deleting characters (`EFG`/`AFG`), the smudge character is replaced by a question mark, and the line is displayed in protect mode with a '?' in the mark column. The original contents of the record remain intact.



In LOWER OFF mode, lowercase letters in the file are also displayed as smudge characters, in order to remind the user that he/she has selected the “wrong” mode. Texts which contain nonprintable characters should be entered in hexadecimal mode (see `@PAR HEX`) or in code mode (see `@CODE`).

Statement line

Inputs in the statement line are interpreted as statements. An overview of the F mode statements can be found in [chapter “EDT statements” on page 161ff](#). The EDT statement escape symbol `@` need not be entered.

The user may enter a single statement or several statements (a statement sequence) in the statement line. If several statements are entered, they must be separated from each other by semicolons (;). If an error occurs during execution of these statements, processing is aborted, an error message is issued and that part of the input which has not yet been processed - including the statement which caused the error - is displayed again.

After successful execution, the statement line on the screen is cleared. However, the last statement entered can be displayed again by entering `#` and can be executed again with or without modification. Note, however, that at least one character must be overwritten in the statement or appended to the statement. The contents of the statement line from the current cursor position to the end can be deleted by hitting `LZF`.

If the semicolon (;) appears within a string enclosed in single quotes, then it is not regarded as the statement delimiter.

If, as part of a statement sequence, @EDIT is used to switch to L mode, then any statements after this @EDIT will not be executed.

Statement line continuation

If, when the screen is sent off, the last character in the statement line is not null or blank, EDT assumes that the user needs a continuation line for this input and displays a second line. The contents of the statement line are placed in the preceding line and the statement line, which is now empty, is offered as a continuation line.

A maximum of two continuation lines are offered, which means that the maximum length of the input in the statement line is 198 characters.

Treatment of blanks in the statement line

Leading blanks before statements and blanks between keywords (operands) are ignored. Blanks are not permitted within keywords.

Treatment of null characters in the statement line

A null character in the last position of the statement line indicates the end of the statement input. Before the input is analyzed, any null characters in the input are converted to blanks.

Status display

The status display shows, from left to right:

- the line number of the first line in the work window (6 digits)
- the column number at which the display of records in the data window begins (3 digits)
- the number of the work file currently displayed (in parentheses)

The status display cannot be overwritten.

Example

```
..... 0008.00:001(3)
```

Line 8.00 is the first line of work window 3.

Modifying the work window

The user can modify the format of the work window by

- suppressing the line number display
- splitting the screen into two work windows.

Suppressing the line number display

@PAR INDEX=OFF extends the work window to the full 80 characters per screen line (3270 Data Display Terminal: 77 characters). The first character in each line is overwriteable and corresponds to the mark column.

Splitting the screen

If the screen is divided into two work windows (see @PAR SPLIT), then this is called a split screen. The upper work window is called work window 1 and the lower one is called work window 2. A work window must consist of at least two lines, one of which is the statement line.

Example

```
par split 10 $3 ..... 0000.00:001(0)
```

The screen is to be split into two work windows. To do this, @PAR SPLIT 10 \$3 is entered in the statement line and the **DUE** key is pressed.


```

1.00 BERGER THOMAS 10. HIGH ST. DONCASTER.....
2.00 DUCK DONALD 8. WALT ST. DISNEYLAND.....
3.00 GREEN JENNIFER 16. LOW RD. POUGHKEEPSIE.....
4.00 HOPPER LARRY P.O. BOX 99 MUNICH.....
5.00 STUBBS MANUELA 3. POST ST. GRANTHAM.....
6.00 .....
7.00 .....
8.00 .....
9.00 .....
10.00 .....
11.00 .....
12.00 .....
13.00 .....
.....0001.00:001(0)
1.00 YOU CAN NOW PROCESS.....
2.00 WORK FILES 0 AND 3.....
3.00 EITHER ALTERNALLY.....
4.00 OR SIMULTANEOUSLY.....
5.00 .....
6.00 .....
7.00 .....
8.00 .....
9.00 .....
.....0001.00:001(3)

```

The upper work window (work window 1) is the one in which @PAR SPLIT was entered, reduced in size to provide space for work window 2.

The lower work window (work window 2) is the new window. It comprises 10 lines, including the statement line, and displays the contents of work file 3 (@PAR SPLIT 10 \$3).

Processing sequence

Processing sequence with one work window:

1. Evaluation of the data window
2. Statement codes in the mark column
3. Statement(s) in the statement line

Only the data window and the mark column are evaluated as long as there are change or insert marks in the mark column or as long as the permanent insert function ([section "n/l Insert lines" on page 100ff](#)) is active. The records in the data window are first transferred to the file and the mark column is then evaluated. The contents of the statement line remain unchanged and are evaluated only when no further insert or change marks are specified or when the permanent insert function is switched off.

Processing sequence with two work windows:

1. Evaluation of the data window in the upper work window
2. Statement codes in the mark column of the upper work window
3. Evaluation of the data window in the lower work window
4. Statement codes in the mark column of the lower work window
5. Statement(s) in the upper statement line
6. Statement(s) in the lower statement line

As long as there are insert or change marks in the mark column, or as long as the permanent insert function is active, the above sequence is interrupted and only the data window and the mark column of the affected work window are evaluated. If there are no further change or insert marks, or if the permanent insert function is switched off, the data window and the mark column of the other work window are evaluated.



If the screen format is reset to one work window by means of a @PAR SPLIT=OFF statement entered in the upper statement line, any statements in the lower statement line will not be executed.

4.1.2 F keys

Function	Key	F1	F2	F3	F4	F5
Transferring data lines		x	x	x	x	x
Statement codes			x		x	x
Setting marks				x		
Setting to overwritable			x		x	x
Positioning to records of the same structure depth		x				
Positioning to record marks				x		

F1 Position to records with the same structure depth

F1 positions the cursor to the next record with the same structure depth as the marked record (see the [section “+/- Position work window by structure depth” on page 114ff](#)).

F2 Modify all lines

If the screen is sent off using **F2**, the entire work window or, if appropriate, both work windows are set to overwritable when the screen is next output.

If changes have been made in the data window, or if there are statement codes in the mark column, then only these changes and the statement codes are first executed. The data window is then set to overwritable. The statement line is not evaluated at this time.

If there is no input in the mark column, the statement line is processed and the data window is then set to overwritable.

If an error occurs while the statement in the statement line is being executed, or if EDT issues an error message, the data window is not set to overwritable.

F3 Process record marks

F3 initiates the following functions:

- setting of record marks
- deletion of record marks
- positioning to records with record marks.

4.1.3 K keys

Function	Key	K1	K2	K3
Terminating F mode screen dialog and @CODE SHOW		x		
Interrupting the EDT session			x	
Restoring the screen contents				x

[K4] through [K15] have no functions.

[K1] Terminate F mode screen dialog

[K1] terminates the screen dialog, and produces the message: % EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO). If Y is entered, the screen dialog is terminated; if N is entered, the screen dialog is continued.

[K1] also terminates the display of the code table after @CODE SHOW.

[K2] Interrupt the EDT session

The EDT session can be interrupted, with a switch to system mode, by means of either @SYSTEM or [K2].

The RESUME-PROGRAM command returns EDT to F mode and the entire screen is displayed again.

If, after RESUME-PROGRAM, the work window in which the EDT session was interrupted is not displayed, or is displayed incompletely, its original contents can be restored by means of [K3].

If the START-PROGRAM or LOAD-PROGRAM command is entered or a procedure containing one of these commands is started while the EDT session is interrupted, EDT is unloaded.

[K3] Restore screen contents

If the screen contents have been shifted (e.g. by a broadcast message), the original display can be restored by means of [K3].

4.1.4 Statement codes in F mode

The statement codes - also called marks - are single-character statements which are entered as uppercase or lowercase letters in the mark column of the work window.

Syntax and semantics checking

Before the screen is processed, the syntax and semantics of the statement codes are checked. If invalid statement codes or invalid code combinations (such as M followed by C) are detected, the input is not processed. Instead, the invalid statement code is replaced by ? and the cursor is positioned to the first invalid statement code.

Processing sequence in the mark column

Depending on the function key or statement code used, distinctions are made between the following cases when processing the mark column:

1. If **F3** is used, EDT evaluates only the statement codes which may be sent off using **F3** (statement codes for setting and deleting record marks).
2. If one of the destination marks A (after), B (before) or O (on) is entered, the statement codes are evaluated in the following order:
 - the K mark
 - all D marks
 - the * mark for clearing the copy buffer
 - all C, R, M marks for copying
 - the L and U marks
 - all A, B, O marks as the destinations for copy operations.Any other statement codes are executed after this.
3. If neither an S mark nor an A, B or O mark is entered, the statement codes are processed in the following order:
 - the K mark
 - all D marks
 - the * mark for clearing the copy buffer
 - all C, R, M marks for copying
 - the L and U marks
 - all T marks for SDF syntax testing
 - all X (change), E (insert characters), n and I (insert lines) marks.

The evaluation of X, E and I and n depends on the order in which they are entered. X and E after I or n may be lost if the insert operation causes the lines containing them to be moved off the screen. No warning is issued.

The statement line is evaluated after all statement codes have been executed.

Overview of EDT statement codes

Statement code	Function
*	Clear copy buffer
A, B, O	Mark line as destination for copying
C	Mark for copying
D	Delete records
E	Insert character
J	Chain two records
K	Copy line to statement line
L	Convert marked records to lowercase
M	Copy and delete marked lines
n/l	Insert lines
R	Mark for copying (without clearing copy buffer)
S	Position work window (horizontal and vertical)
T	Syntax test by SDF
U	Convert marked records to uppercase
+ / -	Position work window (horizontal)
+ / - F1	Position work window by structure depth
X	Modify lines
D F3	Delete record mark
m F3	Set record mark

*** Clear copy buffer**

* clears the contents of a copy buffer generated by means of C, M or R.

Statement code	Key
*	<code>[DUE]</code> or <code>[F2]</code>

* is evaluated before A, B, O, C, M or R, regardless of the line in which it is entered. This means that the copy buffer is always cleared first if any line is marked with *.

If a C, M or R mark is entered in the same window as the * mark without a destination specification, then the message % EDT0292 COPYBUFFER CLEARED is not issued, since these marks write new lines into the copy buffer.

Example

See the example in the [section "R Mark for copying \(without clearing copy buffer\)" on page 104](#).

A,B,O Mark line as destination

These statement codes mark the destination for lines marked with C, M and R.

Statement code	Key
A	[DUE] or [F2]
B	
O	

- A The lines to be copied are inserted after the line marked with A.
- B The lines to be copied are inserted before the line marked with B.
- O The line to be copied or moved overwrites the line marked with O.
The following two cases must be distinguished:
- The first column of the file is the first column displayed in the data window.
The line to be copied overwrites the entire contents of the line marked with O.
 - The first column of the file is not displayed in the first column of the data window.
If the line to be copied is shorter than the line marked with O, the remainder of the line marked with O is not overwritten. This can be used to insert or append text in another line. The text in the line marked with O is overwritten (or text is appended to it) as of the active column position.

If several lines are copied, they are copied into the lines following the target line. If the end of the work file is reached as this is done, new lines are created at the end of the file.

EDT assigns line numbers to the copied lines in one of three ways (see @COPY, format 2):

1. Default numbering with an increment of 1.0000, unless some other value has been set by means of @PAR INCREMENT.
2. Numbering with the increment set by the user (@PAR INCREMENT).
3. Automatic numbering and renumbering (only if @PAR RENUMBER=ON is specified). EDT renumbers the lines automatically if the increment is too large to permit insertion of all copied lines. For further details, see @COPY, format 2, "Calculation of line numbers".



The copy operation is executed only after the C, M and R marks have been evaluated. This means that the specified destination may be before the line(s) to be copied in the same data window in a single dialog step.

Example

```

c 1.00 THE STATEMENT CODES A, B AND O MARK THE DESTINATION.....
a 2.00 FOR THE LINES MARKED WITH C, M AND R.....
  3.00 .....

```

Line 1.00 is to be copied after line 2.00. To this end, line 1.00 is marked with C in the mark column and line 2.00 with A.

```

m 1.00 THE STATEMENT CODES A, B AND O MARK THE DESTINATION.....
o 2.00 FOR THE LINES MARKED WITH C, M AND R.....
  3.00 THE STATEMENT CODES A, B AND O MARK THE DESTINATION.....
  4.00.....

```

Line 2.00 is now to be moved into line 3.00. Line 2.00 is marked with M in the mark column and line 3.00 with O.

```

  1.00 THE STATEMENT CODES A, B AND O MARK THE DESTINATION.....
  3.00 FOR THE LINES MARKED WITH C, M AND R.....
  4.00 .....

```

Line 3.00 has been overwritten with the contents of line 2.00 and line 2.00 has been deleted.

C Mark for copying

C marks lines which are to be copied to a destination marked with A, B or O. The line numbers (up to 255 lines) are stored in the copy buffer.

As soon as the destination is specified, the copy operation is executed and the contents of the copy buffer are deleted.

Statement code	Key
C	[DUE] or [F2]

EDT assigns line numbers to the copied lines in one of three ways (see @COPY, format 2):

1. Default numbering with an increment of 1.0000.
2. Numbering with the increment set by the user (@PAR INCREMENT).
3. Automatic numbering and renumbering (only if @PAR RENUMBER=ON is specified). EDT renumbers the lines automatically if the increment is too large to permit insertion of all copied lines. For further details, see @COPY, format 2, "Calculation of line numbers".

C, M and R are never executed simultaneously. Input of a combination of C, M and R in the mark column of one work window is detected by the syntax and semantics check and rejected: a "?" is displayed instead of the invalid statement code and the cursor is positioned to this code to permit correction.

A, B, O and * clear the copy buffer. A copy buffer created with C is cleared by a subsequent M or R statement code.

If the screen is split, lines can be copied from work window 1 to work window 2 in a single dialog step.

However, due to the processing sequence, two dialog steps are necessary to copy lines from work window 2 to work window 1.

The copy buffer contents can also be created by marking lines in several different work files, and the destination may also be marked in any work file.

The copy buffer contains the work file numbers and the line numbers of the lines marked with C. For this reason, the line numbers in the work files must not be changed between marking the lines with C and entering the statement code A, B or O.

Example

```

1.00 C   MARK FOR COPYING.....
2.00 =====
c 3.00 .....
4.00 C MARKS LINES WHICH ARE TO BE COPIED TO A DESTINATION MARKED WITH.....
5.00 A, B OR O. THE LINE NUMBERS (UP TO 255 LINES) ARE STORED IN THE.....
6.00 COPY BUFFER. ....
b 7.00 AS SOON AS THE DESTINATION IS SPECIFIED, THE COPY OPERATION IS.....
a 8.00 EXECUTED AND THE CONTENTS OF THE COPY BUFFER ARE DELETED. ....
9.00 .....

```

Line 3.00 is to be copied before line 7.00 and after line 8.00: line 3.00 is marked with C, line 7.00 with B and line 8.00 with A.

```

1.00 C   MARK FOR COPYING.....
2.00 =====
3.00 .....
4.00 C MARKS LINES WHICH ARE TO BE COPIED TO A DESTINATION MARKED WITH.....
5.00 A, B OR O. THE LINE NUMBERS (UP TO 255 LINES) ARE STORED IN THE.....
6.00 COPY BUFFER. ....
6.10 .....
7.00 AS SOON AS THE DESTINATION IS SPECIFIED, THE COPY OPERATION IS.....
8.00 EXECUTED AND THE CONTENTS OF THE COPY BUFFER ARE DELETED. ....
9.00 .....
10.00 .....
11.00 .....
12.00 .....
13.00 .....
14.00 .....
15.00 .....
16.00 .....
17.00 .....
18.00 .....
19.00 .....
20.00 .....
21.00 .....
% EDT5360 NO COPY. BUFFER EMPTY
..... 0001.00:001(0)

```

Line 3.00 was copied before line 7.00 but not after line 8.00. Instead, EDT has issued an error message.

Lines marked with C can be copied only to a single destination since the first copy operation also clears the copy buffer. The specification of a second destination thus resulted in an error message. If lines are to be copied several times, then they must be marked with R.

D Delete records

Any records marked with D are deleted.

Statement code	Key
D	<input type="checkbox"/> DUE or <input type="checkbox"/> F2

Example

	1.00	BERGER	THOMAS	10, HIGH ST.	DONCASTER.....
d	2.00	DUCK	DONALD	8, WALT ST.	DISNEYLAND.....
	3.00	GREEN	JENNIFER	16, LOW RD.	POUGHKEEPSIE.....
	4.00	HOPPER	LARRY	P.O. BOX 99	MUNICH.....
	5.00	STUBBS	MANUELA	3, POST ST.	GRANTHAM.....
	6.00			

Line 2.00 is to be deleted: it is marked with D in the mark column.

	1.00	BERGER	THOMAS	10, HIGH ST.	DONCASTER.....
	3.00	GREEN	JENNIFER	16, LOW RD.	POUGHKEEPSIE.....
	4.00	HOPPER	LARRY	P.O. BOX 99	MUNICH.....
	5.00	STUBBS	MANUELA	3, POST ST.	GRANTHAM.....
	6.00			

E Insert characters

E permits characters to be inserted in the line, which is set to overwritable for this purpose.

Statement code	Key
E	<code>[DUE]</code> or <code>[F2]</code>

If there are not at least 20 null characters or filler characters at the end of the line marked with E, EDT provides 20 null characters at the end of this line. Any characters in the line which disappear from the data window due to the insertion of the null characters are not lost.

The user can now insert up to 20 characters anywhere in the line (using `[EFG]`). If less than 20 characters are inserted, the part of the record which was shifted is moved back into the data window.

In EDIT LONG mode, an additional line with 80 null characters is provided in addition to the normal display of the line.

Example

```

1.00 E      INSERT CHARACTERS.....
2.00 .....
3.00 .....
4.00 E PERMITS CHARACTERS TO BE INSERTED IN THE LINE, WHICH IS SET.....
5.00 TO OVERWRITABLE FOR THIS PURPOSE.....
6.00 .....
x 7.00 IF THE END OF THE LINE MARKED WITH E CONTAINS LESS THAN 20 NULL.....
e 8.00 CHARACTERS, EDT PROVIDES 20 NULL CHARACTERS AT THAT POSITION.....
9.00 ANY CHARACTERS IN THE LINE WHICH DISAPPEAR FROM THE DATA WINDOW DUE.....
10.00 TO THIS INSERTION ARE NOT LOST.....
11.00 .....
    
```

Line 8.00 is marked with E for insertion and line 7.00 with X for updating.

```

1.00 E      INSERT CHARACTERS.....
2.00 .....
3.00 .....
4.00 E PERMITS CHARACTERS TO BE INSERTED IN THE LINE, WHICH IS SET.....
5.00 TO OVERWRITABLE FOR THIS PURPOSE.....
6.00 .....
7.00 IF THE END OF THE LINE MARKED WITH E CONTAINS LESS THAN 20 NULL.....
8.00 CHARACTERS, EDT PROVIDES 20 NULL CHARACTERS AT THAT .....
9.00 ANY CHARACTERS IN THE LINE WHICH DISAPPEAR FROM THE DATA WINDOW DUE.....
10.00 TO THIS INSERTION ARE NOT LOST.....
11.00 .....
    
```

Since there is not enough space at the end of line 8.00 for 20 characters, EDT moves the rest of the line out of the data window and displays 20 null characters.

```

1.00 E      INSERT CHARACTERS.....
2.00 .....
3.00 .....
4.00 E PERMITS CHARACTERS TO BE INSERTED IN THE LINE, WHICH IS SET.....
5.00 TO OVERWRITABLE FOR THIS PURPOSE.....
6.00 .....
7.00 IF THE END OF THE LINE MARKED WITH E CONTAINS LESS THAN 20 filler.....
8.00 (default=null) CHARACTERS, EDT PROVIDES 20 NULL CHARACTERS AT THAT.....
9.00 ANY CHARACTERS IN THE LINE WHICH DISAPPEAR FROM THE DATA WINDOW DUE.....
10.00 TO THIS INSERTION ARE NOT LOST.....
11.00 .....

```

Line 7.00 has been changed and text has been inserted in line 8.00 by means of the **EFG** key.

```

1.00 E      INSERT CHARACTERS.....
2.00 .....
3.00 .....
4.00 E PERMITS CHARACTERS TO BE INSERTED IN THE LINE, WHICH IS SET.....
5.00 TO OVERWRITABLE FOR THIS PURPOSE.....
6.00 .....
7.00 IF THE END OF THE LINE MARKED WITH E CONTAINS LESS THAN 20 FILLER.....
8.00 (DEFAULT=NULL) CHARACTERS, EDT PROVIDES 20 NULL CHARACTERS AT THAT POSIT
9.00 ANY CHARACTERS IN THE LINE WHICH DISAPPEAR FROM THE DATA WINDOW DUE.....
10.00 TO THIS INSERTION ARE NOT LOST.....
11.00 .....

```

Since less than 20 characters were inserted in line 8.00, EDT moves the remainder of the line back into the data window (as far as possible).

J Chain two records

A record which is to be chained to the preceding record must be marked with J. The marked record is deleted after chaining.

Statement code	Key
J	<input type="checkbox"/> DUE or <input type="checkbox"/> F2

If the total length of the chained record exceeds the maximum record length of 256 characters, the record is truncated to the maximum record length. In this case, the marked record is not deleted and the error message: % EDT2267 LINE TRUNCATED AFTER 256 CHARACTERS is issued. No error switch is set.

If the first record of a file is marked with J, this record remains unchanged and is not deleted.

For information on splitting records, see the [section "Statement in the data window - split record" on page 119](#).

Example

```

1.00 THE RECORD WHICH IS.....
j 2.00 TO BE CHAINED TO THE PRECEDING.....
3.00 RECORD MUST BE.....
j 4.00 MARKED WITH J. THE MARKED.....
j 5.00 RECORD IS.....
6.00 DELETED AFTER.....
j 7.00 CHAINING.....
8.00 .....
```

```

1.00 THE RECORD WHICH IS TO BE CHAINED TO THE PRECEDING.....
3.00 RECORD MUST BE MARKED WITH J. THE MARKED RECORD IS.....
6.00 DELETED AFTER CHAINING.....
7.00 .....
```

K Copy line to statement line

The contents of a line marked with K (up to 65 characters) are copied into the statement line, overwriting anything currently in the statement line.

Statement code	Key
K	<input type="button" value="DUE"/> or <input type="button" value="F2"/>

Only one line may be marked with K in any given work window. If several lines are marked with K, the input is not processed and the superfluous K marks are overwritten with ?.

At least one character in the statement line created in this manner must be overwritten, modified or added in the statement line if the text is to be used as a statement.

Example

```

1.00 THE CONTENTS OF A LINE MARKED WITH K (UP TO 65 CHARACTERS) ARE.....
2.00 COPIED INTO THE STATEMENT LINE, OVERWRITING ANYTHING IN THE.....
3.00 STATEMENT LINE.....
4.00 .....
k 5.00 SCALE ON.....

```

```

1.00 THE CONTENTS OF A LINE MARKED WITH K (UP TO 65 CHARACTERS) ARE.....
2.00 COPIED INTO THE STATEMENT LINE, OVERWRITING ANYTHING IN THE.....
3.00 STATEMENT LINE.....
4.00 .....
5.00 SCALE ON.....
6.00 .....

SCALE ON .....0001.00:001(0)

```

```

-----1-----2-----3-----4-----5-----6-----7--
1.00 THE CONTENTS OF A LINE MARKED WITH K (UP TO 65 CHARACTERS) ARE.....
2.00 COPIED INTO THE STATEMENT LINE, OVERWRITING ANYTHING IN THE.....
3.00 STATEMENT LINE.....
4.00 .....
5.00 SCALE ON.....
6.00 .....

```


L Convert marked records to lowercase

All records marked with aL are converted to lowercase notation. Conversion is analogous to @CONVERT TO=LOWER.

Statement code	Key
L	<code>[DUE]</code> or <code>[F2]</code>

Record marks (including the special marks 13, 14 and 15) are retained. Separator characters entered in an overwriteable line marked with an L are taken into account before conversion starts, i.e. only the portion preceding the first separator character is converted.

Activation of the coding function (@CODE) has no effect on data conversion.

Example

```

1.00 THE CONTENTS OF A LINE MARKED WITH L ARE CONVERTED TO LOWERCASE.....
2.00 THE CONTENTS ARE NOT AFFECTED.....
4.00 .....
L 5.00 THE CONTENTS OF A LINE MARKED WITH L ARE CONVERTED TO LOWERCASE.....

```

```

1.00 THE CONTENTS OF A LINE MARKED WITH L ARE CONVERTED TO LOWERCASE.....
2.00 THE CONTENTS ARE NOT AFFECTED.....
4.00 .....
5.00 the contents of a line marked with l are converted to lowercase.....

```

M Copy and delete marked lines

M is used to mark lines which are to be moved to a specified destination (A, B or O). The lines marked with M are then deleted. The line numbers (up to 255) of the lines to be moved are stored in a copy buffer. As soon as the destination is specified, the move operation is executed and the contents of the copy buffer are deleted.

Statement code	Key
M	<code>[DUE]</code> or <code>[F2]</code>

EDT assigns line numbers to the copied lines in one of three ways (see @COPY, format 2):

1. Default numbering with an increment of 1.0000.
2. Numbering with the increment set by the user (@PAR INCREMENT).
3. Automatic numbering and renumbering (only if @PAR RENUMBER=ON is specified). EDT renumbers the lines automatically if the increment is too large to permit insertion of all copied lines. For further details, see @COPY, format 2, "Calculation of line numbers".

C, M and R are never executed simultaneously. Input of a combination of C, M and R in the mark column of one work window is detected by the syntax and semantics check and rejected: a ? is displayed instead of the invalid statement code and the cursor is positioned to this code to permit correction.

A, B, O and * clear the copy buffer.

If the screen is split, lines can be moved from work window 1 to work window 2 in a single dialog step.

However, due to the processing sequence, two dialog steps are necessary to move lines from work window 2 to work window 1.

The copy buffer contents can also be created by marking lines in several different work files, and the destination may also be marked in any work file. Copy buffer contents created with M are deleted by a subsequent C or R.

The copy buffer contains the work file numbers and the line numbers of the lines marked with M. For this reason, the line numbers in the work files must not be changed between marking the lines with M and entering the statement code A, B or O.

If a line marked with M is subsequently marked with O, then this line is deleted.

Example

```
1.00 M IS USED TO MARK LINES WHICH ARE TO BE MOVED TO A.....  
2.00 SPECIFIED DESTINATION (A, B OR O).....  
m 3.00 .....  
m 4.00 AAA.....  
a 5.00   BBB.....  
6.00   CCC.....  
7.00 .....
```

```
1.00 M IS USED TO MARK LINES WHICH ARE TO BE MOVED TO A.....  
2.00 SPECIFIED DESTINATION (A, B OR O).....  
5.00   BBB.....  
5.10 .....  
5.20 AAA.....  
6.00   CCC.....  
7.00 .....
```

n/l Insert lines

With the aid of n/l, records can be inserted into a work file.

Statement code	Key
n	[DUE] or [F2]
l	

n The number of lines to be inserted.

$$1 \leq n \leq 9$$

The lines are inserted before the line marked with n.

l The mark l (for "Insert") activates a permanent insert function which - provided the work window is large enough - creates a 9-line area for insertions.

When these nine empty lines have been filled and the work window has been sent off by means of **[DUE]**, a new insert area is displayed. This is repeated until

- no input is written into the last insert line, or
- S is entered, or
- the insert area can no longer be displayed in the data window, either due to a change of work file or because the window has been positioned elsewhere in the current work file.

Only one l mark may be entered in a work window.

Input of an l mark during use of the permanent insert function closes the first insert area and opens a new one.

The number of empty lines specified by n is displayed before the marked line. These empty lines already have line numbers, and these are displayed if the line number display is active. The line numbers are formed by adding the selected increment value to the number of the line before the insert area. The third and fourth digits of the number of the line before the insert area are ignored. If the work file is numbered with an increment of 1, it is possible to insert 99 lines at any point in the file without having to change the original line numbers.

If the difference between two line numbers is too small to permit insertion of the specified number of lines with an increment of 0.01, the following lines are renumbered. Line numbering continues with an increment of 0.01 until the ascending order can be established again. This renumbering is retained even if only some of the available insert lines are filled with text.

If no text is entered in one of the empty insert lines, no record is created in the work file for this line. The l mark can also be sent off with **[F2]**, which causes the lines in the work window to be set to overwritable.

Example

```

1.00 n/I      INSERT LINES.....
2.00 .....
3.00 .....
4.00 With the aid of n/I, records can be inserted into a work file.....
5.00 .....
6.00 111.....
3 7.00      222.....
8.00          333.....

```

Three lines are to be inserted before line 7.00. Line 7.00 is marked with 3.

```

1.00 n/I      INSERT LINES.....
2.00 .....
3.00 .....
4.00 With the aid of n/I, records can be inserted into a work file.....
5.00 .....
6.00 111.....
6.10 .....
6.20 .....
6.30 .....
i 7.00      222.....
8.00          333.....
9.00 .....

```

```

1.00 n/I      INSERT LINES.....
2.00 .....
3.00 .....
4.00 With the aid of n/I, records can be inserted into a work file.....
5.00 .....
6.00 111.....
6.10 112.....
6.20 113.....
6.30 114.....
i 7.00      222.....
8.00          333.....
9.00 .....

```

More than 9 lines are to be inserted before line 8.00. Line 8.00 is marked with I (permanent insert function).

```

1.00 n/I      INSERT LINES.....
2.00 .....
3.00 .....
4.00 With the aid of n/I, records can be inserted into a work file.....
5.00 .....
6.00 111.....
6.10 112.....
6.20 113.....
6.30 114.....
7.00    222.....
7.10      1.....
7.20      2.....
7.30      3.....
7.40      4.....
7.50      5.....
7.60      6.....
7.70      7.....
7.80      8.....
7.90      9.....
8.00    333.....
9.00 .....
10.00 .....
11.00 .....
.....0001.00:001(0)
    
```

All 9 lines of the insert area are filled with data.

```

7.90      9.....
7.91 .....
7.92 .....
7.93 .....
7.94 .....
7.95 .....
7.96 .....
7.97 .....
7.98 .....
7.99 .....
8.00    333.....
9.00 .....
    
```

Since all lines of the insert area contain data, 9 further lines with an increment value of 0.01 are displayed as a new insert area.

R Mark for copying (without clearing copy buffer)

R is used to mark lines for copying to a specified destination (A, B, O). The copy buffer is not cleared when the lines are copied.

The line numbers (up to 255) are stored in the copy buffer and remain there until a C, M or * mark is entered. This means that the lines marked with R can be copied to several different destinations.

Statement code	Key
R	<input type="text" value="DUE"/> or <input type="text" value="F2"/>

EDT assigns line numbers to the copied lines in one of three ways (see @COPY, format 2):

1. Default numbering with an increment of 1.0000.
2. Numbering with the increment set by the user (@PAR INCREMENT).
3. Automatic numbering and renumbering (only if @PAR RENUMBER=ON is specified). EDT renumbers the lines automatically if the increment is too large to permit insertion of all copied lines. For further details, see @COPY, format 2, "Calculation of line numbers".

C, M and R are never executed simultaneously. Input of a combination of C, M and R in the mark column of one work window is detected by the syntax and semantics check and rejected: a '?' is displayed instead of the invalid statement code and the cursor is positioned to this code to permit correction.

If the screen is split, lines can be moved from work window 1 to work window 2 in a single dialog step.

However, due to the processing sequence, two dialog steps are necessary to move lines from work window 2 to work window 1.

The copy buffer contents can also be created by marking lines in several different work files, and the destination may also be marked in any work file. Copy buffer contents created with R are deleted by a subsequent C or M.

The copy buffer contains the work file numbers and the line numbers of the lines marked with R. For this reason, the line numbers in the work files must not be changed between marking the lines with R and entering the statement code A, B or O.

Example

```
1.00 R   MARK FOR COPYING (WITHOUT CLEARING COPY BUFFER).....  
r 2.00 .....  
b 3.00 R is used to mark lines for copying to a destination (A, B, O).....  
b 4.00 The line numbers (up to 255) are stored in the copy buffer and.....  
5.00 remain there until a C, M or * mark is entered.....  
6.00 This means that the lines marked with R can be copied to several.....  
7.00 different destinations.....  
8.00 .....  
9.00 .....
```

```
1.00 R   MARK FOR COPYING (WITHOUT CLEARING COPY BUFFER).....  
2.00 .....  
2.10 .....  
3.00 R is used to mark lines for copying to a destination (A, B, O).....  
3.10 .....  
4.00 The line numbers (up to 255) are stored in the copy buffer and.....  
5.00 remain there until a C, M or * mark is entered.....  
6.00 This means that the lines marked with R can be copied to several.....  
7.00 different destinations.....  
8.00 .....  
9.00 .....
```


S Position work window (horizontal and vertical)

S sets the marked line to overwritable and positions it in the second line of the work window. A column counter (scale) is displayed in the first line of the work window.

If the user writes blanks in the marked line up to the position before the desired column, then EDT positions

- the first line marked with S in the first line of the work window,
- the work window to the first column containing a non-blank character in the line marked with S.

If the line is not changed, no column positioning is executed. Blanks and any other characters which are entered do not change the original line contents. To position to a column within a sequence of blanks, a non-blank character must be entered at the desired position.

Statement code	Key
S	DUE

If the user overwrites all the text in a line marked with S with blanks, then EDT positions the work window to column 73 or 81 (for INDEX=ON/OFF, respectively). S may be combined only with K and D, and these must precede the S mark in the data window.

If other statement codes (X, E) are entered below S in the mark column, they are overwritten with ? and rejected. If there are other statement codes above S in the mark column, S is overwritten with ? and rejected.

The << statement repositions the work window to column 1 (see the [section ">/< Position horizontally within work file" on page 122ff.](#)).

Example

	1.00	SEQ.NO	ART.NO.	ART.NAME	STOCK	ORDERED.....
	2.00	1	0024	SOAP	3000	150.....
	3.00	2	0015	DEODORANT	2500	600.....
s	4.00	3	0048	PERFUME	400	60.....
	5.00	4	0003	CREME	987	555.....
	6.00	5	0091	SHAVING FOAM	350	30.....
	7.00	6	0090	AFTER SHAVE	340	30.....
	8.00	7	0092	SHAVING BRUSH	200	30.....
	9.00				

The work window is to be positioned to line 4.00 and this line is to be set to overwriteable.

	1	2	3	4	5	6	7
4.00			PERFUME	400	60.....		
5.00	4	0003	CREME	987	555.....		
6.00	5	0091	SHAVING FOAM	350	30.....		
7.00	6	0090	AFTER SHAVE	340	30.....		
8.00	7	0092	SHAVING BRUSH	200	30.....		
9.00						

The work window is now to be positioned to column 18: line 4.00 is overwritten with blanks up to the desired column.

4.00	PERFUME	400	60.....
5.00	CREME	987	555.....
6.00	SHAVING FOAM	350	30.....
7.00	AFTER SHAVE	340	30.....
8.00	SHAVING BRUSH	200	30.....
9.00		

EDT has positioned the work window to column 18.



Entering the << statement in the statement line positions the cursor back to column 1 (see the [section ">/< Position horizontally within work file" on page 122](#)).

T Syntax test by SDF

A line marked with T is passed together with its continuation lines to SDF for a check on the command or statement syntax.

Depending on the GUIDANCE mode set for SDF (GUIDANCE=MINIMEDIMAX), faulty SDF syntax causes the program to branch to SDF's guided correction dialog.

If the user aborts the correction dialog or if none is possible, the faulty line is displayed and can be overwritten at the top of the window, and an error message is issued as in the case of the @SDFTEST statement.

If the SDF syntax was correct or has now been corrected, the commands and statements are transferred to the work file.

The format is determined by the SDF setting LOGGING (see the description of the MODIFY-SDF-OPTIONS command).

This statement code is subject to the current SDF settings, which can be modified with MODIFY-SDF-OPTIONS.

Statement code	Key
T	<input type="checkbox"/> DUE or <input type="checkbox"/> F2

EDT distinguishes between three types of lines:

- lines beginning with one (and only one) / in column 1
These lines are checked for command syntax in accordance with the SDF syntax file hierarchy. Their admissability in regard to privileges or system environment (e.g. batch process or procedure) is determined by the current user and the current environment.
- lines beginning with //.
These lines are passed to SDF, where they are subjected to a statement check. The program name is preset by means of the statement @PAR SDF-PROGRAM or is known through a preceding @SDFTEST PROGRAM=name statement. The program name must be known in a current SDF syntax file.
- lines of pure data
If you specify a t, it is ignored.

Continuation lines in the input

If the last character of a line beginning with / or // is a continuation character (i.e. -) and the next line also begins with / or //, it is a continuation line and is passed to SDF as a concatenated character string. Any t mark is ignored.

Output of the checked text and continuation lines in the output

Starting with the marked line number, the text is written into the file - if necessary, in pieces.

The continuation character is written in the 72nd column. If necessary, the subsequent lines are renumbered.

Line numbers are assigned as in @COPY format 2.



- T marks must not be used in combination with positioning marks (+, - or S) or insertion marks (1-9 or I).
- If the GUIDANCE setting is MIN, MED or MAX, passwords and other operands defined with OUTPUT=SECRET-PROMPT are replaced with P.
- SDF does not detect faulty operands in ISP commands.
- A maximum of 255 continuation lines are permitted in the input or the output.
- Ampersand (&) replacement is accepted only in operand values, not in commands, statements or operand names or any part thereof.

Example 1

```
/MODIFY-SDF-OPTIONS GUIDANCE=EXPERT,LOGGING=INPUT-FORM
```

The EXPERT form of the unguided dialog is set for SDF.

```
t 1.00 /SET-JOB-STEP.....
t 2.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE2, -.....
t 3.00 / NEW-NAME=FILE3, -.....
t 4.00 / PROT=*PARAMETERS(UCCES=*READ).....
t 5.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE1, -.....
t 6.00 / NEW-NAME=FILE2, -.....
t 7.00 / PROT=*PARAMETERS(ACCESS=*READ).....
t 8.00 .....
```

SDF is to check lines 1-8 for correct SDF syntax.

```
1.00 /SET-JOB-STEP.....
2.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE2, -.....
3.00 / NEW-NAME=FILE3, -.....
4.00 / PROT=*PARAMETERS(UCCES=*READ).....
5.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE1, -.....
6.00 / NEW-NAME=FILE2, -.....
7.00 / PROT=*PARAMETERS(ACCESS=*READ).....
8.00 .....
```

```
% EDT4310 SDF: SYNTAX ERROR IN LINE 0002.000 .....0002.00:001(0)
```

The cursor is positioned at the first line of the faulty command, and the lines of the command are displayed and can be overwritten.

Example 2

```
/MODIFY-SDF-OPTIONS GUIDANCE=MINIMUM,LOGGING=INPUT-FORM
```

The guided dialog with minimum help is set for SDF.

```
t 1.00 /SET-JOB-STEP.....
t 2.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE2, -.....
t 3.00 / NEW-NAME=FILE3, -.....
t 4.00 / PROT=*PARAMETERS(UCCES=*READ).....
t 5.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE1, -.....
t 6.00 / NEW-NAME=FILE2, -.....
t 7.00 / PROT=*PARAMETERS(ACCESS=*READ).....
t 8.00 .....
```

SDF is to check lines 1-8 for correct SDF syntax.

```
SITUATION: ERROR IN PROG/S-PROC          COMMAND: MODIFY-FILE-ATTRIBUTES

-----
FILE-NAME           = FILE3
NEW-NAME            = FILE4
SUPPORT             = *UNCHANGED
PROTECTION          = *PARAMETERS (ACCESS=*UNCHANGED, USER-ACCESS=*UNCHANGED, BASI
C-ACL=*UNCHANGED, GUARDS=*UNCHANGED, WRITE-PASSWORD=*UNCHAN
GED, READ-PASSWORD=*UNCHANGED, EXEC-PASSWORD=*UNCHANGED, DES
TROY-BY-DELETE=*UNCHANGED, AUDIT=*UNCHANGED, SPACE-RELEASE-
LOCK=*UNCHANGED, RETENTION-PERIOD=*UNCHANGED)
SAVE                = *UNCHANGED
MIGRATE             = *UNCHANGED
CODED-CHARACTER-SET = *UNCHANGED

-----
NEXT = *CONTINUE
      *EXECUTE"F3" OR + OR *EXIT"K1" OR *EXIT-ALL"F1"

ERROR:  CMD0185 OPERAND NAME ,UCCES*' COULD NOT BE IDENTIFIED
```

The program branches to SDF's guided correction dialog.

SITUATION: ERROR IN PROG/S-PROC COMMAND: MODIFY-FILE-ATTRIBUTES

```
-----
FILE-NAME           = FILE3
NEW-NAME            = FILE4
SUPPORT             = *UNCHANGED
PROTECTION          = *PARAMETERS(Access=R)
```

```
SAVE                = *UNCHANGED
MIGRATE             = *UNCHANGED
CODED-CHARACTER-SET = *UNCHANGED
```

```
-----
NEXT = *CONTINUE
      *EXECUTE"F3" OR + OR *EXIT"K1" OR *EXIT-ALL"F1"
```

ERROR: CMD0185 OPERAND NAME ,ACCESS' COULD NOT BE IDENTIFIED

The error is corrected.

```
1.00 /SET-JOB-STEP.....
2.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE3,NEW-NAME=FILE4,PROTECTION= -
3.00 /*PARAMETERS(Access=*READ).....
5.00 /MODIFY-FILE-ATTRIBUTES FILE-NAME=FILE1, -.....
6.00 /                               NEW-NAME=FILE2, -.....
7.00 /                               PROT=*PARAMETERS(Access=*READ).....
8.00 .....
```

Lines 2-4 are replaced with lines 2-3.

U Convert marked records to uppercase

All records marked with a U are converted to uppercase notation. Conversion is analogous to @CONVERT TO=UPPER.

Statement code	Key
U	[DUE] or [F2]

Record marks (including the special marks 13, 14 and 15) are retained. Separator characters entered in an overwriteable line marked with a U are taken into account before conversion starts, i.e. only the portion preceding the first separator character is converted.

Activation of the coding function (@CODE) has no effect on data conversion.

Example

```

1.00 the contents of a line marked with u are converted to uppercase.....
2.00 the contents are not affected.....
4.00 .....
u 5.00 the contents of a line marked with u are converted to uppercase.....

```

```

1.00 the contents of a line marked with u are converted to uppercase.....
2.00 the contents are not affected.....
4.00 .....
5.00 THE CONTENTS OF A LINE MARKED WITH U ARE CONVERTED TO UPPERCASE.....

```


+/- Position work window

+ makes the marked record the first record in the work window.

– makes the marked line the last record in the work window.

Statement code	Key
+	<code>[DUE]</code> or <code>[F2]</code>
–	

In the mark column of one work window, + or –

- may be specified only once
- must not be specified together with another + or - or with S
- must not be specified after X, E, n or l.

The column position is not affected by these statement codes.

The character + has no effect if only the last line of data is being displayed on the screen.

The character – has no effect in the first screen of a file (at the start of the data).

+/- Position work window by structure depth

With the aid of these statement codes, the work window can be positioned to the next or previous record with the same structure depth.

The structure depth is defined as the distance between the first non-blank character and the start of the record.

If a non-blank character is defined as the structure symbol (see @PAR STRUCTURE), only those records containing at least this structure symbol are evaluated. If a blank is specified as the structure symbol, all records are evaluated. The default value of the structure symbol is @.

Statement code	Key
+	F1
-	

If no record with the same structure depth is found, the position remains unchanged.

If the marked record contains no structure symbol, the statement code is rejected with the message: % EDT5354 STRUCTURE SYMBOL 'symbol' NOT FOUND

Example

```

1.00 STRUCTURE DEPTH 1.....
2.00   STRUCTURE DEPTH 2.....
3.00     STRUCTURE DEPTH 3.....
4.00 .....
5.00   NOW STRUCTURE DEPTH 2 AGAIN.....
6.00 AND BACK TO STRUCTURE DEPTH 1.....
7.00 .....

par structure=' '.....0001.00:001(0)

```

A blank is defined as a structure symbol.

```
1.00 STRUCTURE DEPTH 1.....  
+ 2.00  STRUCTURE DEPTH 2.....  
  3.00    STRUCTURE DEPTH 3.....  
  4.00 .....  
  5.00  NOW STRUCTURE DEPTH 2 AGAIN.....  
  6.00 AND BACK TO STRUCTURE DEPTH 1.....  
  7.00 .....  
  
.....0001.00:001(0)
```

Line 2 is marked as the structure depth.

```
5.00  NOW STRUCTURE DEPTH 2 AGAIN.....  
6.00 AND BACK TO STRUCTURE DEPTH 1.....  
7.00 .....  
  
.....0005.00:001(0)
```

F1 positions the cursor to the next record with the same structure depth.

X Modify lines

X is used to mark lines which are to be modified. EDT sets these lines to overwritable and displays them with high intensity. The cursor is positioned to the beginning of the first overwritable line within the data window.

Statement code	Key
X	<code>[DUE]</code> or <code>[F2]</code>

Changes are effective in the parts of the records displayed in the data window; the remainder of each record remains unchanged. However, the user should note that inserting by means of `[EFG]` will cause any characters which are shifted past the right-hand edge of the data window to be lost. This can be avoided by using the statement code E to insert characters.

Lines which, after modification, contain only null characters (as a result, for example, of `[LZF]` in column 1) are deleted from the work file.

If scrolling statements (+,-, ...) are entered in the statement line, any modification statement codes in the mark column are executed first.

The contents of the work file can also be modified with the aid of statements in the statement line.

The entire data window can be set to overwritable by means of `[F2]` (see the [section "F keys" on page 83ff.](#)).

In the new operating mode, all records in the data window are always overwritable (see `@PAR EDIT FULL`).

Example

```
x 1.00 BERGER THOMAS 10, HIGH ST. 9876 DONCASTER.....  
2.00 DUCK DONALD 8, WALT ST. DISNEYLAND.....  
3.00 GREEN JENNIFER 16, LOW RD. 5532 POUGHKEEPSIE.....  
4.00 HOPPER LARRY P.O. BOX 99 8000 MUNICH.....  
5.00 STUBBS MANUELA 3, POST ST. 5217 GRANTHAM.....  
6.00 .....
```

Line 2 has been marked for modification.

```
1.00 BERGER THOMAS 10, HIGH ST. 9876 DONCASTER.....  
2.00 DUCK DONALD 8, WALT ST. 3333 DISNEYLAND.....  
3.00 GREEN JENNIFER 16, LOW RD. 5532 POUGHKEEPSIE.....  
4.00 HOPPER LARRY P.O. BOX 99 8000 MUNICH.....  
5.00 STUBBS MANUELA 3, POST ST. 5217 GRANTHAM.....  
6.00 .....
```

The line to be modified is displayed with high intensity and the zip code is inserted before DISNEYLAND.

D Delete record mark

D F3 deletes any existing record mark (see [“Deleting record marks” on page 136](#)).

Statement code	Key
D	F3

m Set record mark

The record mark m is set in the specified record.

The data window can be positioned to this record mark (see the [section “+/- Position within work file” on page 121](#)).

Statement code	Key
m	F3

m The number of the record mark, where $1 \leq m \leq 9$ (see [“Setting record marks” on page 136](#)).

m cannot be specified as an integer variable.

@ON, format 4, can also be used to set record marks.

4.1.5 Statement in the data window - split record

@PAR can be used to define a freely selectable record separator (such as @PAR SEPARATOR = ';', see @PAR).

If this record separator is entered in a line in the data window, the record is split at this point. Several separators may be entered in one record. The first part of the original record keeps the original line number and all following parts are inserted as new records. The lines are numbered as described for @COPY, format 2.

When inserting the record separator, the user should take care that no characters are lost at the end of the line.

A record is split only when new records or separators are entered, or when existing records or separators are modified. For example, if a record is inserted by copying, at least one character must be overwritten, modified or inserted before the record can be split using record separators.

The characters NIL (null) and AM cannot be defined as record separators.

Different characters should be selected as the record separator and the tab character.

The record separator must not be redefined while the code function is active.

Empty records (record length = 0) cannot be created; however, line numbers are reserved where such records would have been generated.

A character which is not included in the data should be used as a separator.

4.1.6 Statements in the statement line

In F mode, statements are entered in the statement line (see the [section "Work window" on page 75ff](#)). In the statement line, statements can be entered with or without the statement symbol @, since EDT interprets all entries made there as statements.

The following pages contain descriptions only of those statements which can be entered exclusively in F mode dialog. A description of the statements which can be used in both F and L mode and in EDT procedures can be found in [chapter "EDT statements" on page 161ff](#).

+/- Position within work file

The +/- statements have two formats offering the following functions:

- positioning forwards or backwards by one data window or by any desired number of lines within the current work file (format 1)
- positioning to record marks within the current work file (format 2).

+/- (Format 1) Position within the work file

These statements are used to move to the desired position within the work file.

Operation	Operands	F mode
+ -	[n]	
++ --		

- + Position forwards (towards the end of the file) in the current work file by one data window. The first record in the new window is the one which follows the last record in the old window.
- ++ Position to the end of the current work file. The last line of the data window contains the last record of the work file if the current work file contains more lines than the current work window.
- Position backwards (towards the beginning of the file) in the current work file by one data window. The last line of the new window contains the record before the first record displayed in the old window.
- Position to the beginning of the current work file. The first line of the data window contains the first record of the work file.
- n Any integer > 0: specifies a displacement for + or -.
 - +n displays the nth record, counting from the first record in the current data window, as the first record in the new data window.
 - n displays the nth record before the first record in the current data window as the first record in the new data window.

If statement codes for inserting (I, n) or updating (X, E) lines are entered in the mark column at the same time as a positioning statement, the codes are processed before the file is positioned.

+/- (Format 2) Position to record marks

This format of +/- is used to position to specific record marks.

The first record with the specified mark is displayed in the first line of the data window. Further scrolling causes the next record with the specified mark to be displayed in the first line of the data window.

If no mark is found, the position within the work file remains unchanged.

Operation	Operands	F mode
+	[[[m, ...]]]	
++		
-		
--		

The statement is sent off with **F3** .

If the statement input is terminated with **DUE** or **F2** , the parentheses must be entered.

- + Position forwards to the next record which contains a record mark.
- ++ Position to the last record in the work file which contains a record mark.
- Position backwards to the next record which contains a record mark.
- Position to the first record in the work file which contains a record mark.
- m This specifies one of the nine possible record marks to which the file is to be positioned. Several marks may be specified in one statement.
Marks with special functions (such as mark 15 for write protection) are ignored.
If m is not specified, the appropriate record (first, previous, next, last) containing any record mark is used for positioning.

Example

See the example for @ON, format 4.

>/< Position horizontally within work file

These statements are used to position horizontally within the work file, i.e. to move the data window to the right or left (towards the end or the beginning of the record, respectively).

The number of the first column displayed in the data window is shown in the status display of the work window.

Operation	Operands	F mode
> <	[n]	
<<		

> Move the work window to the right by the width of the data window within the current work file.

< Move the work window to the left by the width of the data window within the current work file.

<< Move the work window back to column 1.

n Number of columns by which the work window is to be moved by the < or > statement.

$1 \leq n \leq 184$ (for INDEX ON)

$1 \leq n \leq 176$ (for INDEX OFF)

>n moves the window number columns to the right.

<n moves the window number columns to the left.

<<n is not permitted; any attempt to specify it will be rejected with an error message.

Example

```

1.00 BERGER THOMAS 10, HIGH ST. DONCASTER.....
2.00 DUCK DONALD 8, WALT ST. DISNEYLAND.....
3.00 GREEN JENNIFER 16, LOW RD. POUGHKEEPSIE.....
4.00 HOPPER LARRY P.O. BOX 99 MUNICH.....
5.00 STUBBS MANUELA 3, POST ST. GRANTHAM.....
6.00 .....

.....0001.00:001(0)

```

The data window is moved 9 columns to the right.

```

1.00 THOMAS 10, HIGH ST. DONCASTER.....
2.00 DONALD 8, WALT ST. DISNEYLAND.....
3.00 JENNIFER 16, LOW RD. POUGHKEEPSIE.....
4.00 LARRY P.O. BOX 99 MUNICH.....
5.00 MANUELA 3, POST ST. GRANTHAM.....
6.00 .....

.....0001.00:010(0)

```

The data window now begins at column 10 of each record.

Display last statement

causes the last statement executed by EDT to be displayed again in the statement line. This does not apply to scroll statements and statements for switching to another work file.

Operation	Operands	F mode
[n]#		

n Specifies which of the preceding statements is to be displayed, i.e. the nth preceding statement.

$$1 \leq n \leq 256$$

or 1# re-displays the last statement executed by EDT in the statement line. Specifying 2# causes the statement preceding the last statement to be displayed. The # statement can be entered in subsequent dialog steps so that each time the program moves back the specified number of places in the statement buffer.

If the beginning of the buffer has been reached, the statement line remains empty. If another # statement follows, the program then returns to the most recently stored statement (the end of the buffer).

After each dialog step not ending with #, the program returns to the end of the buffer.

The statement buffer has a fixed size, i.e. the length of all the statements entered determines how many statements the program can move back through.

It makes no difference whether a given statement was entered in the upper or the lower window of a split screen.

If # is entered within a sequence of statements, processing is interrupted when # is executed, and the last statement executed is displayed.

Statements following # are no longer executed.

fwkfnr/fwkfv Switch work files

This statement causes EDT to switch to another work file.

Operation	Operands	F mode
fwkfnr fwkfv		

fwkfnr The number (0,...,9) of the work file to which EDT is to switch.

fwkfv The number of a work file variable (\$0,...,\$9) which contains the number (0,...,9) of the work file to which EDT is to switch.

There is no functional difference between fwkfnr and fwkfv.

Example

```

1.00 This statement causes EDT to switch to a.....
2.00 different work file.....
3.00 .....

7.....0000.00:001(0)
    
```

The entry 7 tells EDT to switch to work file 7.

```

.....0000.00:001(7)
    
```

EDIT LONG Display records with more than 80 characters

With the aid of EDIT LONG, the user can modify the output on the screen. For records longer than 80 characters, he/she can specify that

- the complete records are to be displayed in the data window, or
- an 80-character section of each record is to be displayed in the data window.

Operation	Operands	F mode
EDIT LONG	[ON] OFF	

ON The complete records are to be displayed in the data window.

OFF Only an 80-character section of longer records is to be displayed in the data window.

EDIT LONG mode does not have a line number display. Each record is displayed on several consecutive lines of the screen. Blanks at the end of a record are ignored. The last non-blank or non-null character in the record is regarded as the end of the record in the case of entry at the screen.

If a record finishes at the end of a screen line, one line full of null characters is displayed as a record separator.

If the entire data window is set to overwriteable by means of **F2**, and the last record in the data window is not displayed entirely, then this record cannot be overwritten. If this record is marked using the statement code X, then this statement is ignored. In order to modify this record, the data window must be positioned so that the entire record is displayed in the data window.

The mark column is the first column on the screen. Records which extend over several lines must be marked in their first line. If, during extension, a record which is shorter than 240 characters is marked with E, an entire line of null characters is also displayed.

After EDIT LONG OFF, the line number displayed remains switched off.

EDIT LONG mode is switched off by INDEX ON, INDEX OFF and HEX ON.



The statements for horizontal scrolling (>,<) are accepted, but become effective only when EDIT LONG mode is switched off.

Example

1.00	NO.	ART.NO.	ART.NAME	STOCK	ORDERED	RETAIL PRICE..
2.00	1	0024	SOAP	3000	150	1.59 DM.....
3.00	2	0015	DEODORANT	2500	600	4.38 DM.....
4.00	3	0048	PERFUME	400	60	18.60 DM.....
5.00	4	0003	CREME	987	555	2.83 DM.....
6.00	5	0091	SHAVING FOAM	350	30	4.39 DM.....
7.00	6	0090	AFTER SHAVE	340	30	10.55 DM.....
8.00	7	0092	SHAVING BRUSH	200	30	11.50 DM.....

edit long on0001.00:001(0)

The entire records are to be displayed in the data window.

NO.	ART.NO.	ART.NAME	STOCK	ORDERED	RETAIL PRICE	PU
1	0024	SOAP	3000	150	1.59 DM	1
2	0015	DEODORANT	2500	600	4.38 DM	3
3	0048	PERFUME	400	60	18.60 DM	14
4	0003	CREME	987	555	2.83 DM	2
5	0091	SHAVING FOAM	350	30	4.39 DM	3
6	0090	AFTER SHAVE	340	30	10.55 DM	9
7	0092	SHAVING BRUSH	200	30	11.50 DM	9

It can now be seen that each record contains a further field, namely PURCHASE PRICE.

HEX Switch on hexadecimal code

HEX switches on hexadecimal mode, i.e. all records are displayed in both printable and hexadecimal format on the screen:

Line 1: Data line in printable format

Lines 2,3: Data line in hexadecimal format (vertical representation)

Line 4: A column counter as for SCALE ON. The column counter switched on by means of SCALE is not displayed in HEX mode.

Hexadecimal mode applies to the current work file, regardless of the work window.

Operation	Operands	F mode
HEX	[ON] OFF	


ON Switches on hexadecimal mode.

OFF Switches off hexadecimal mode.

Hexadecimal mode is also switched off by EDIT LONG OFF.

When EDT is started, the default value is OFF.

Modifying data lines in hexadecimal mode

Hitting  in hexadecimal mode positions the cursor to the next hexadecimal line (not the next data line).

Changes may be made in both the first line (printable format) and the two lines with the hexadecimal display. If changes are made to the printable format and the hexadecimal display in a single dialog step, only the changes in the hexadecimal lines are actually used.

If invalid hexadecimal characters (characters other than 0...9, A...F) are entered in the hexadecimal lines, the invalid lines are displayed as overwritable and the invalid characters are overwritten with question marks. The cursor is located in the first invalid line. All other valid lines in the data window are set to "not overwritable".

The hexadecimal code displayed depends on the CODE setting (see @PAR CODE).

Split screen display

If the screen is split (@PAR SPLIT), the display in hexadecimal mode depends on the number of data lines (lines in the data window) on the screen in question:

Overview table:

Number of data lines		Display on the screen
1	Only	Printable format of the record
2	Line 1 : Line 2 :	Printable format of the record Column counter line
3	Line 1: Line 2. and 3.:	Printable format of the record Hexadecimal format (column counter line not displayed)

Note: the number of screen lines = number (data lines + statement line)

If the number of data lines is a multiple of 4, four screen lines per record are displayed.

If the number of data lines is not a multiple of 4, the remaining (1, 2 or 3) lines are treated as shown in the overview table.

INDEX Select work window format

INDEX is used to select the format of the work window. By default, the format is set to 72 characters per line with a 6-digit line number display. Column 1 of each line is the mark column.

Operation	Operands	F mode
INDEX	[ON] OFF	

ON Selects the work window with the default format.

OFF Selects the work window with 80 characters per line and no line number display.

If the screen is split (see @PAR SPLIT), INDEX affects only the work window in which it is entered.

INDEX switches off EDIT LONG mode.

Example

```

1.00 INDEX is used to select the format of the work window.....
2.00 By default, the format is set to 72 characters per line.....
3.00 with a 6-digit line number display.....
4.00 Column 1 of each line is the mark column.....
5.00 .....

index off .....0000.00:001(0)
    
```

The line number display is switched off.

```

INDEX is used to select the format of the work window.....
By default, the format is set to 72 characters per line.....
with a 6-digit line number display.....
Column 1 of each line is the mark column.....
.....
    
```

SCALE Display column counter

SCALE displays a column counter (line scale) in the work window. This is displayed as the first line in the work window (not valid in EDIT LONG mode).

Operation	Operands	F mode
SCALE	[ON] <u>OFF</u>	

ON The column counter appears as the first line after any information line present and displays the current column numbers of the work window (e.g. after horizontal shifting of the work window).

If a tab has been defined (see @TABS), a further screen line is displayed, in which the current positions of the tabs are indicated by “I”. The tab character is shown in the mark column position.

OFF Switches off the column counter and any existing tab display scale.

The default value is OFF when EDT is called.

If the screen is split (see @PAR SPLIT), SCALE affects only the work window in which it is entered.

The column counter is not displayed in EDIT LONG mode.

Example

1.00	BERGER	THOMAS	10, HIGH ST.	DONCASTER.....
2.00	DUCK	DONALD	8, WALT ST.	DISNEYLAND.....
3.00	GREEN	JENNIFER	16, LOW RD.	POUGHKEEPSIE.....
4.00	HOPPER	LARRY	P.O. BOX 99	MUNICH.....
5.00	STUBBS	MANUELA	3, POST ST.	GRANTHAM.....
6.00			
scale on				0001.00:001(0)

In order to check column alignment, a column counter is requested.

		1		2		3		4	
1.00	BERGER	THOMAS	10, HIGH ST.	DONCASTER.....					
2.00	DUCK	DONALD	8, WALT ST.	DISNEYLAND.....					
3.00	GREEN	JENNIFER	16, LOW RD.	POUGHKEEPSIE.....					
4.00	HOPPER	LARRY	P.O. BOX 99	MUNICH.....					
5.00	STUBBS	MANUELA	3, POST ST.	GRANTHAM.....					
6.00								

SHIH Display statement buffer

The following statement can be used to display the last executed EDT statements on the monitor. This statement is only permitted in full screen mode.

Operation	Operands	F mode
SHIH		

The buffer does not contain any scroll statements, any statements to change the work file nor the SHIH statement itself.

The size of the statement buffer is fixed. The maximum number of statements displayed depends on the length of the separate statements (see also [section "# Display last statement" on page 124](#)).

The last executed statements are written line by line into work file 9. The content of work file 9 is deleted before it is used. If output of an information line is enabled (@PAR INFORMATION=ON), a header line is output in work file 9.

Statement code K can be used to place the output line containing the desired statement into the statement line. You can then execute it in the desired work file, if you precede it with a "Switch work files" statement.

The EDT line number corresponds to the relative position in the statement buffer. You can therefore also place the statement in line n.00 with statement n# in the statement line, if you did not previously position in the statement buffer.

SPLIT Display 2 work windows

SPLIT causes a second work window to be displayed on the screen. Each work window has its own statement line.

After the screen has been split, the cursor is positioned to the upper statement line. After each subsequent output, it is positioned to the statement line in which the last statement or statement sequence was entered.

If statements are entered in both statement lines, the cursor is positioned to the upper statement line. If an error occurs during execution of a statement, the cursor is positioned to the statement line in which the incorrect statement was entered.

If, on a split screen, SPLIT OFF is entered in the upper statement line and some other statement is entered in the lower statement line, then SPLIT OFF is rejected with an error message.

The default value when EDT is started is SPLIT OFF.

Operation	Operands	F mode
SPLIT	n(fwkfnr) 0 <u>OFF</u>	

n The number of lines (including the statement line) to be displayed in the lower work window, where

$$2 \leq n \leq 22.$$

fwkfnr The number of the work file to be displayed in the lower work window. The upper work window displays the work file in which the SPLIT statement was entered.

0 The work window in which this SPLIT statement is entered is

OFF expanded to occupy the full screen (24 lines).

Example

```

1.00 BERGER THOMAS 10, HIGH ST. DONCASTER.....
2.00 DUCK DONALD 8, WALT ST. DISNEYLAND.....
3.00 GREEN JENNIFER 16, LOW RD. POUGHKEEPSIE.....
4.00 HOPPER LARRY P.O. BOX 99 MUNICH.....
5.00 STUBBS MANUELA 3, POST ST. GRANTHAM.....
6.00 .....

split 10(2) .....0001.00:001(0)
    
```

SPLIT 10(2) requests a second work window with 10 screen lines (including the statement line) and work file 2 is to be displayed in this window.

```

1.00 BERGER THOMAS 10, HIGH ST. DONCASTER.....
2.00 DUCK DONALD 8, WALT ST. DISNEYLAND.....
3.00 GREEN JENNIFER 16, LOW RD. POUGHKEEPSIE.....
4.00 HOPPER LARRY P.O. BOX 99 MUNICH.....
5.00 STUBBS MANUELA 3, POST ST. GRANTHAM.....
6.00 .....
7.00 .....
8.00 .....
9.00 .....
10.00 .....
11.00 .....
12.00 .....
13.00 .....

.....0001.00:001(0)
1.00 YOU CAN NOW.....
2.00 PROCESS WORK FILES 0 AND 2.....
3.00 EITHER ALTERNATELY.....
4.00 OR SIMULTANEOUSLY.....
5.00 .....
6.00 .....
7.00 .....
8.00 .....
9.00 .....

.....0001.00:001(2)
    
```

4.1.7 Description of the record marks in F mode

Each record in an EDT work file can be marked with one or more record marks. The record marks are stored in the virtual data area of EDT and are not visible to the user. They are not transferred to the real file when the work file is saved.

The record marks can be used for processing the work file (see @ON, +(m), etc.).

ISAM files opened in real mode by means of @OPEN, format 1, cannot be marked.

Setting record marks

Record marks can be set by means of

- the @ON statement in the statement line (see @ON, format 4)
- the functions IEDTPUT and IEDTPTM if EDT is called as a subroutine (see the “EDT Subroutine Interfaces” manual [1])
- the statement code m **F3** in the mark column. The one-character record marks (1 to 9) are entered in the mark column and sent off by means of **F3** .

Deleting record marks

The record marks in a work file can be deleted by means of @DELETE MARK (see @DELETE MARK).

@COMPARE deletes the record marks in the files which are compared.

The statement code D **F3** in the mark column deletes any existing record marks of the appropriate record.



In the preceding sections, the term “marking” has been used with respect to the mark column and the statement codes which can be entered there. This form of mark must be regarded as a function indicator; after execution of the statement code, the mark no longer exists.

4.1.8 Statements in F mode

The following statements may be used in F mode:

-- [(m)]	@GETVAR	@SETSW
+ [(m)]	@HALT	@SETVAR
- [(m)]	@INPUT ***)	@SHOW
<	@LIMITS	@SORT
<<	@LIST	@STAJV
>	@LOAD	@STATUS
#	@LOG	@SUFFIX
@:	@LOWER	@SYMBOLS
@AUTOSAVE	@MOVE	@SYNTAX
@BLOCK	@ON **)	@SYSTEM
@CLOSE	@OPEN	@TABS **)
@CODE	@P-KEYS *)	@TMODE
@CODENAME	@PAGE	@UNLOAD
@COLUMN	@PAR	@UNSAVE
@COMPARE	@PREFIX	@USE
@CONVERT	@PRINT	@VDT
@COPY	@QUOTE	@VTCSET
@CREATE (Format 1)	@RANGE	@WRITE
@DELETE	@READ	@XCOPY
@DELIMIT	@RENUMBER	@XOPEN
@DO (Format 1)	@RESET	@XWRITE
@DROP	@RETURN	@ZERO-RECORDS
@EDIT	@RUN	EDIT LONG
@ELIM	@SAVE	HEX
@END	@SEARCH-OPTION	INDEX
@ERAJV	@SDFTEST	SCALE
@EXEC	@SEPARATE	SHIH
@FILE	@SEQUENCE	SPLIT
@FSTAT	@SET	fwkfnr
@GET	@SETF	fwkfv
@GETJV	@SETJV	
@GETLIST	@SETLIST	

*) Not supported by the 3270 Data Display Terminal.

***) Not @ON, format 3.

***) Not @INPUT, format 3.

The EDT statement symbol (default value: @) may be omitted in F mode.

See [chapter "EDT statements" on page 161ff](#), for a description of the statements.

4.2 L mode

In LINE mode (L mode), files are processed line by line, i.e. EDT only offers one line at a time, in which both records and statements can be written. Records are always written in the current line. Statements are executed immediately. The statement symbol @ is used to distinguish between the two (see below).

L mode can be used in both interactive and batch mode.

The following are only supported in L mode:

- EDT procedures
- INPUT files and
- reading from SYSDTA by means of RDATA (system procedures, batch mode).

Statements from F mode are interpreted as records in L mode.

@EDIT is used to switch to L mode.

4.2.1 Input in L mode

EDT interprets an input in L mode as a statement if:

- the first character is the statement symbol (default value: @), and
- the following character is not the statement symbol.

Otherwise input is written as text to the current line.

EDT ignores leading and trailing blanks. If an input has more than one statement symbol (@@, or @ @, ...), EDT does not interpret it as a statement. In this case, the input is stored as text in the current line, and the characters (first statement symbol and blank) before the second statement symbol are truncated.

Inputs with more than one statement symbol are used to create procedures (see also the [chapter “EDT procedures” on page 141ff.](#)). Instead of being immediately executed as a statement, the input is stored as a statement, as it were, and can thus later be executed several times.

All inputs without a statement symbol are always interpreted as records and stored immediately in the current line.

4.2.2 Statements in L mode

The following statements may be used in L mode:

@	@GETLIST	@SEPARATE
@+	@GETVAR	@SEQUENCE
@-	@HALT	@SET
@:	@IF (Format 1)	@SETF
@AUTOSAVE	@INPUT	@SETJV
@BLOCK	@LIMITS	@SETLIST
@CHECK	@LIST	@SETSW
@CLOSE	@LOAD	@SETVAR
@CODE	@LOG	@SHOW
@CODENAME	@LOWER	@SORT
@COLUMN	@MOVE	@STAJV
@COMPARE	@NOTE	@STATUS
@CONTINUE	@ON	@SUFFIX
@CONVERT	@OPEN	@SYMBOLS
@COPY	@P-KEYS	@SYNTAX
@CREATE	@PAGE	@SYSTEM
@DELETE	@PAR	@TABS
@DELIMIT	@PREFIX	@TMODE
@DIALOG	@PRINT	@UNLOAD
@DO (Format 1)	@PROC	@UNSAVE
@DROP	@QUOTE	@UPDATE
@EDIT	@RANGE	@USE
@ELIM	@READ	@VDT
@END	@RENUMBER	@VTCSET
@ERAJV	@RESET	@WRITE
@EXEC	@RETURN	@XCOPY
@FILE	@RUN	@XOPEN
@FSTAT	@SAVE	@XWRITE
@GET	@SEARCH-OPTION	@ZERO-RECORDS
@GETJV	@SDFTEST	

The following statements may *not* be used in EDT procedures:

@CODENAME, @DROP

The following statements may *only* be used in EDT procedures:

@GOTO, @DO (format 2), @IF (formats 2, 3, 4), @PARAMS

The EDT statement symbol (default value: @) is mandatory in L mode.

For a description of the statements see [chapter "EDT statements" on page 161ff.](#)

5 EDT procedures

When working with EDT, it is quite possible that different files may have to be processed with identical or similar statement sequences. Such frequently used statement sequences can be combined in so-called EDT procedures and called for execution as and when necessary.

In addition to EDT statements which are used only in EDT procedures, all EDT statements which can also be entered directly in L mode are permitted.

EDT procedures can be executed:

- in work files (temporarily during an EDT session) or
- in cataloged files (SAM or ISAM files).

5.1 EDT input sources

EDT receives the statements from any of the following sources:

- directly from the screen (F or L mode dialog),
- from a cataloged file or from a library element as an @INPUT procedure, or
- from a work file as a @DO procedure.

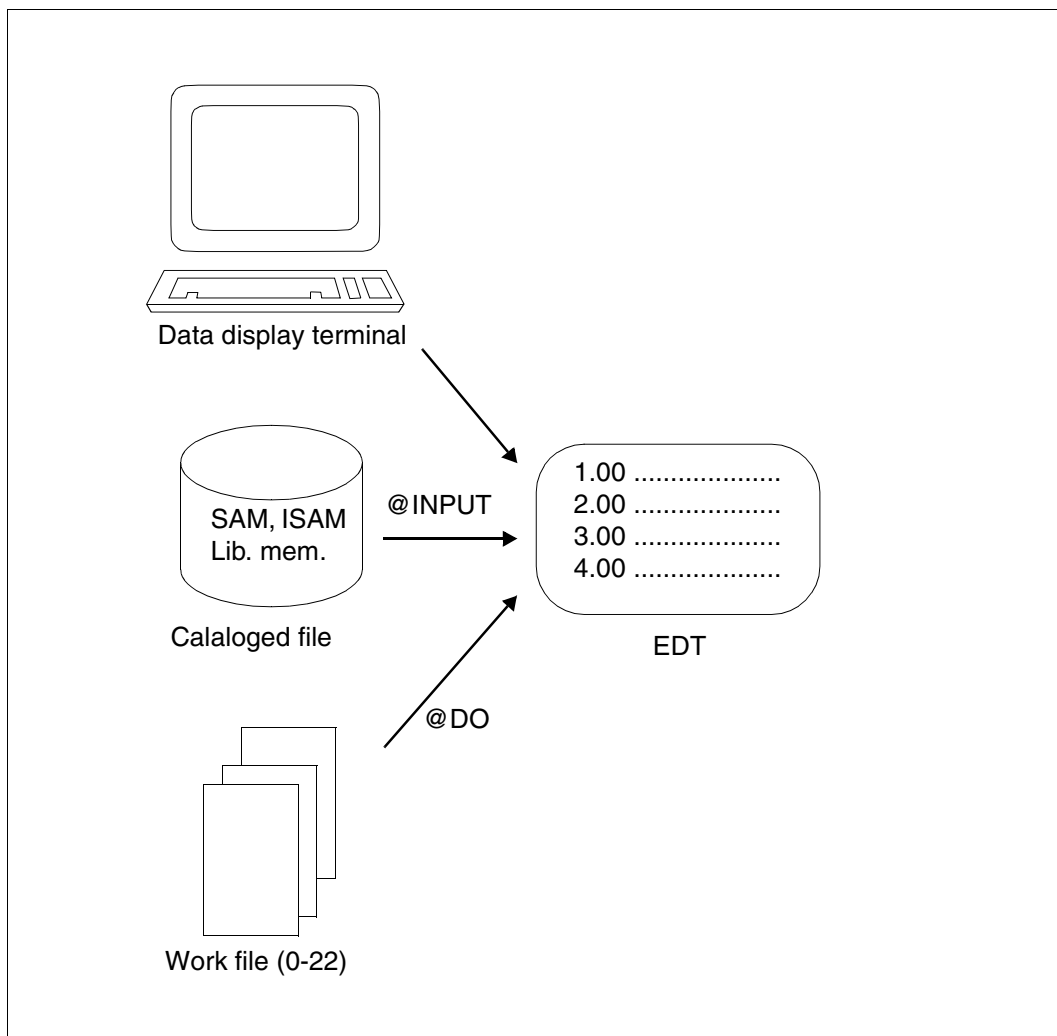


Figure 7: Statements to EDT

The input source can be changed by means of @DO, @INPUT, @EDIT, @DIALOG, @RETURN and @HALT (see the descriptions of these statements).

5.2 EDT variables

The EDT variables are used to store values. These values can be integers, character strings or line numbers. The many and varied areas of application of variables include buffering values, defining loop counters and termination conditions, entering character strings (file names, search strings, etc.) or making simple calculations.

EDT variables are only valid during the EDT run. They can be occupied, used or queried from all work files. This means that if a value is assigned to a variable in one work file, this variable is also available with the same value in another work file.

EDT offers three types of variables, which can be supplied with appropriate values. 21 variables of each type are available with the indices 0 to 20:

- integers (#I0-#I20)
- string variables (#S0-#S20)
- line number variables (#L0-#L20)

The EDT variables are supplied with values via the formats of the @SET statement or via @CREATE (see @SET, formats 1 - 5, and @CREATE).

The line number variable #L0 and the integer variables #I0 and #I1 should not be used, since in the event of a hit in the @ON statement they are overwritten with values.

Integer variables

Positive or negative integers (#I0-#I20) can be stored in the integer variables. The highest number possible is 2.147.483.647 ($2^{31} - 1$).

The integer variables can be assigned values via the @SET statement, format 1. The contents of the integer variables can be displayed on the screen by means of @STATUS.

String variables

Character strings can be stored in the string variables (#S0-#S20) in EBCDIC code. A character string can be up to 256 characters long.

The string variables can be assigned values by means of the @SET statement, formats 2 and 5, or via @CREATE. The contents of the string variables can be displayed on the screen using @PRINT.

String variables are preset to a blank when EDT is started if no values are transferred to the variables from any existing S variables SYSEDT-S00, ... SYSEDT-S20 (see [section "Calling EDT" on page 33](#)).

If the name of a file or library element in a statement can be specified both as a string and as a character variable, the variable name must be preceded by a period to ensure it is not confused with another name.

Line number variables

Line numbers can be stored in the line number variables (#L0-#L20). Line numbers may lie anywhere between 0.0001 and 9999.9999.

The line number variables can be assigned values via the @SET statement, format 3. The contents of the line number variables can be displayed on the screen by means of @STATUS.

Job variables

In systems in which the subsystem 'job variable support' is installed, job variables (JV) can be used in EDT. Unlike integer, string and line number variables, job variables remain in existence even after EDT is terminated and existing job variables can be accessed in EDT (see [section "Job variables" on page 67](#)).

S variables

In systems in which the SDF-P subsystem is installed, S variables can be used in EDT. Unlike integer, string and line number variables, S variables remain in existence even after EDT has terminated, and existing S variables can be accessed in EDT (see [section "SDF-P support" on page 68](#)).

The contents of S variables can be assigned to string variables or the contents of string variables can be assigned to S variables

- implicitly when EDT is started or terminated (see [section "Calling EDT" on page 33ff](#)).
- explicitly by means of @GETVAR or @SETVAR.

5.3 Creating, calling and executing EDT procedures

Input in an EDT procedure

The input to be processed by EDT from a procedure can take any of the following forms:

- a character string (any text).

This does not begin with the statement symbol (default value: '@'). EDT incorporates the string into the current output file as data.

If a given character string begins with more than one statement symbol (e.g. @@...) or more than one user escape symbol, EDT transfers the character string without the first statement symbol or user escape symbol, respectively, as data to the current output file.

- a statement.

This begins with the statement symbol (default value: @), and is executed immediately by EDT.

- an external statement.

This begins with the user escape character (see @USE). EDT executes the external statement immediately.

Statement and text sequences can be written to any work file.

Creating EDT procedures

EDT procedures should be created in F mode, as this means that the various options available for processing data can also be used to create EDT procedures.

In F mode, a total of 10 work files (0-9) are available for creating EDT procedures, while in L mode 23 work files (0-22) are available. However, the following should be borne in mind with regard to subsequent execution of the procedures:

- In work file 0, EDT procedures can only be created, i.e. they cannot be executed.
- Work files 9 and 10 serve as output files for some statements (e.g. @COMPARE or @FSTAT). An existing EDT procedure can thus be deleted in these work files if necessary.

Calling an EDT procedure

There are two types of EDT procedures:

- @DO procedures (call with @DO)
- @INPUT procedures (call with @INPUT)

Differences between @DO and @INPUT procedures

@DO procedure	@INPUT procedure
contained in a work file	contained in a cataloged file on disk or in a library element
only the whole procedure can be executed	selected parts of the procedure can be executed
several @DO procedures can be nested; further @DO calls are possible; can be called from an @INPUT procedure	can only be nested with one @DO procedure, i.e. no further @INPUT call is possible
branch statements within a @DO procedure	no branch statements
parameters can be passed	parameters cannot be passed

Executing procedures

Procedures are always executed in L mode.

If a procedure is called in F mode, the following occurs:

- EDT automatically branches to L mode,
- the procedure is processed, and then
- EDT automatically switches back to F mode.

5.4 @DO procedures

Statements contained in one of the work files 1-9 (in L mode 1-22) can be called directly from a work file by means of @DO.

The advantages of a @DO procedure are as follows:

- the procedures can be called directly from a work file,
- parameters can be transferred (see @PARAMS) and
- @DO procedures can be nested (further @DO calls within a procedure).

@DO procedures are available in the EDT work files throughout the whole EDT session, i.e. when EDT is terminated, they must be written to a cataloged file on disk (SAM or ISAM file) to prevent them from being deleted.

Example: Calling a procedure as a @DO procedure in F mode

```

1.00 .....
2.00 .....
3.00 .....
4.00 .....
5.00 .....
6.00 .....
7.00 .....

23.00 .....
1.....0001.00:001(0)

```

EDT switches to work file 1, where the procedure is to be written.

```

1.00 @READ 'TESTFILE'.....
2.00 @PAR LOWER=ON.....
3.00 .....
4.00 .....
5.00 .....
6.00 .....
7.00 .....

23.00 .....
0.....0001.00:001(1)

```

The statements are created, complete with the statement symbol, in work file 1. Control then returns to work file 0.

```
1.00 .....  
2.00 .....  
3.00 .....  
4.00 .....  
5.00 .....  
6.00 .....  
7.00 .....  
  
23.00 .....  
do 1.....0001.00:001(0)
```

The procedure in work file 1 is called from work file 0 by means of @DO.

```
1.00 This is a test file.....  
2.00 which has been read into work window 0.....  
3.00 using a procedure.....  
4.00 .....  
5.00 .....  
6.00 .....  
7.00 .....  
  
23.00 .....  
do 1.....0001.00:001(0)
```

Result of the procedure run.

Example: Calling a procedure as a @DO procedure in L mode

```

1.      @PROC 1 ----- (01)
1.      @ @CREATE 1: 'THIS IS AN EXAMPLE'
2.      @ @CREATE 2: 'OF A PROCEDURE IN L MODE'
3.      @ @COPY 1 TO 3 ----- (02)
4.      @ @DELETE 1:1-9
5.      @ @PRINT
6.      @END ----- (03)
1.      @DO 1 ----- (04)
1.0000 AN EXAMPLE
2.0000 OF A PROCEDURE IN L MODE
3.0000 THIS IS AN EXAMPLE
4.

```

(01) EDT switches to work file 1.

(02) 5 EDT statements are written into work file 1 (@DO procedure).

(03) Control returns to work file 0.

(04) The @DO procedure is called. The statements in the work file are executed.

For further examples, see [chapter "EDT statements" on page 161ff](#) (e.g. @DO).

5.5 @INPUT procedures

EDT procedures can be written as @INPUT procedures to SAM or ISAM files on disk.

@INPUT procedures offer the following advantages:

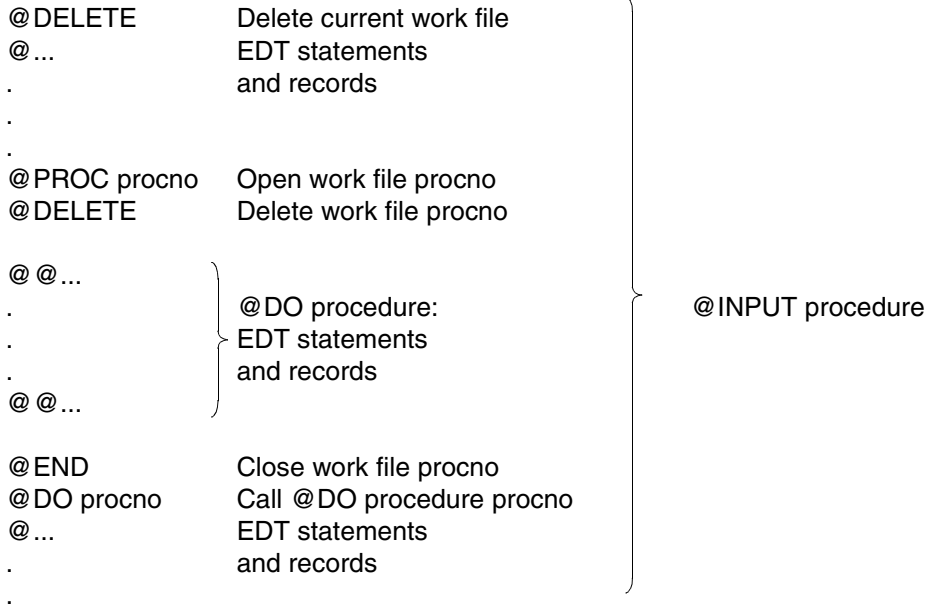
- procedures can be called at any time during any EDT session
- selected parts of the procedure can be executed
- @INPUT procedures can be nested with @DO procedures (@DO call within a @INPUT procedure).

@INPUT procedures cannot be nested within one another. A pure @INPUT procedure (without @DO) must not contain branch statements and parameters.

In order to make full use of the benefits of @INPUT procedures and @DO procedures, @INPUT procedures can be nested with @DO procedures.

Structure of a nested @INPUT procedure

(@DO procedure within an @INPUT procedure)



Example of a nested @INPUT procedure

```

1.00 @NOTE *** READ IN FILE NAME ***.....
2.00 @CREATE #S00 READ : 'FILE NAME:' .....
3.00 @NOTE *** OPEN WORK FILE 1 ***.....
4.00 @PROC 1.....
5.00 @DELETE.....
6.00 @NOTE *** STORE DATE IN THE FORM ddmmy IN #S0***.....
7.00 @ @SET #S01 = DATE ISO.....
8.00 @ @SET #S01 = #S01:1-8.....
9.00 @ @ON #S01 CHANGE ALL '-' TO ' '.....
10.00 @NOTE *** STORE TIME IN THE FORM hhmss IN #S02 ***.....
11.00 @ @SET #S02 = TIME.....
12.00 @NOTE *** CREATE COPY-FILE COMMAND AND STORE IT IN #S03.....
13.00 @ @CREATE #S03 : '/COPY-FILE ',#S00,',',#S00,',',#S01,',',#S02.....
14.00 @NOTE *** EXECUTE COPY-FILE COMMAND AS SYSTEM COMMAND ***.....
15.00 @ @SYSTEM #S03.....
16.00 @NOTE *** CLOSE WORK FILE 1 ***.....
17.00 @END.....
18.00 @DO 1.....
19.00 .....
20.00 .....
21.00 .....
22.00 .....
23.00 .....
write 'backupcopy.input'..... 0001.00:001(1)

```

The procedure is created in work file 1 in F mode and is stored as a SAM file under the name BACKUPCOPY.INPUT.

The @DO procedure comprises all statements with @@ (see below).

```

1.00 @NOTE *** READ IN FILE NAME ***.....
2.00 @CREATE #S00 READ : 'FILE NAME:' .....
3.00 @NOTE *** OPEN WORK FILE 1 ***.....
4.00 @PROC 1.....
5.00 @DELETE.....
6.00 @NOTE *** STORE DATE IN THE FORM ddmmy IN #S0***.....
7.00 @ @SET #S01 = DATE ISO.....
8.00 @ @SET #S01 = #S01:1-8.....
9.00 @ @ON #S01 CHANGE ALL '-' TO ' '.....
10.00 @NOTE *** STORE TIME IN THE FORM hhmmss IN #S02 ***.....
11.00 @ @SET #S02 = TIME.....
12.00 @NOTE *** CREATE COPY-FILE COMMAND AND STORE IT IN #S03.....
13.00 @ @CREATE #S03 : '/COPY-FILE ',#S00,',',#S00,',',#S01,',',#S02.....
14.00 @NOTE *** EXECUTE COPY-FILE COMMAND AS SYSTEM COMMAND ***.....
15.00 @ @SYSTEM #S03.....
16.00 @NOTE *** CLOSE WORK FILE 1 ***.....
17.00 @END.....
18.00 @DO 1.....
19.00 .....
20.00 .....
21.00 .....
22.00 .....
% EDT0160 FILE 'BACKUPCOPY.INPUT' WRITTEN
input 'backupcopy.input';1.....0001.00:001(0)

```

The procedure is called by means of @INPUT. The statements of the SAM file BACKUPCOPY.INPUT are processed. This stores the nested @DO procedure (lines 6.00 to 16.00, EDT statements with more than one statement symbol '@') in work file 1 (see below). Control then switches to work file 1.

```
FILE NAME: testfile
```

When processing the @CREATE ... READ statement, the input request is output. When the file name TESTFILE is entered, a backup copy with the name TESTFILE.ddmmyy.hhmmss is created.

```
1.00 @SET      #S01 = DATE ISO.....  
2.00 @SET      #S01 = #S01:1-8.....  
3.00 @ON       #S01 CHANGE ALL '-' TO ''.....  
4.00 @SET      #S02 = TIME.....  
5.00 @CREATE   #S03 : '/COPY-FILE ',#S00,',',#S00,',',#S01,',',#S02.....  
6.00 @SYSTEM   #S03.....  
7.00 .....  
  
.....0001.00:001(1)
```

The statements of the @INPUT procedure with more than one statement symbol have been stored in work file 1 as a @DO procedure with one statement symbol less.

5.6 Calling an EDT procedure in a BS2000 system procedure

An EDT procedure can also be called from a BS2000 system procedure.

For further information on BS2000 system procedures, see the description of BEGIN-PROCEDURE in Volume 1 of the "Commands" manual [6].

Example of an EDT procedure in a BS2000 system procedure

```

/BEGIN-PROCEDURE -
/ PARAMETER=YES (PROCEDURE-PARAMETERS=(&DATEI),ESCAPE-CHARACTER='&')
/SHOW-FILE-ATTRIBUT &DATEI ----- (01)
/ASSIGN-SYSDTA TO-FILE=*SYSCMD ----- (02)
/MODIFY-JOB-SWITCHES ON=(4,5) ----- (03)
/START-PROGRAM $EDT ----- (04)
@DELETE ----- (05)
@PROC 1 ----- (06)
@DELETE ----- (07)
@ @READ '&DATEI'
@ @PAR LOWER=ON
@ @PAR SCALE=ON ----- (08)
@ @PAR INFORMATION=ON
@ @PAR EDIT FULL=ON
@END ----- (09)
@DO 1 ----- (10)
@DIALOG ----- (11)
@HALT ----- (12)
/ASSIGN-SYSDTA TO-FILE=*PRIMARY ----- (13)
/MODIFY-JOB-SWITCHES OFF=(4,5) ----- (14)
/SET-JOB-STEP
/END-PROCEDURE

```

- (01) The system checks whether the file exists. If it does not, it branches to SET-JOB-STEP.
- (02) The input source is assigned. The system reads in both commands and data via SYSCMD.
- (03) Task switches 4 and 5 are set (see [section "Task switches" on page 69ff](#)).
- (04) EDT is called.
- (05) The contents of work file 0 are deleted.
- (06) EDT switches to work file 1.
- (07) The contents of work file 1 are deleted.
- (08) The EDT statements are stored in work file 1.

- (09) EDT returns to work file 0.
- (10) The @DO procedure in work window 1 is called (reading in a file, differentiating between uppercase and lowercase letters, outputting a column counter, outputting an information line, setting data window and mark column to overwritable).
- (11) EDT switches to F mode dialog. Once dialog mode has been terminated by means of @HALT or @RETURN, the procedure run is resumed at the point at which it was interrupted.
- (12) EDT is terminated.
- (13) The input sources are reset (reset optionally from END-PROCEDURE).
- (14) The task switches are reset.

5.7 Unconditional and conditional branches

Within a @DO procedure, the @GOTO statement causes a branch to another line. The line number is specified in @GOTO. The line must exist and must not be outside the procedure. The system can only branch to a line, i.e. it cannot branch to a mark.

The @IF statement within a @DO procedure causes a branch which is dependent on a condition. If the condition is fulfilled, the system branches to the line specified in the @IF statement. If the condition is not fulfilled, the procedure is resumed with the statement immediately following the @IF statement.

In order to avoid the possible displacement of lines when the procedure is updated, the line numbers before the branch destinations should be redefined by means of @SET, format 6 (abbreviated to @ to make it easier to read); see example.

Within a pure @INPUT procedure, branching is not permitted.

Example of unconditional and conditional branching

```

1.00 @PROC 2.....
2.00 @DELETE.....
3.00 @1.00.....
4.00 @ @CONTINUE *** AS OF HERE LINE NUMBER 1.00 ***.....
5.00 @ @CREATE #S1 READ 'PLEASE ENTER SEARCH STRING: '.....
6.00 @ @ON & FIND #S1 MARK 5.....
7.00 @ @NOTE *** IF HIT FOUND IN @ON, GOTO LINE NUMBER 2.00 ***.....
8.00 @ @IF .TRUE. GOTO 2.....
9.00 @ @CREATE #S2: 'NO HIT FOUND'.....
10.00 @ @PRINT #S2.....
11.00 @ @NOTE *** IF NO HIT FOUND GOTO LINE NUMBER 3.00 ***.....
12.00 @ @GOTO 3.....
13.00 @2.00.....
14.00 @ @CONTINUE *** AS OF HERE LINE NUMBER 2.00 ***.....
15.00 @ @DELETE MARK 5.....
16.00 @ @ON & PRINT #S1.....
17.00 @3.00.....
18.00 @ @CONTINUE *** AS OF HERE LINE NUMBER 3.00 ***.....
19.00 @END.....
20.00 @DO 2.....
21.00 .....

```

Once a search string (line 5.00) has been entered, the system first checks whether the file contains any records containing this search string (line 6.00). If no such records are found, NO HIT FOUND is displayed on the screen (lines 9.00-10.00) and the procedure is terminated. Otherwise, all the records containing the search string are listed on the screen (lines 15.00-16.00).



The statement @1.00 sets the current line number to 1 and also implicitly sets the current increment to 0.01.

5.8 External and internal loops

With external loops, procedures can be completely executed several times. However, if only certain parts of a procedure are to be executed several times, these parts must be formulated in the form of internal loops.

External loops can be replaced by internal loops. In an external loop, only a fixed increment - a freely selectable positive or negative value can be specified. In an internal loop, a variable increment can be specified, e.g. in the form of a line number variable.

Example of an external loop

```

1.00 @PROC 3.....
2.00 @DELETE.....
3.00 @NOTE *** ! IS REPLACED BY THE APPROPRIATE LINE NUMBER .....
4.00 @NOTE LINES 11,12,13,14 AND 15 ARE PROCESSED ***.....
5.00 @ @COLUMN 10 ON ! INSERT !:27-36:.....
6.00 @END.....
7.00 @NOTE *** PROCEDURE IS CALLED WITH ! AS LOOP COUNTER .....
8.00 @NOTE INITIAL VALUE: 11 .....
9.00 @NOTE FINAL VALUE: 15 .....
10.00 @NOTE INCREMENT: 1 .....
11.00 @DO 3, !=11,15.....
12.00 .....

```

The procedure enables copying to be performed column by column within a file. In the above example, the contents of the appropriate line from column 27 through 36 are inserted only in lines 11, 12, 13, 14, and 15, starting in column 10. The exclamation mark (!) is used as a loop counter (see also @DO).

In order to copy not only lines 11, 12, 13, 14 and 15 but also all lines in between (e.g. line 12.34), the procedure must be supplied with an internal loop if no fixed increment is given (e.g. ISAM file, ISAM key as line number).

Example of an internal loop

```

1.00 @PROC 4.....
2.00 @DELETE.....
3.00 @RESET.....
4.00 @NOTE *** SET INITIAL VALUE OF LOOP TO 11 ***.....
5.00 @SET #L10 = 11.....
6.00 @1.00.....
7.00 @NOTE *** ABORT LOOP IF ERROR OCCURS ***.....
8.00 @ @IF ERRORS : @GOTO 2.....
9.00 @NOTE *** ABORT LOOP ON REACHING FINAL VALUE 15 ***.....
10.00 @ @IF #L10 > 15 GOTO 2.....
11.00 @NOTE *** #L10 IS REPLACED BY THE APPROPRIATE LINE NUMBER ***.....
12.00 @ @COLUMN 10 ON #L10 INSERT #L10:27-36:.....
13.00 @NOTE *** INCREMENT LOOP COUNTER TO NEXT AVAILABLE LINE ***.....
14.00 @ @SET #L10 = #L10 + 1L.....
15.00 @ @GOTO 1.....
16.00 @2.00.....
17.00 @ @CONTINUE.....
18.00 @END.....
19.00 @DO 4.....
20.00 .....

```

From line 11 through all lines up to and including line 15 of a file (i.e. including, for example, line 13.314), the contents of the appropriate line from column 27 through 36 are reinserted, starting in column 10. The loop counter here is the line number variable #L10.

If line 11 in the procedure is changed to @ @SET #L10 + 1, only lines 11, 12, 13, 14 and 15 are modified (the effects of this are shown in the previous example).



Line 11.00 must exist in the file which is to be processed, and the last line in this file should be greater than 15.00, otherwise the procedure is terminated with an error message.

5.9 Variable EDT procedures - parameters

When creating procedures in EDT, parameters can be defined by means of @PARAMS. Parameters can be used to transfer different values to a procedure in a @DO call.

The @PARAMS statement must be the first statement in a @DO procedure and must not appear more than once in the procedure. Just as in the BS2000 system, positional and keyword parameters are permitted. All positional parameters must be defined before the keyword parameters.

When the procedure is called, the parameters in the @DO statement are specified as actual parameters. When the procedure is executed, the formal parameters in the procedure are supplied with the values of these actual parameters.

A parameter begins with the character &. This is followed by a letter, which in turn can be followed by 6 further letters or digits.

Example of the use of parameters in an EDT procedure

In the following example, a file is read into work file 0. The records containing the search string are copied into work file 5, processed accordingly and displayed on the screen.

```

1.      @PROC 4
1.      @DELETE
1.      @ @PARAMS &FILE,&SUCH ----- (01)
2.      @ @DELETE
3.      @ @READ '&FILE'
4.      @ @ON & FIND PATTERN '&SRCH' COPY TO (5)
5.      @ @PROC 5
6.      @ @CREATE 0.01: '-' * 68
7.      @ @CREATE 0.02: 'MANUAL LIST FOR:', '&SRCH'
8.      @ @CREATE 0.03: '-' * 68
9.      @ @RENUMBER
10.     @ @PREFIX 4-.$ WITH '      '
11.     @ @SEQUENCE 4-.$:1:0001(1)
12.     @ @CREATE $+1: '-' * 68
13.     @ @PRINT
14.     @ @END
15.     @END
1.      @DO 4 (MANUAL FILE,EDT) -----(02)
1.0000 -----
2.0000 MANUAL LIST FOR: EDT
3.0000 -----
4.0000 0001 EDT V16.3A      STATEMENTS      U1884-J-Z125-5-7600
5.0000 0002 EDT V16.3A      STATEMENTS FORMATS  U1978-J-Z125-4-7600
6.0000 0003 EDT V16.3A      SUBROUTINE INTERFACES U5133-J-Z125-1-7600
7.0000 0004 EDT V16.4A      STATEMENTS      U1884-J-Z125-6-7600

```

```

8.0000 0005 EDT V16.4A      STATEMENTS FORMATS      U1978-J-Z125-5-7600
9.0000 0006 EDT V16.4A      SUBROUTINE INTERFACES   U5133-J-Z125-2-7600
9.0000 0007 EDT V16.4A      EDT-OPERANDEN           U20207-J-Z125-1-7600
7.0000 0008 EDT V16.5A      STATEMENTS               U1884-J-Z125-7-7600
8.0000 0009 EDT V16.5A      STATEMENTS FORMATS      U1978-J-Z125-6-7600
9.0000 0010 EDT V16.5A      SUBROUTINE INTERFACES   U5133-J-Z125-3-7600
10.0000 0011 EDT V16.5A     EDT-OPERANDS             U20207-J-Z125-2-7600
11.0000 -----
1.      @DO 4 (MANUAL FILE,EDT*V16.5*) -----(02)
1.0000 -----
2.0000 MANUAL LIST FOR: EDT*V16.5*
3.0000 -----
4.0000 0001 EDT V16.5A      STATEMENTS               U1884-J-Z125-7-7600
5.0000 0001 EDT V16.5A      STATEMENT FORMATS       U1978-J-Z125-6-7600
6.0000 0002 EDT V16.5A      SUBROUTINE INTERFACES   U5133-J-Z125-3-7600
7.0000 0011 EDT V16.5A     EDT-OPERANDS             U20207-J-Z125-2-7600
8.0000 -----

```

- (01) The symbolic parameters are defined (two positional parameters).
- (02) The procedure is called with the appropriate actual parameters. The formal parameters in the @READ, @ON statements are replaced by the current values in every @DO call.

For further examples see @PARAMS.

BS2000 system procedure parameters

```

/BEGIN-PROCEDURE
/ PARAMETER=YES(PROCEDURE-PARAMETERS=( -
/ &FILE, -
/ &BEGCOL, -
/ &ENDCOL, - ----- (01)
/ &DESTCOL,-
/ &FROMLINE=%, -
/ &TOLINE=$),-
/ ESCAPE-CHARACTER='&')
/SHOW-FILE-ATTRIBUTE &FILE
/ASSIGN-SYSDTA TO-FILE=*SYSCMD
/MODIFY-JOB-SWITCHES ON=(4,5)
/START-PROGRAM $EDT ----- (02)
@READ '&FILE' ----- (03)
@PROC 4
@DELETE
 @NOTE INSERT COLUMN BY COLUMN ----- (04)
@ @COLUMN &DESTCOL ON ! INSERT !:&BEGCOL-&ENDCOL:
@END
@DO 4, !=&FROMLINE,&TOLINE ----- (05)
@WRITE '&FILE.NEW' ----- (06)
@HALT ----- (07)
/ASSIGN-SYSDTA TO-FILE=*PRIMARY
/MODIFY-JOB-SWITCHES OFF=(4,5)
/SET-JOB-STEP
/END-PROCEDURE

```

- (01) The symbolic parameters are defined.
- (02) EDT is called.
- (03) File '&file' is read into work file 0.
- (04) EDT procedure:
- switch to work file 4
 - delete work file 4
 - write the @COLUMN statement with the current parameter values into work file 4
 - switch back to work file 0.
- (05) The EDT procedure is called:
In the file '&file', the columns '&begcol' through '&endcol' are inserted for the specified line range '&fromline' to '&toline', starting at the column '&destcol'.
- (06) The modified file is stored under the name '&file.NEW'.
- (07) EDT is terminated.

6 EDT statements

This chapter describes all statements which may be entered either in F mode in the EDT statement line and/or in L mode. The statement codes used in the mark column are described in [section "F mode" on page 73ff.](#)

6.1 Description of the syntax

General format of a statement:

Operation	Operands	F mode / L mode / @PROC
operation	$\left. \begin{array}{l} \text{operands} \\ \&\text{str-var} \end{array} \right\}$	

- operation The operation is the same as the statement name, e.g. @OPEN, @COPY, @WRITE... This must be at the beginning of the statement. In F mode, the EDT statement symbol (default value: @) may be omitted.
- operands The operation is followed by operands, which are separated from the operation by one or more blanks. The operands must be entered in the specified order. Each operand may be preceded or followed by any number of blanks.
- str-var String variable which contains the operands (indirect specification). For more details on specifying operands indirectly see [section "Indirect specification of operands" on page 45.](#)

The separator (blank) between the operation and the operands, and between the individual operands, must be entered if it is not possible to distinguish between the operation and the operand or between two operands (example: @SYMBOLS='?' is incorrect; @SYMBOL S='?' is correct).

The following metasyntax is used for representing the statements:

Representation	Explanation	Examples
UPPERCASE LETTERS and special characters	Uppercase letters and special characters designate constants; these must be entered exactly as shown.	UPDATE, OVERWRITE
UPPERCASE LETTERS in bold type	Uppercase letters in bold type designate the short form of the constant. You may use the short or long form of the constant or anything in between.	@HALT Enter: @H, @HA, @HAL or @HALT
lowercase letters	Lowercase letters designate variables; these must be replaced by current values n the user input.	@CODE In, SHOW Enter: @CODE 3,SHOW
{ }	Braces enclose alternatives, i.e. one of these options must be specified.	@LOWER { ON } { OFF } Enter: @LOWER ON or @LOWER OFF
	separates alternatives (in braces)	
[]	Square brackets enclose optional entries.	
....	3 dots indicate that the preceding syntactical unit may be repeated several times in succession.	In,... Enter: 1,3,7
<u>Underscore</u>	Value preset by EDT when the editor is called.	

Example

The statements have the following format:

Operation	Operands	F mode / L mode / @PROC
@COPY	rng [(procno)] [TO ln1 [(inc)] [:] [ln2]] [,...]	

Key:

F mode The statement may be specified in F mode.

L mode The statement may be specified in L mode.

@PROC The statement may be specified only in EDT procedures, or it is used primarily in such procedures.

In L mode, the statement must be preceded by the EDT statement character (symbol) so that EDT can distinguish between statements and data in the input. When EDT is called, this character is set to @ by default. If desired, any other character can be defined as the statement character (see the @: statement).

In F mode, the statement character does not need to be specified in the statement line, since this line may contain only statements and it is therefore not necessary to distinguish between statements and data.

6.2 Overview of the EDT operands

The symbol "|" in the definitions is used to separate alternatives.

Definition	Meaning
binary::=0 1	A binary number.
char	Any character which can be entered on a terminal or from a card reader.
chars::=char chars char	Character string.
cl::=int	A column number in the range 1 through 256. Some EDT statements, however, require a cl value of less than 256.
clrng::=cl[-cl]	Column range from the beginning of the record in the direction of the end of the record.
col::=domain col,domain cl col,cl	Several column ranges (domains) or columns, separated from each other by commas, e.g. 1-6, 8-11 or 1-6, 3-10, 3-10.
comment::=chars	Any desired comment text.
dd::=0 1 2 3 4 5 6 7 8 9	A decimal digit.
domain::=cl[-cl]	A column range in the form cl-cl which restricts a specified line range to a certain column range beginning at the first cl value. The second cl value must not be less than the first. If the second cl value is omitted, the column range begins at the first cl and ends at the end of the line. If the second cl is specified and is greater than the line length, the column range likewise extends to the end of the line. If the first cl is greater than the line length, the related line is ignored.
edtsymb::=spec	The statement symbol. The default value is the @ character (this can be changed by means of @:).
entry::=name .str-var	Name of an entry point (ENTRY) or a CSECT statement in a program.
elemname::=chars .str-var	Name of the library element.
elemtyp::=SIMIRICIP IJ D X H L U F * S D I freetyp .str-var	Type of the library element. User-defined type names may also be used, but no check is made on the base type.

Definition	Meaning
file::=str	A file name, which can be specified in printable characters or in binary or hexadecimal notation. It may have up to 54 characters, and the DMS restrictions on file names must be observed. EDT does not check if the file name is valid, which means that use of an invalid file name will result in a DMS error message. Instead of the file name, the complete path name may also be specified for "file". Use of the special file name "/" (file link) is not permitted with the @OPEN statement.
formal::=&id	A formal parameter in the form &id, which must be specified in the @PARAMS statement of a procedure file. id is the name, which may consist of up to 7 letters or numbers. The first character must be a letter. This operand is used in keyword and formal parameters.
fraction::=.ddl fraction dd	The line number portion after the decimal point (i.e. .0001 to .9999).
freetyp::=chars	User-defined type name of a library element. The type name is a string from 2 to 8 characters in length and must not begin with \$ or SYS.
fwkfnr::=dd	The number of a work file. Minimum value is 0, maximum is 9.
fwkfv::=\$dd	The number of a work file variable (\$0 through \$9).
hd::=ddlAIBICIDIEIF	A hexadecimal digit.
hex::=hdlhex hd	A hexadecimal expression.
hpos::=op nlvpos-opl vpos-op (m,...)	Relative vertical position.
hpos-op::=> <lhpos-op n	Vertical positioning statement.
inc::=nlfractionln fraction	Line number or an increment for line numbers. This must lie between 0.0001 and 9999.9999.
int::=nlop nlint-var	An integer, which is specified either explicitly or as an integer variable, e.g.: -5,0,23456,... or #I0,#I1,... #I20.
int-var::=#In	One of the integer variables #I0, #I1, ..., #I20. The maximum permissible value of such a variable is 2^{31} (=2147483648), but the actual limits depend on the application. All integer variables are preset to 0. The integer variables #I0 and #I1 are modified by an @ON statement.

Definition	Meaning																																				
linkname::=chars	Specifies a file or job variable via its link name. No wildcards may be used in "linkname".																																				
line::=In str-In	A line number, which may be specified either as for operand In or as for operand str-In.																																				
In::= In-sym[op int-var]l In-sym[op nL]l	<p>With the operand In, line numbers can be specified symbolically in one of the following formats:</p> <p>Format 1:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>In-sym</th> <th>op</th> <th>int-var</th> </tr> </thead> <tbody> <tr> <td>%</td> <td></td> <td></td> </tr> <tr> <td>*</td> <td></td> <td></td> </tr> <tr> <td>\$</td> <td>+ / -</td> <td>#l0...#l20</td> </tr> <tr> <td>?</td> <td></td> <td></td> </tr> <tr> <td>In-var</td> <td></td> <td></td> </tr> </tbody> </table> <p>Symbols %, *, \$, ? are explained under In-sym. Example: % + #l15. If %=1.0000 and #l15=6, % + #l15 addresses the sixth line after line number 1.0000 (which is not necessarily line 7.0000).</p> <p>Format 2:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>In-sym</th> <th>op</th> <th>nL</th> </tr> </thead> <tbody> <tr> <td>%</td> <td></td> <td></td> </tr> <tr> <td>*</td> <td></td> <td></td> </tr> <tr> <td>\$</td> <td>+ / -</td> <td>1L...nL</td> </tr> <tr> <td>?</td> <td></td> <td></td> </tr> <tr> <td>In-var</td> <td></td> <td></td> </tr> </tbody> </table> <p>nL skips n lines, starting at In-sym. The maximum value for n thus depends on the number of lines in the file. Example: if % = 1.000, then % + 2L addresses the second line after line 1.0000.</p>	In-sym	op	int-var	%			*			\$	+ / -	#l0...#l20	?			In-var			In-sym	op	nL	%			*			\$	+ / -	1L...nL	?			In-var		
In-sym	op	int-var																																			
%																																					
*																																					
\$	+ / -	#l0...#l20																																			
?																																					
In-var																																					
In-sym	op	nL																																			
%																																					
*																																					
\$	+ / -	1L...nL																																			
?																																					
In-var																																					

Definition	Meaning																														
In-sym[op In-sym]	<p>Format 3:</p> <table border="1" data-bbox="579 275 1262 488"> <thead> <tr> <th>In-sym</th> <th>op</th> <th>In-sym</th> </tr> </thead> <tbody> <tr> <td>%</td> <td></td> <td>%</td> </tr> <tr> <td>*</td> <td></td> <td>*</td> </tr> <tr> <td>\$</td> <td>+ / -</td> <td>\$</td> </tr> <tr> <td>?</td> <td></td> <td>?</td> </tr> <tr> <td>In-var</td> <td></td> <td>In-var</td> </tr> </tbody> </table> <p>Example: if % = 1.0000 and * = 3.0000, then % + * addresses line 4.0000.</p>	In-sym	op	In-sym	%		%	*		*	\$	+ / -	\$?		?	In-var		In-var												
In-sym	op	In-sym																													
%		%																													
*		*																													
\$	+ / -	\$																													
?		?																													
In-var		In-var																													
[In-sym op]inc[op In-sym]	<p>Format 4:</p> <table border="1" data-bbox="579 645 1262 858"> <thead> <tr> <th>In-sym</th> <th>op</th> <th>inc</th> <th>op</th> <th>In-sym</th> </tr> </thead> <tbody> <tr> <td>%</td> <td></td> <td>0000.0001</td> <td></td> <td>%</td> </tr> <tr> <td>*</td> <td></td> <td>.</td> <td></td> <td>*</td> </tr> <tr> <td>\$</td> <td>+ / -</td> <td>.</td> <td>+ / -</td> <td>\$</td> </tr> <tr> <td>?</td> <td></td> <td>.</td> <td></td> <td>?</td> </tr> <tr> <td>In-var</td> <td></td> <td>9999.9999</td> <td></td> <td>In-var</td> </tr> </tbody> </table> <p>Example: if * = 50.1 and % = 1.0000, then * + 3.5-% addresses line 52.6000.</p>	In-sym	op	inc	op	In-sym	%		0000.0001		%	*		.		*	\$	+ / -	.	+ / -	\$?		.		?	In-var		9999.9999		In-var
In-sym	op	inc	op	In-sym																											
%		0000.0001		%																											
*		.		*																											
\$	+ / -	.	+ / -	\$																											
?		.		?																											
In-var		9999.9999		In-var																											
[In-sym op]inc[op int-var]	<p>Format 5:</p> <table border="1" data-bbox="579 1014 1262 1228"> <thead> <tr> <th>In-sym</th> <th>op</th> <th>inc</th> <th>op</th> <th>In-sym</th> </tr> </thead> <tbody> <tr> <td>%</td> <td></td> <td>0000.0001</td> <td></td> <td></td> </tr> <tr> <td>*</td> <td></td> <td>.</td> <td></td> <td></td> </tr> <tr> <td>\$</td> <td>+ / -</td> <td>.</td> <td>+ / -</td> <td>#10...#120</td> </tr> <tr> <td>?</td> <td></td> <td>.</td> <td></td> <td></td> </tr> <tr> <td>In-var</td> <td></td> <td>9999.9999</td> <td></td> <td></td> </tr> </tbody> </table> <p>Example: if * = 50.1 and #15 = 1, then * + 3.5 + #15 addresses the line which follows line 53.6000 (which is not necessarily line 53.7000).</p>	In-sym	op	inc	op	In-sym	%		0000.0001			*		.			\$	+ / -	.	+ / -	#10...#120	?		.			In-var		9999.9999		
In-sym	op	inc	op	In-sym																											
%		0000.0001																													
*		.																													
\$	+ / -	.	+ / -	#10...#120																											
?		.																													
In-var		9999.9999																													

Definition	Meaning																														
[In-sym op]inc[op nL]	<p>Format 6:</p> <table border="1" data-bbox="579 275 1262 488"> <thead> <tr> <th>In-sym</th> <th>op</th> <th>inc</th> <th>op</th> <th>nL</th> </tr> </thead> <tbody> <tr> <td>%</td> <td></td> <td>0000.0001</td> <td></td> <td></td> </tr> <tr> <td>*</td> <td></td> <td>.</td> <td></td> <td></td> </tr> <tr> <td>\$</td> <td>+ / -</td> <td>.</td> <td>+ / -</td> <td>1L...nL</td> </tr> <tr> <td>?</td> <td></td> <td>.</td> <td></td> <td></td> </tr> <tr> <td>In-var</td> <td></td> <td>9999.9999</td> <td></td> <td></td> </tr> </tbody> </table> <p>nL skips n lines. The maximum value for n thus depends on the number of lines in the file. Example: if * = 50.1000, then * + 3.5 + 6L addresses the sixth line after line 53.6000.</p> <p>These formats can be divided into two groups:</p> <ol style="list-style-type: none"> Absolute line numbers These are line numbers which are addressed without the use of int-var or nL. These absolute line numbers thus result from addition of the individual line numbers. If, for example, % = 1.0000 and * = 3.0000, line number % + 2.1 + * has the value 6.1000. Relative line numbers Relative line numbers are those which are obtained by skipping certain lines, regardless of the increment between the numbers of the skipped lines. If, for example, % = 1.0000, then % + 2.1 + 5L addresses the fifth line after line 3.1000. <p>n in the expression nL must not have the value 0. However, this value can be stored in an integer variable. A relative line number can be assigned only if there is actually a line with this number; otherwise, an error message is issued.</p>	In-sym	op	inc	op	nL	%		0000.0001			*		.			\$	+ / -	.	+ / -	1L...nL	?		.			In-var		9999.9999		
In-sym	op	inc	op	nL																											
%		0000.0001																													
*		.																													
\$	+ / -	.	+ / -	1L...nL																											
?		.																													
In-var		9999.9999																													

Definition	Meaning										
In-sym::=In-var % * \$?	<p>Line number variable or one of the following symbolic line numbers:</p> <table border="1"> <thead> <tr> <th>Symbol</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>*</td> <td>The lowest line number in the file.</td> </tr> <tr> <td>%</td> <td>The current line number, i.e. the line number last displayed by EDT as an acknowledgment.</td> </tr> <tr> <td>\$</td> <td>The highest line number in the file. If the file is empty or contains only a single line, then % = \$.</td> </tr> <tr> <td>?</td> <td>The line number of the first line in which a preceding @ON statement found a hit. The initial value of ? is 0.0001 and this is changed only by a successful @ON statement. After @ON, ? thus has the same value as #L00.</td> </tr> </tbody> </table> <p>*,%,\$,? all refer to the current work file.</p>	Symbol	Meaning	*	The lowest line number in the file.	%	The current line number, i.e. the line number last displayed by EDT as an acknowledgment.	\$	The highest line number in the file. If the file is empty or contains only a single line, then % = \$.	?	The line number of the first line in which a preceding @ON statement found a hit. The initial value of ? is 0.0001 and this is changed only by a successful @ON statement. After @ON, ? thus has the same value as #L00.
Symbol	Meaning										
*	The lowest line number in the file.										
%	The current line number, i.e. the line number last displayed by EDT as an acknowledgment.										
\$	The highest line number in the file. If the file is empty or contains only a single line, then % = \$.										
?	The line number of the first line in which a preceding @ON statement found a hit. The initial value of ? is 0.0001 and this is changed only by a successful @ON statement. After @ON, ? thus has the same value as #L00.										
In-var::=#Ln	One of the line number variables #L0, #L1, ..., #L20. The minimum value of a line number variable is 0.0001, the maximum 9999.9999. All line number variables are initially set to 0, but this is an invalid value. This means that valid values must be assigned to line number variables before they are used. The line number variable #L0 is modified by an @ON statement.										
m::=ddlint-var	Record marks 1 to 9.										
message::=chars	Any desired text. If EDT was called as a subroutine, this text is returned to the calling program when EDT is terminated by means of @RETURN or @HALT.										
modlib::=pathl.str-var	Module library or program library containing a particular module or load unit.										
n::=ddln dd	One of the digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 or a combination of these digits. The number of digits depends on the statement in which "n" is used. 00005 is thus not necessarily equivalent to 5.										
name::=chars	Character string not exceeding eight characters in length.										

Definition	Meaning
op::=+ -	One of the mathematical functions + or -. These are used with the operands int, str-ln and ln.
param::= '[any string]' any string	With the aid of @DO, parameters can be passed to the procedure file to be executed. Such a parameter consists of any string, which must be enclosed in quotes if it includes a comma or a right parenthesis as part of the parameter. In this case, any single quotes character in the parameter string must be entered twice. The single quotes used to enclose the string must not be redefined by means of the @QUOTE statement.
path::=chars .str-var	Path name of a file or job variable. path may have up to 54 characters, and the DMS restrictions on file names must be observed.
pfile::=str	File name that can be specified in printable, hexadecimal or binary form. The file name may have up to 80 characters (with wildcards), and the DMS restrictions for file names apply. The name of a file or job variable may be partially qualified.
ppath::=chars .str-var	Path name of a file or a job variable. ppath may have up to 80 characters (with wildcards), and the DMS restrictions for file names apply. A file or a job variable may also be partially qualified.
procno::=int	The number of a work file in the range 1 through 22. Exception: 0 is permitted in the statements @COMPARE, @SETF, @COPY, @MOVE, @STATUS and @ON.
r::=spec	The range symbol. The default value is the & character; this can be changed by means of @RANGE.
range::=rng range,rng	One or more line ranges, separated from each other by commas.
range*::=rng* range*,rng*	Analogous to the "range" operand, but no string variables may be used.

Definition	Meaning														
rel::=GT LT GE LE EQ NE > < >= <= = <>	<p>A relational operator for specifying conditions in an @IF statement.</p> <p>Permissible operators are::</p> <table border="1" data-bbox="579 337 1262 640"> <thead> <tr> <th data-bbox="579 337 919 376">Symbol</th> <th data-bbox="919 337 1262 376">Meaning</th> </tr> </thead> <tbody> <tr> <td data-bbox="579 376 919 421">GT or ></td> <td data-bbox="919 376 1262 421">greater than</td> </tr> <tr> <td data-bbox="579 421 919 467">LE or <</td> <td data-bbox="919 421 1262 467">less than</td> </tr> <tr> <td data-bbox="579 467 919 512">GE or >=</td> <td data-bbox="919 467 1262 512">greater than or equal to</td> </tr> <tr> <td data-bbox="579 512 919 557">LE or <=</td> <td data-bbox="919 512 1262 557">less than or equal to</td> </tr> <tr> <td data-bbox="579 557 919 603">EQ or =</td> <td data-bbox="919 557 1262 603">equal to</td> </tr> <tr> <td data-bbox="579 603 919 640">NE or <></td> <td data-bbox="919 603 1262 640">not equal to</td> </tr> </tbody> </table>	Symbol	Meaning	GT or >	greater than	LE or <	less than	GE or >=	greater than or equal to	LE or <=	less than or equal to	EQ or =	equal to	NE or <>	not equal to
Symbol	Meaning														
GT or >	greater than														
LE or <	less than														
GE or >=	greater than or equal to														
LE or <=	less than or equal to														
EQ or =	equal to														
NE or <>	not equal to														
rng::=line[-line]lr	<p>A line range in the form line-line. Specifying line1-line2 (e.g. 1-10) has the same effect as line2-line1 (10-1).</p> <p>If only one line is specified, the line range consists only of this one line.</p> <p>If two line operands are specified and the first of these is a string variable, the other must also be a string variable.</p> <p>The operand r shown above represents a line range which can be declared by means of the @RANGE statement and is preset to a value in the range 0.0001-9999.9999. Both the range symbol and the line range can be modified by means of the @RANGE statement. The desired line range may also extend over string variables. Care must be taken when using the - symbol and the operands ln-sym and str-var: since the minus sign is also used for expressing a line range, ambiguities can result.</p> <p>In order to avoid this problem, the following conventions have been introduced.</p> <ol style="list-style-type: none"> <li data-bbox="579 1228 1262 1323">1. If a range begins with ln-sym and ends with n (in any form), then enter: ln-sym.-ln 														

Definition	Meaning																								
	<p>2. If a range begins with ln (except ln-sym) and ends with a line number expressed via the operands ln-sym and op, it is advisable to enter: ln - 0 ln-sym [op int-var] ln - 0 ln-sym [op nL] ln - 0 ln-sym [op ln-sym] ln - [0 ln sym op] inc [op ln-sym] ln - [0 ln-sym op] inc [op int-var] ln - [0 ln sym op] inc [op nL] Specifying 0 indicates that this is a range and not a difference. 0 may also be replaced by . (period).</p> <p>3. If a range starts with a string variable and ends with str-ln (in any form), enter: str-var[-0 str-var][op int-var] str-var[-0 str-var][op nL]</p>																								
	<p>Examples:</p> <table border="1" data-bbox="579 782 1260 1362"> <thead> <tr> <th data-bbox="579 782 803 824">rng</th> <th data-bbox="803 782 1260 824">Meaning</th> </tr> </thead> <tbody> <tr> <td data-bbox="579 824 803 866">1-10</td> <td data-bbox="803 824 1260 866">Line 1 to 10</td> </tr> <tr> <td data-bbox="579 866 803 908">%.-5</td> <td data-bbox="803 866 1260 908">1st line of the file to line 5</td> </tr> <tr> <td data-bbox="579 908 803 987">%+5L-\$-10L</td> <td data-bbox="803 908 1260 987">6th line of the file to the 10th line before the end of the file</td> </tr> <tr> <td data-bbox="579 987 803 1029">%.-\$</td> <td data-bbox="803 987 1260 1029">From *+2.1-? to the 6th line of the file</td> </tr> <tr> <td data-bbox="579 1029 803 1071">*+2.1-?-%+5L</td> <td data-bbox="803 1029 1260 1071">From the 1st line to line #L2 + 5L</td> </tr> <tr> <td data-bbox="579 1071 803 1113">#L1-#L2</td> <td data-bbox="803 1071 1260 1113">From #L1 to #L2</td> </tr> <tr> <td data-bbox="579 1113 803 1155">12.011</td> <td data-bbox="803 1113 1260 1155">Only line 12.011</td> </tr> <tr> <td data-bbox="579 1155 803 1234">#L9</td> <td data-bbox="803 1155 1260 1234">Only the line whose number is stored in #L9</td> </tr> <tr> <td data-bbox="579 1234 803 1276">#S3</td> <td data-bbox="803 1234 1260 1276">Only the string variable #S3</td> </tr> <tr> <td data-bbox="579 1276 803 1318">#S4-#S7</td> <td data-bbox="803 1276 1260 1318">#S4, #S5, #S6, #S7</td> </tr> <tr> <td data-bbox="579 1318 803 1362">#S2+1L-#S6-#I3</td> <td data-bbox="803 1318 1260 1362">#S3,..., #S6-#I3</td> </tr> </tbody> </table>	rng	Meaning	1-10	Line 1 to 10	%.-5	1st line of the file to line 5	%+5L-\$-10L	6th line of the file to the 10th line before the end of the file	%.-\$	From *+2.1-? to the 6th line of the file	*+2.1-?-%+5L	From the 1st line to line #L2 + 5L	#L1-#L2	From #L1 to #L2	12.011	Only line 12.011	#L9	Only the line whose number is stored in #L9	#S3	Only the string variable #S3	#S4-#S7	#S4, #S5, #S6, #S7	#S2+1L-#S6-#I3	#S3,..., #S6-#I3
rng	Meaning																								
1-10	Line 1 to 10																								
%.-5	1st line of the file to line 5																								
%+5L-\$-10L	6th line of the file to the 10th line before the end of the file																								
%.-\$	From *+2.1-? to the 6th line of the file																								
*+2.1-?-%+5L	From the 1st line to line #L2 + 5L																								
#L1-#L2	From #L1 to #L2																								
12.011	Only line 12.011																								
#L9	Only the line whose number is stored in #L9																								
#S3	Only the string variable #S3																								
#S4-#S7	#S4, #S5, #S6, #S7																								
#S2+1L-#S6-#I3	#S3,..., #S6-#I3																								
rng*::=ln[-ln]lr	<p>A special line range expressed via r or ln-ln. In contrast to "rng", string variables must not be used here. For this reason, the operand "ln" applies instead of "line" (see above).</p>																								
search::=strnglline[:col:]	Search string.																								

Definition	Meaning										
spec	Special character.										
str::= 'any string'[*int] B'binary'[*int] X'hex'[*int]	<p>3 cases are possible:</p> <ol style="list-style-type: none"> 1. 'any string' [*int] 2. B'binary number' [*int] 3. X'hexadecimal number' [*int] <p>If B or X is used, then this letter must immediately precede the binary or hexadecimal number, which is enclosed in single quotes.</p> <p>If, in case 1, the string includes a single quote, then this must be entered twice. The valid character for a single quote can be changed by means of @QUOTE.</p> <p>The optional operand *int permits repetition of a string. 'ab'*3, for example, is equivalent to 'ababab'. Since the maximum length of a string is 256 bytes, "int" must not exceed this value. If "int" is 0, or if the length of the string is 0, then the resulting string also has a length of 0.</p> <p>Examples:</p> <table border="1" data-bbox="572 819 1260 1031"> <thead> <tr> <th data-bbox="572 819 807 861">str</th> <th data-bbox="807 819 1260 861">Resulting string</th> </tr> </thead> <tbody> <tr> <td data-bbox="572 861 807 903">'A"BC"D'</td> <td data-bbox="807 861 1260 903">A'BC'D</td> </tr> <tr> <td data-bbox="572 903 807 945">'ABC' *5</td> <td data-bbox="807 903 1260 945">ABCABCABCABCABC</td> </tr> <tr> <td data-bbox="572 945 807 987">X'C1F2' *4</td> <td data-bbox="807 945 1260 987">A2A2A2A2</td> </tr> <tr> <td data-bbox="572 987 807 1031">B'11110000' *3</td> <td data-bbox="807 987 1260 1031">000</td> </tr> </tbody> </table> <p>Note: if an odd number of digits is entered for a hexadecimal number, then EDT inserts leading zeros as necessary. X'F', for example is equivalent to X'0F', or X'A' *4 equivalent to X'0A' *4.</p> <p>The same applies to a binary number if the number of binary digits specified is not a multiple of 8. Leading zeros are inserted up to the next multiple of 8 binary digits. B'1', for example, is equivalent to B'00000001', or B'1111' *2 is equivalent to B'00001111' *2.</p>	str	Resulting string	'A"BC"D'	A'BC'D	'ABC' *5	ABCABCABCABCABC	X'C1F2' *4	A2A2A2A2	B'11110000' *3	000
str	Resulting string										
'A"BC"D'	A'BC'D										
'ABC' *5	ABCABCABCABCABC										
X'C1F2' *4	A2A2A2A2										
B'11110000' *3	000										

Definition	Meaning
string:=strlline[:col:]	<p>Either operand "str" or "line [:col:]". If a string variable or a line number is specified, EDT uses the contents of this string variable or of this line as "string". If the specified line does not exist, an error message is issued and the statement is rejected.</p> <p>If only part of a line is to be used as "string", then the desired columns may be specified. If column values greater than the line length are specified, blanks are inserted into "string".</p> <p>If, for example, line 6 contains the string AB3CD6EF9</p> <p>and if "string" is specified as 6:1-3,9,8,9,8-9,5-7,30,1,30,1,;</p> <p>the resulting string is AB39F9F9D6E A A</p> <p>If the column specification is used in one of the statements @INPUT, @GET, @SAVE, @READ or @WRITE, then the above applies to the lines to be read from or written to the disk file. If the line number specified for "string" is the number of a line which is to be modified using the EDT statement in which "string" is used as an operand, then "string" refers to the original contents of this line. If, for example, line 1 contains ABC</p> <p>and the EDT statement is: @CREATE1:1,'D'*3,1</p> <p>the line 1 contains, after execute of this statement ABCDDDABC</p>

Definition	Meaning												
str-ln::= str-var[op int-var] str-var[op nL]	<p>A special form for expressing a string variable. There are two possibilities:</p> <p>Format 1:</p> <table border="1" data-bbox="579 371 1260 455"> <tr> <td style="text-align: center;">str-var</td> <td style="text-align: center;">op</td> <td style="text-align: center;">int-var</td> </tr> <tr> <td style="text-align: center;">#Sn1</td> <td style="text-align: center;">+ / -</td> <td style="text-align: center;">#ln2</td> </tr> </table> <p>Example: if #l10=5, then #S0+#l10=#S5 or #S13+#l10=#S18.</p> <p>Format 2:</p> <table border="1" data-bbox="579 640 1260 724"> <tr> <td style="text-align: center;">str-var</td> <td style="text-align: center;">op</td> <td style="text-align: center;">nL</td> </tr> <tr> <td style="text-align: center;">#Sn</td> <td style="text-align: center;">+ / -</td> <td style="text-align: center;">1L...nL</td> </tr> </table> <p>Example: #S15-5L=#S10 or #S3+8L=#S11.</p>	str-var	op	int-var	#Sn1	+ / -	#ln2	str-var	op	nL	#Sn	+ / -	1L...nL
str-var	op	int-var											
#Sn1	+ / -	#ln2											
str-var	op	nL											
#Sn	+ / -	1L...nL											

Definition	Meaning
<p>strng::= 'any string'[*int] B'binary'[*int] X'hex'[*int]</p>	<p>Same as operand "str", but double quotes (") may be used instead of single quotes (').</p> <p>This operand may be used only in the first operand ("search") in the @ON statement. If the string includes a single or double quote character, it must be entered twice. Again as for operand str, a repetition factor (*int) may be specified, but the resulting search string must not be longer than 256 characters.</p> <p>A search string is delimited on the left and the right by a single or double quote, both of which can be redefined by means of @QUOTE. If the double quote is used to delimit the string on the left or right, then this indicates that there must be a text delimiter on the left or right, respectively, of the search string. The set of text delimiters is initially set by EDT to the characters +.!*();-/,?:'=" and the blank (X'40'). This character set can be changed by means of the @DELIMIT statement. By definition, there is always a text delimiter before the first character and after the last character. For the following example, it is assumed that @DELIMIT has been used to define the comma and the blank as delimiters. The @ON statement is now to be used to search for a line with the following contents:</p> <pre>FLIPPER+*FLIPPER ,FLIPPER +FLIPPER* 1 10 20 30</pre> <p>The numbers under the line show the column numbers. The search string in each case is the word FLIPPER; however, note how the single and double quotes are used to enclose this word:</p> <pre>'FLIPPER' Hits in columns 1, 10, 20 and 30 'FLIPPER" Hits in columns 10 and 20 "FLIPPER' Hits in columns 1 and 20 "FLIPPER" Hit in columns 20</pre>
<p>structured-name::=chars</p>	<p>String with a maximum length of 30 characters. Permitted characters: A...Z 0...9 \$, #, @ hyphen First character: A...Z oder \$, #, @</p>

Definition	Meaning
str-var::=#Sn	One of the 21 string variables #S0, #S1 ..., #S20. These must be regarded as special additional lines in a file which may be used to buffer any desired strings or the contents of any other lines. With only a few exceptions, which are mentioned in the syntax of the statements, string variables can be treated just like normal lines. The contents of a string variable have a minimum length of 1 and must not be longer than 256 characters. If a string variable is deleted, it contains one blank, which is also the default content of all string variables.
tab::=char	The tab character. This must be a character other than the statement symbol "edtsymb".
text::=chars	A text, with a maximum length of 256 characters, which is entered from the terminal. If the string which is entered starts with the EDT statement symbol (@) and the next character is neither this symbol nor a blank, the input is regarded as an EDT statement. If this is not desired, and the statement symbol is to be used as a text character in column 1, then this symbol must be entered twice (@@).
usersymb::=char	The user escape symbol for external statement routines. This is defined by means of @USE.

Definition	Meaning
ver.:=*int	<p>The version number of a cataloged file. Either an asterisk (*) or (int) may be specified, where "int" is a non-negative number. Such a version number may be used in the EDT statements @OPEN, @ELIM, @GET, @INPUT, @READ, @SAVE, @UNSAVE and @WRITE, for example</p> <pre>@GET 'P.BEISPIEL'(2)</pre> <p>If the asterisk (*) is specified for the version number in @GET, @INPUT or @READ, the current version number is displayed on the terminal after the statement is executed.</p> <p>If however, a number - such as (2) - is entered, the current version number is displayed only if the specified version number is invalid. The specified file is still read in this case.</p> <p>Exception: execution of an @INPUT file is aborted as for a DMS error, even if the statement with the invalid version number is an input statement.</p> <p>If version number (*) is specified in an output statement, the new version number is displayed on the terminal and the file is then output.</p> <p>If a number is specified instead of the asterisk, the output is executed only if the correct version number was specified. Otherwise, the correct version number is simply displayed on the terminal.</p> <p>A new version number is generated when a file is first created and each time the file is updated. This new version number is formed by incrementing the old number by 1 (exceptions are possible for the @OPEN statement).</p>

Definition	Meaning
	<p>A new file has the version number 1. The highest possible version number is 256 and the next new version will have the version number 0. A file also has version number 0 before it is cataloged. The version number is not changed if an output statement does not access the file. If, for example, a range which does not exist in this file is specified in @ELIM, the version number of this file remains unchanged. In the other cases, however, the version number is changed even if a file is completely overwritten or if a SAM file is converted to an ISAM file or vice versa.</p> <p>The version number offers increased protection against destruction of a file. If a version number is specified when the file is read in, the actual version number is output and the user can see if this is an old version of the file.</p>
vers::=chars!*STD	Version identifier of a library element.
vpos::=op n vpos-op vpos-op (m,...)	Relative horizontal position.
vpos-op::=+ - ++ --	Horizontal positioning statement.
xpath::=chars .str-var	Character string not exceeding 256 characters in length. Specifies the name of a POSIX file (perhaps also its path). Blanks and non-printable characters are permitted in the name only if specified in str-var.

6.3 Overview of the EDT statements

EDT management

	Define a new statement symbol for the statements.	F mode L mode
@:	edtsymb	
	Switch on/off automatic saving of unsaved virtual files.	F mode L mode
@AUTOSAVE	{ [ID=name] [,] TIME=n [ON] } OFF	
	Switch block mode for block input on or off.	F mode L mode
@BLOCK @BK	{ [ON [,] [AUTOFORM]] } OFF	
	Display lines changed by statements. Check the length of newly entered lines.	L mode
@CHECK	{ [ON] } [,] [cl] OFF	
	Switch on the code table for line number In and display it on the screen (format 1).	F mode L mode
@CODE	In, SHOW	
	Switch on the code table for line number In format 2).	F mode L mode
@CODE	In	
	Switch on, display or switch off the current code table (format 3).	F mode L mode
@CODE	[ON] SHOW OFF	
	The specified coded character set is selected.	F mode L mode
@CODENAME	[name]	

@DELIMIT	Modify the text delimiter set (see @ON).	F mode L mode
	= $\left. \begin{array}{l} \mathbf{R} \\ \text{str1} \\ \mathbf{+ - str2} \end{array} \right\}$	
@INPUT	Change the input mode in L mode (format 3).	L mode
	$\left\{ \begin{array}{l} \mathbf{[CHAR]} \\ \mathbf{HEXIX [ISO]} \\ \mathbf{BINARY} \end{array} \right\}$	
@LIMITS	Display the lowest and highest assigned line numbers in the current work file.	F mode L mode
@LOWER	Specify whether a distinction is to be made between uppercase and lowercase letters.	F mode L mode
	[ON] OFF	
@P-KEYS	Load or display the default values for the programmable keys.	F mode L mode
	[SHOW]	

<p>@PAR</p>	<p>Preset values for input and output and for file processing.</p>	<p>F mode L mode</p>
	<p>[fwkfv GLOBAL]</p> <p>[,] [EDIT[-] LONG] [=ON] =OFF] [,] [HEX] [=ON] =OFF] [,] [LOWER] [=ON] =OFF] [,] [EDIT[-]FULL] [=ON] =OFF] [,] [PROTECTION] [=ON] =OFF] [,] [SCALE] [=ON] =OFF] [,] [INFORMATION] [=ON] =OFF]</p> <p>[,] [INDEX] [=ON] =OFF] [,] [OPTIMIZE] [=ON] =OFF] [,] [RENUMBER] [=ON] =OFF]</p> <p>[,] [SPLIT = n fwkfv =OFF] [,] [SEPARATOR = 'char' =OFF] [,] [CODE= EBCDIC =ISO] [,] [[ELEMENT] [-] TYPE = elemtyp =*STD] [,] [INCREMENT = inc] [,] [LIBRARY = path] [,] [LIMIT = cl] [,] [STRUCTURE = 'char'] [,] [SDF-PROGRAM =structured-name =*NONE] [,] [SDF-NAME-TYPE = INTERNAL = EXTERNAL]</p>	
<p>@QUOTE @QE</p>	<p>Replace the single and double quotes used as delimiters for file names and character strings in statements by the specified characters.</p> <p>{ spec, char } { spec ,char }</p>	<p>F mode L mode</p>
<p>@RANGE</p>	<p>Change the range symbol (default: &) and the domain default: 0.0001 to 9999.9999).</p> <p>[=r =rng [:domain]]</p>	<p>F mode L mode</p>
<p>@SEA[RCH]-OPTION</p>	<p>Preset whether or not a distinction is to be made between uppercase and lowercase letters when searching for strings with @ON.</p> <p>CASELESS-SEA[RCH] {=ON =OFF}</p>	<p>F mode L mode</p>

@SETSW	Set and reset the task and user switches.	F mode L mode
	$\left. \begin{array}{l} \{\text{ON}=\} \\ \{\text{OFF}=\} \end{array} \right\} [\text{U}] \text{int1} [-\text{int2}] [, \dots]$	
@SYMBOLS	Define the current filler character and the wildcard symbols asterisk (*) and slash (/).	F mode L mode
	$\begin{array}{l} [,] [\text{ASTERISK} [= '*' \mid =\text{'spec1'}] \\ [,] [\text{SLASH} [= '/' \mid =\text{'spec2'}] \\ [,] [\text{FILLER} [= \text{X}'00' \mid =\text{X}'hex' \mid =\text{'char'}] \end{array}$	
@SYNTAX	Set the syntax check and execution mode.	F mode L mode
	$[\text{SECURITY} \left\{ \begin{array}{l} [= \text{HIGH}] \\ = \text{LOW} \end{array} \right\}] [[,] \text{TESTMODE} \left\{ \begin{array}{l} [= \text{ON}] \\ = \text{OFF} \end{array} \right\}]$	
@TABS	Define and display a tab character and the related columns.	F mode L mode
	$\left. \begin{array}{l} \left\{ \begin{array}{l} :: [\text{tab} [[:] \text{cl1} [, \text{cl2}, \dots]] [\left\{ \begin{array}{l} \text{CHECK} \\ \text{FORWARD} \end{array} \right\} [\text{cl}]]] \\ \text{RANGE} [= \text{range}] \\ [\text{cl1} [, \text{cl2}, \dots]] \{ \text{ON} \mid \underline{\text{OFF}} \} \\ [::] \text{VALUES} \end{array} \right\}$	
@VDT	Change the number of lines displayed on the screen (for the 9763 Data Display Terminal, the number of screen columns can be changed as well).	F mode L mode
	$[\text{int}] [,] [\text{E1} \mid \text{F2}]$	
@VTCSET	Activate or deactivate the interpretation of line mode control characters during screen output.	F mode L mode
	$[\text{ON}] \mid \text{OFF}$	
EDIT LONG	Switch the display of entire records on the screen on or off.	F mode
	$[\text{ON}] \mid \underline{\text{OFF}}$	
HEX	Switch hexadecimal mode on or off.	F mode
	$[\text{ON}] \mid \underline{\text{OFF}}$	

INDEX	Switch line number display on or off.	F mode
	[ON] OFF	
SCALE	Switch the column counter scale on or off.	F mode
	[ON] OFF	
SPLIT	Split the screen into two work windows.	F mode
	n(fwkfno) 0 OFF	

File processing

@CLOSE	Close the file previously opened by means of @OPEN.	F mode L mode
	[NOWRITE]	
@ELIM	Delete an ISAM file. The catalog entry is retained.	F mode L mode
	['file'] [(ver)] range* [BOTH]	
@FILE	Define a file name for @READ, @WRITE, @GET, @SAVE and @OPEN.	F mode L mode
	[string [(ver)]] [LOCAL]	
@GET	Read an ISAM file into the current work file.	F mode L mode
	['file'] [(ver)] [range*] [:col:] [NORESEQ]	
@OPEN	Physically open an ISAM file in work file 0.	F mode L mode
	['file1'] [(ver)] [[KEY] [AS 'file2' [OVERWRITE]]]]	
@READ	Read a SAM file into the current work file.	F mode L mode
	['file'] [(ver)] [range*] [:col:] [{ RECORDS } { KEY }] [STRIP]	
@SAVE	Store the current work file as an ISAM file.	F mode L mode
	['file'] [(ver)] [range*] [:col:] [{ UPDATE [RENUMBER [ln [(inc)]]] [OVERWRITE] }]]	
@UNSAVE	Delete a file and its catalog entry.	F mode L mode
	'file' [(ver)]	
@WRITE	Store the current work file as a SAM file (format 1).	F mode L mode
	['file'] [(ver)] [range*] [:col:] [KEY] [{ UPDATE { OVERWRITE } }]]	

Processing of POSIX files

@CLOSE	Close a POSIX file opened earlier with @XOPEN.	F mode L mode
	[NOWRITE]	
@XCOPY	Read in a POSIX file.	F mode L mode
	FILE=xpath [,CODE=EBCDIC ISO]	
@XOPEN	Open and read in a POSIX file.	F mode L mode
	FILE=xpath [,CODE=EBCDIC ISO] [,MODE=ANY UPDATE NEW REPLACE]	
@XWRITE	Write the contents of the current work file into a POSIX file.	F mode L mode
	FILE=xpath [,CODE=EBCDIC ISO] [,MODE=ANY UPDATE NEW REPLACE]	

Program library and file processing

@CLOSE	Store and close a program library element previously opened by means of @OPEN (format 2).	F mode L mode
	[NOWRITE]	
@COPY	Copy a program library element or a file into the current work file (format 2).	F mode L mode
	$\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)][,elemtyp])} \\ \text{ELEMENT=elemname [(vers)][,elemtyp]} \\ \text{FILE=path2} \end{array} \right\}$ $[\left. \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{In}]$	
@DELETE	Delete the specified program library element or file (format 2).	F mode L mode
	$\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)][,elemtyp])} \\ \text{FILE=path2} \end{array} \right\}$	
@OPEN	Open a program library element or a file (SAM/ISAM) in the current work file (format 2).	F mode L mode
	$\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)][,elemtyp])} \\ \text{ELEMENT=elemname [(vers)][,elemtyp]} \\ \text{FILE=path2 [,TYPE=ISAM SAM CATALOG]} \end{array} \right\}$ [,MODE=ANY UPDATE NEW REPLACE]	
@SHOW	Display the directory of a program library or a user catalog or output it to a work file (format 1).	F mode L mode
	$\left. \begin{array}{l} \text{[LIBRARY=path1 [,TYPE=]elemtyp]} \\ \text{TYPE=elemtyp} \\ \text{FILES[=ppath]} \end{array} \right\}$ [[TO] In [(inc)]] $\left. \begin{array}{l} \text{[SHORT]} \\ \text{LONG [ISO4]} \end{array} \right\}$	

@WRITE	Write the current work file to the specified program library element or to a SAM/ISAM file (format 2).	F mode L mode
	$\left[\left\{ \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)],elemtyp)} \\ \text{ELEMENT=elemname [(vers)],elemtyp} \\ \text{FILE=path2 [,TYPE=ISAM SAM]} \end{array} \right\} \right]$ [,MODE=ANY UPDATE NEW REPLACE]	

Switching or positioning the work file

+ - ++ --	Scroll the work file forwards or backwards.	F mode
	[n]	
+ ++ - --	Scroll in the specified direction to the next record mark.	F mode
	[([m,...])]	
> < <<	Scroll to the left or the right.	F mode
	[n]	
@END	Switch to the previous work file.	F mode L mode @PROC
	[comment]	
@ON	Mark records which contain the search string, and (in F mode) position the data window to the first hit record (format 4).	F mode L mode
	range* [:domain] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int] MARK [m]	

@PROC	Switch to another work file.	L mode @PROC
	procno [comment]	
@SETF #	Position a work file to the specified line and column and, optionally, switch to another work file. # is only permissible in F mode.	F mode L mode
	$\left[\left\{ \begin{array}{l} \text{fwkfv} \\ \mathbf{GLOBAL} \\ \text{(fwkfnr)} \end{array} \right\} \right] \left[\left\{ \begin{array}{l} \text{ln} \\ \text{vpos} \end{array} \right\} \right] \left[\left\{ \begin{array}{l} \text{:cl:} \\ \text{hpos} \end{array} \right\} \right]$	
fwkfnr fwkfv	Switch to another work file.	F mode

Line number handling

@	Change the current line number and increment, and shift the stack entries.	L mode
	[ln [(inc)] [:text]]	
@+	Increment the current line number by the current increment value.	L mode
	[:text]	
@-	Decrement the current line number by the current increment value.	L mode
	[:text]	
@RENUMBER	Renumber the lines of the current work file.	F mode L mode
	[ln [(inc)]]	
@SEQUENCE	Place a sequence number in each line of the specified range, starting at the specified column (format 1).	F mode L mode
	[rng] [:cl] [:[n1] (n2)]]]	
@SEQUENCE	Place the line number of each line in each line in the specified range, starting at the specified column (format 2).	F mode L mode
	[rng*] : [cl] : LINE	

	Check that the sequence numbers in the specified range are in ascending order (format 3).	F mode L mode
@SEQUENCE	[rng] : [cl] : CHECK [int]	
	Define a new current line number and a new increment value. Text may be entered at the same time (format 6).	F mode L mode
@SET	In [(inc)] [:text]	

Creating, inserting and modifying texts

@COLUMN	Insert or overwrite text, starting at the specified column. Blanks at the end of the record are deleted.	F mode L mode
	cl ON range { [CHANGE] INSERT } [:] string	
@CREATE	Create the specified line (format 1).	F mode L mode
	line [:] [string[,...]]	
@CONVERT	Convert the specified line or line range to uppercase or lowercase letters.	F mode L mode
	[range] [TO=] { UPPER LOWER }	
@ON	If the search string is found in the specified range, replace it with the specified string (format 7).	F mode L mode
	range [:domain] CHANGE [ALL] [F] [R] [PATTERN] search [,int] [TO] string [V]	
@ON	If the search string is found in the specified range, either replace the text before or after the hit with the specified string or insert the specified string before or after the hit (format 8).	F mode L mode
	range [:domain] FIND [ALL] [F] [R] [PATTERN] search [,int] { CHANGE } { PREFIX INSERT } { SUFFIX } string	
@PREFIX	Insert the specified string at the beginning of each line in the specified range.	F mode L mode
	range WITH string	
@SDFTEST	Initiate SDF check on syntax of data lines.	F mode L mode
	[range*] [PROGRAM [=structured-name [{ INTERNAL EXTERNAL }]]]	
@SEPARATE	Insert a line break.	F mode L mode
	[range*] [AT {'char' X'hex' cl}]	

@SORT	Sort the specified contiguous line ranges.	F mode L mode
	[rng*] $\left\{ \begin{array}{l} [:\text{domain}] \\ :R (\text{clrng}) \end{array} \right\} \left\{ \begin{array}{l} [A] \\ D \end{array} \right\}$	
@SUFFIX	Append the specified string to each line in the specified range.	F mode L mode
	range WITH string	
@UPDATE	Update or delete existing records, either partially or completely (format 1).	L mode
	In [:\text{domain}] ; text	
@UPDATE	Output a file section in edited form (format 2).	L mode
	[In] [:\text{domain}]	
@UPDATE	Define a standard column range for updating records with @UPDATE (format 3).	L mode
	COLUMN [domain]	

Copying and moving lines

@COPY	Copy the specified range from any work file to the current work file (format 1).	F mode L mode
	rng [(procno)] [TO In1 [(inc)] [:] [In2]] [,...]	
@MOVE	Move the specified range from any work file to the current work file; the range is deleted from the source work file.	F mode L mode
	rng [(procno)] [TO In1 [(inc)] [:] [In2]] [,...]	
@ON	Copy the records marked with "m" into work file "procno". If KEEP is specified, the line numbers are retained (format 5).	F mode L mode
	range* [:\text{domain}] FIND [ALL] [F] [NOT] MARK m [COPY [TO]] (procno) [KEEP] [OLD]	
@ON	Copy records containing the search string into work file "procno". If KEEP is specified, the line numbers are retained (format 6).	F mode L mode
	range* [:\text{domain}] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int] [COPY [TO]] (procno) [KEEP] [OLD]	

Deleting work files, lines, texts and record marks

@DELETE	Delete the specified range(s) (format 1).	F mode L mode
	[rng [:domain]] [,...]	
@DELETE	Delete the specified record mark(s) (format 3).	F mode L mode
	MARK [m, [...]	
@DROP	Delete and release the specified work files or all work files.	F mode L mode
	{ procno [,...] }	
@ON	Delete the search string from any lines within the range in which it is found (format 9).	F mode L mode
	range [:domain] DELETE [ALL] [F] [R] [PATTERN] search [,int]	
@ON	In any lines in the specified range in which the search string is found, delete the text before or after this string (format 10).	F mode L mode
	range [:domain] FIND [ALL] [F] [R] [PATTERN] search [,int] DELETE { PREFIX SUFFIX }	
@ON	Delete any lines in the specified range in which the specified search string is found (format 11).	F mode L mode
	range [:domain] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int] DELETE	

Comparing work files

@COMPARE	Compare the specified ranges in two work files; the result of the comparison is a list of line numbers (format 1).	F mode L mode
	[procno1] :rng*1 WITH [procno2] :rng*2 [,[int1] [(int2)] [LIST [In [(inc)]]]]	
@COMPARE	Compare two entire work files; the result of the comparison is a list of line numbers and the contents of the lines (format 2).	F mode L mode
	[[procno1] [WITH]] procno2 [LIST [procno3]] [,procno4]	

Changing the operating mode

@DIALOG	Switch to F mode dialog. Return to the interrupted processing sequence by means of @HALT or @RETURN (e.g. in system procedures).	F mode L mode
@EDIT	Switch from L mode to F mode dialog and vice versa; parameters can be set for editing in L mode.	F mode L mode
	{ FULL SCREEN [ONLY] [PRINT] [SEQUENTIAL] [c] }	

Output of lines and information

[n]#	Show the last n preceding statements in the statement line.	F mode
@FSTAT	Display the requested files on the screen or output them to a work file.	F mode L mode
	{ [pfile'] } [[TO] In [(inc)]] [{ SHORT LONG [ISO4] }]	
@LIST	Output the requested range to SYSLST.	F mode L mode
	[rng [:domain] [X] [N] [C [int]IP int] [I] [S]] [,...]	

@LOG	Control logging in batch mode.	F mode L mode
	[{ ALL COMMANDS }] [{ SYSLST SYSLST nn LIST-VAR=chars }]	
@ON	Display each line in the specified range in which the specified search string occurs (format 1).	F mode L mode
	range [:domain] PRINT [ALL] [F] [R] [NOT] [PATTERN] search [,int] [S] [N] [E]	
@ON	Display the column number at which the specified search string was found (format 2).	F mode L mode
	range [:domain] COLUMN [ALL] [F] [R] [PATTERN] [search [,int]]	
@ON	If the search string is found, place the number of the hit line in #L00 and the numbers of the start and end columns in #I00 and #I01 (format 3).	L mode @PROC
	range [:domain] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int]	
@PAGE	Execute a form feed on SYSLST.	F mode L mode
@PRINT	In F mode, display the contents of string variables; in L mode, the contents of line ranges can also be displayed.	F mode L mode
	[rng [:domain] [X] [N] [S] [V E]] [,...]	
@PROC	Display the current work file.	L mode
@SHIH	Display the statement buffer.	F mode
@SHOW	Display a list of the coded character set names available in the system or output it to a work file (format 1).	F mode L mode
	CCS [[TO] ln [(inc)]]	

<p>@STATUS</p>	<p>Display the current EDT settings and the contents of line numbers and integer variables.</p>	<p>F mode L mode</p>
	<p>[=ALL] </p> <p>[=</p> <p> TIME </p> <p> BUFFER </p> <p> SIZE </p> <p> SYMBOLS </p> <p> DELIM </p> <p> VDT </p> <p> MODES </p> <p> FILE </p> <p> PAR[(procno)] </p> <p> LINEV </p> <p> INTV </p> <p> ln-var </p> <p> int-var </p> <p> SDF </p> <p> CCS </p> <p> LOG]</p> <p> SEARCH-OPTION]</p> <p>[,...]</p> <p>[TO ln [(inc)]]</p>	
<p>@TMODE</p>	<p>Display process information.</p>	<p>F mode L mode</p>

Interrupting or terminating EDT

@END	Terminate EDT.	F mode L mode
	[comment]	
@EXEC	Terminate EDT and load and start the specified program.	F mode L mode
	string	
@HALT	Terminate EDT; if EDT was called as a subroutine, a message can be specified.	F mode L mode
	[ABNORMAL] [message]	
@LOAD	Terminate EDT and load the specified program.	F mode L mode
	string	
@RETURN	Terminate EDT; if EDT was called as a subroutine, a message can be specified.	F mode L mode
	[message]	
@SYSTEM	Branch to the operating system or pass a BS2000 command to BS2000 for execution.	F mode L mode
	[string [TO In [(inc)]]]	

Branching within EDT procedures

@CONTINUE	Create a line to which a @GOTO statement can branch or which acts as a comment line.	L mode @PROC
	[comment]	
@GOTO	Branch to the specified line.	@PROC
	In	
@IF	Check whether the EDT or DMS error switch was set during the execution of previous statements. If the condition is fulfilled, execute "text" (format 1).	L mode @PROC
	$\left. \begin{array}{l} \text{ERRORS} \\ \text{NO ERRORS} \\ \\ \text{DMS ERRORS} \\ \text{NO DMS ERRORS} \end{array} \right\} : \text{text}$	

@IF	Compare the contents of variables or lines. If the condition is fulfilled, branch to the specified line or abort the procedure (format 2).	@PROC
	$\left. \begin{array}{l} \text{[S] string1 rel string2} \\ \text{ln1 rel ln2} \\ \text{[I] int1 rel int2} \end{array} \right\} \left\{ \begin{array}{l} \text{GOTO ln} \\ \text{RETURN} \end{array} \right\}$	
@IF	Query whether a hit was found in the preceding @ON or the current work file is empty. If the condition is satisfied, the program branches to the specified line. If not, the procedure is aborted (format 3).	@PROC
	$\left. \begin{array}{l} \text{.TRUE. [rel cl]} \\ \text{.FALSE.} \\ \text{.EMPTY.} \end{array} \right\} \left\{ \begin{array}{l} \text{GOTO ln} \\ \text{RETURN} \end{array} \right\}$	
@IF	Check which task or user switches are set. If the condition is fulfilled, branch to the specified line or abort the procedure (format 4).	@PROC
	$\left. \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\} = \text{[U] int} \left\{ \begin{array}{l} \text{GOTO ln} \\ \text{RETURN} \end{array} \right\}$	
@RESET	Reset the EDT and DMS error switches.	F mode L mode @PROC

Management and execution of EDT procedures

@CREATE	Write the specified string into the specified line (format 2).	L mode @PROC
	line READ [string [...]]	
@DO	Execute the specified procedure (format 1).	F mode L mode @PROC
	procno [,] [(param [...])] [spec] [=ln1,ln2 [, [-] ln3]] [PRINT]	
@DO	Switch logging of processed lines on or off (format 2).	@PROC
	N P	

@END	Terminate processing of the current file and return to the work file which was previously current.	F mode @PROC
	[comment]	
@INPUT	Execute part or all of the specified file as a procedure (format 1).	F mode L mode
	'file' [(ver)] [range*] [:col:] [$\left. \begin{array}{l} \text{RECORDS} \\ \text{KEY} \end{array} \right\}$]][PRINT]	
@INPUT	Start an @INPUT procedure from a library element or from a file (format 2).	F mode L mode
	$\left\{ \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)][,elemtyp])} \\ \text{ELEMENT=elemname [(vers)][,elemtyp]} \\ \text{FILE=path2} \end{array} \right\}$ [PRINT]	
@NOTE	Place a comment in a procedure.	L mode @PROC
	[comment]	
@PARAMS	Define parameters in a procedure header.	@PROC
	formal [...]	
@PROC	Switch to a different work file (format 1).	L mode @PROC
	procno [comment]	
@PROC	Display the numbers of the work files (1 to 22) which are free or in use (format 2).	L mode @PROC
	FREE USED	
@SET	Set an integer variable to a specified value (format 1).	F mode L mode @PROC
	int-var = $\left\{ \begin{array}{l} [+ -] \text{ int } [+ - \text{ int}] \text{ [...]} \\ \text{SUBSTR string} \\ \text{In-var} \\ \text{LENGTH line} \\ \text{STRING string} \end{array} \right\}$	

@SET	Assign a value to a string variable (format 2).	F mode L mode @PROC
	$\text{str-ln} \left\{ \begin{array}{l} = \text{string} \\ = \text{INTERNAL} \left\{ \begin{array}{l} \text{int-var} \\ \text{ln} \\ \text{str-var} \end{array} \right\} \\ \text{[,cl]} = \text{CHAR} \left\{ \begin{array}{l} \text{int-var} \\ \text{ln-var} \\ \text{str-var} \end{array} \right\} \end{array} \right\}$	
@SET	Assign a value to a line number variable (format 3).	F mode L mode @PROC
	$\text{ln-var} = \left\{ \begin{array}{l} \text{ln} \\ \text{int-var} \\ \text{SUBSTR string} \\ \text{STRING string} \end{array} \right\}$	
@SET	Place a value in the line specified by the line number variable, starting at the specified column (format 4).	F mode L mode @PROC
	$\text{ln-var [,cl]} = \text{CHAR} \left\{ \begin{array}{l} \text{int-var} \\ \text{str-var} \\ \text{ln-var1} \end{array} \right\}$	
@SET	Assign date and time to a string variable or place them in a line (format 5).	F mode L mode @PROC
	$\left\{ \begin{array}{l} \text{str-ln} \\ \text{ln-var} \end{array} \right\} \text{[,cl]} = \left\{ \begin{array}{l} \text{DATE [ISO[4]]} \\ \text{TIME} \end{array} \right\}$	
@ZERO-RECORDS	Set empty line mode.	F mode L mode
	[ON] OFF	

Calling a user program

@RUN	Load and start another program. This program must exist as a module. EDT remains loaded.	F mode L mode
	(entry [,modlib]) [:string] [[,]UNLOAD]	
@UNLOAD	Unload a loaded program. This program must exist as a module.	F mode L mode
	(name)	
@USE	Define an external statement routine and the associated statement symbol.	F mode L mode
	COMMAND = 'usersymb' [({ entry } *) [,modlib])]	

Erasing, reading, cataloging and outputting job variables

@ERAJV	Delete job variable entries from the catalog.	F mode L mode
	str [ALL]	
@GETJV	Display the value of a job variable on the screen, write it to a work file or assign it to a string variable.	F mode L mode
	[string] [=line]	
@SETJV	Enter a job variable in the catalog or assign a value to a job variable.	F mode L mode
	{ [string1] = string2 [,...] } string1	
@STAJV	Inquire which job variables are available and what attributes they have.	F mode L mode
	[string] [TO In [(inc)]] { [SHORT] LONG [ISO4] }	

Declaring and reading S variables and list variables

@GETLIST	Read in elements of a list variable.	F mode L mode
	string [range*] [:col:]	
@GETVAR	Display contents of S variables on the screen r assign them to a string variable.	F mode L mode
	$\left\{ \begin{array}{l} \text{string [=line =int-var]} \\ \text{SYSEDT} \end{array} \right\}$	
@SETLIST	Extend and re-write a list variables.	F mode L mode
	$\text{string} \left\{ \begin{array}{l} \text{[range*] [MARK [m]]} \\ \text{str-var} \end{array} \right\} \text{ [:col:],[,]}$ <p style="text-align: center;">[MODE]=APPEND PREFIX OVERWRITE</p>	
@SETVAR	Declare S variables and assign them a value.	F mode L mode
	$\left\{ \begin{array}{l} \text{string [=string1 =int-var]} \\ \text{SYSEDT} \end{array} \right\} \text{ [,MODE=ANY NEW UPDATE]}$	

6.4 Description of the statements

@ Change current increment value and line number

@ defines a new current line number and increment value and shifts the stack entries.

The three-level EDT stack

EDT uses a three-level stack. Each stack entry consists of a pair of values for the line number and the increment. When EDT is started, the stack is empty. Entries can be placed in the stack and shifted with the aid of the @ statement.

Operation	Operands	L mode
@	[In [(inc)] [:text]]	

- In** The new current line number, e.g. 5.
 The minimum value is 0.0001, the maximum 9999.9999. If inc is omitted, the number of decimal positions in the line number implicitly defines the new increment value: for example, line number 5 implies an increment of 1 and line number 5.0 implies an increment of 0.1. In may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %,\$).
- inc** The new current increment value.
 The minimum value is 0.0001, the maximum 9999.9999.
- text** Any character string.
 If the first non-blank character in this string is
- not the EDT statement symbol (@), then any blanks following the : are regarded as part of the text.
 The following processing guidelines apply:
 - "text" is placed at the beginning of line In
 - any tab characters are interpreted
 - the current line number is incremented by the current increment value.
 - the EDT statement symbol (@), then any blanks following the : are ignored. If the next character is
 - not the EDT statement symbol, then "text" is interpreted as an EDT statement and executed immediately
 - the EDT statement symbol, then "text" is treated as a text line as described in 1), above.

- the user escape symbol, then the external statement routine is executed (cf. @USE).

Effect of @/@In

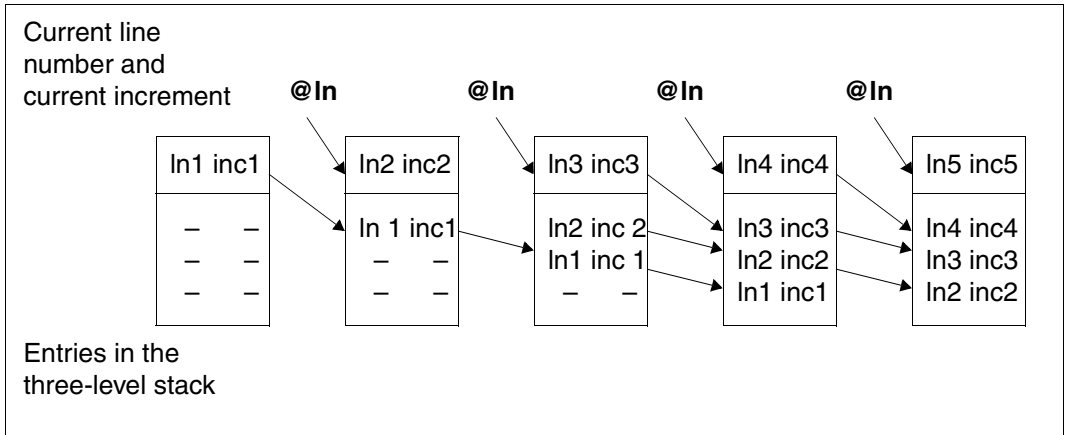


Figure 8: Effect of @In

@In [(inc)] defines a new current line number and a new current increment. The previous current line number and the previous current increment are stored in the three-level EDT stack.

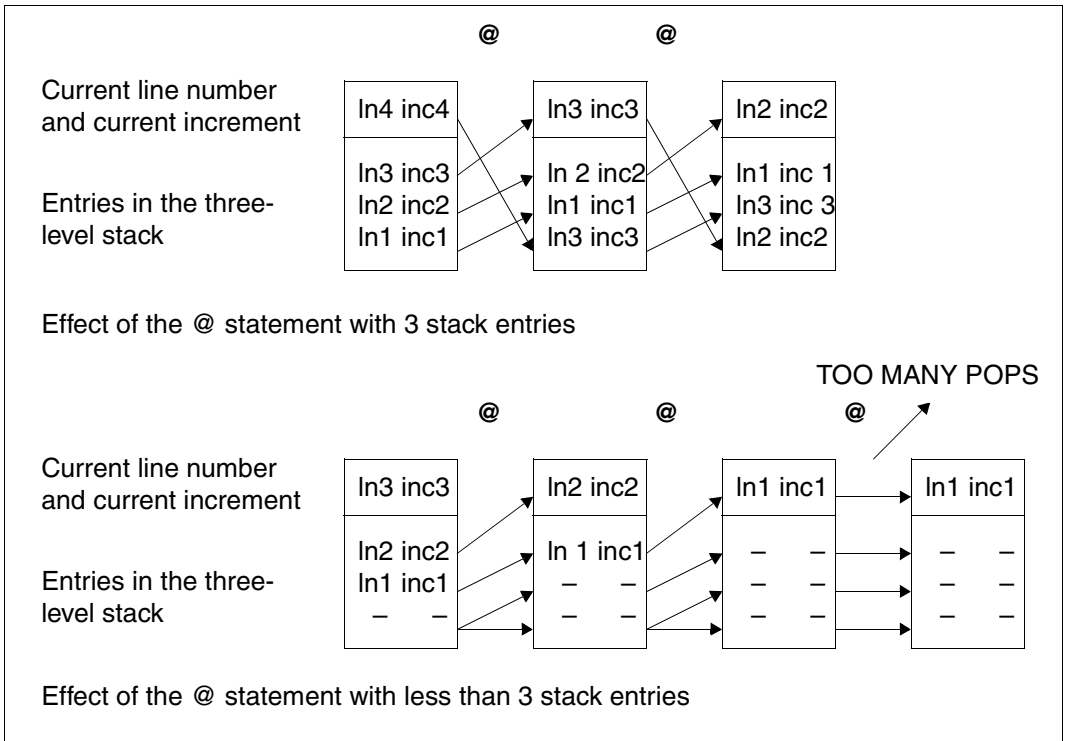


Figure 9: Shifting stack entries

@ without operands pops the top entry from the stack and uses its values as the current line number and the current increment. If no stack entry exists, an error message is issued.

The stack entries created by means of @ln are shifted as shown in [figure 9](#).

@+ Increment current line number

@+ increments the current line number by the current increment value. If sequential mode (see @EDIT) is switched off, the incremented line number becomes the new current line number. If sequential mode is on, the incremented line number becomes the new current line number only if there is no other line between it and the old current line number. If there is a line between the previous line number and the incremented line number, the number of this intermediate line becomes the new current line number. This means that existing lines cannot be skipped in sequential mode.

Operation	Operands	L mode
@+	[:text]	

text

Any character string.

If the first non-blank character in this string is

1. not the EDT statement symbol (@), then any blanks following the : are regarded as part of the text.

The following processing guidelines apply:

- "text" is placed at the beginning of line ln
 - any tab characters are interpreted
 - the current line number is incremented by the current increment value.
2. the EDT statement symbol (@), then any blanks following the : are ignored. If the next character is
 - not the EDT statement symbol, then "text" is interpreted as an EDT statement and executed immediately
 - the EDT statement symbol, then "text" is treated as a text line as described in 1), above.
 3. the user escape symbol, then the external statement routine is executed (cf. @USE).

"text" may also be @+, which makes it possible to chain this statement to itself any desired number of times.

@- Decrement current line number

@- decrements the current line number by the current increment value. If sequential mode (see @EDIT) is off, the current line number is decremented by the current increment value. If sequential mode is on, the current line number is decremented by the current increment value only if there is no other line with a number between the previous and decremented line numbers. If a line number exists between these two values, it becomes the new current line number. If there are several lines with numbers between the previous and decremented line numbers, the number of the line immediately before the current line becomes the new current line number.

Operation	Operands	L mode
@-	[:text]	

text

Any character string.

If the first non-blank character in this string is

1. not the EDT statement symbol (@), then any blanks following the : are regarded as part of the text. The following processing guidelines apply:
 - "text" is placed at the beginning of line ln
 - any tab characters are interpreted
 - the current line number is decremented by the current increment value.
2. the EDT statement symbol (@), then any blanks following the : are ignored.

If the next character is

 - not the EDT statement symbol, then "text" is interpreted as an EDT statement and executed immediately
 - the EDT statement symbol, then "text" is treated as a text line as described in 1), above.
3. the user escape symbol, then the external statement routine is executed (cf. @USE).

"text" may also be @-, which makes it possible to chain this statement to itself as many times as desired.

@: Define statement symbol

This statement permits the user to define a new statement symbol.

Operation	Operands	F mode / L mode
@:	edtsymb	

In this statement, the current statement symbol must always be specified (even in F mode); ":edtsymb" is not permitted.

edtsymb Special character to be used as the new statement symbol.
This must not be a colon or the same as the current range symbol (see @RANGE).

If "edtsymb" is not a special character, @: is rejected with the error message:
% EDT3952 INVALID SYMBOL

In the case of @: the statement symbol is mandatory, even in F mode. Indirect operand specification is not permitted.

Unambiguity of the statements can only be guaranteed if "edtsymb" is not any of the following:

- +, - or the user escape symbol (see @USE)
- <, >, # or the semicolon (;) in F mode
- \$, %, #, * or ? if the statement name is not specified for @SET or a statement is specified in the "text" operand.

Example

```

3.      @print ----- (01)
1.0000 This statement permits the user to define any desired character
2.0000 as the new statement symbol.
3.      @:! ----- (02)
3.      @print ----- (03)
4.      !print ----- (04)
1.0000 This statement permits the user to define any desired character
2.0000 as the new statement symbol.
3.0000 @print
4.

```

(01) @PRINT displays the contents of the work file.

(02) ! is defined as the new statement symbol.

(03) @PRINT is now not regarded as a statement, but as text.

(04) !PRINT displays the contents of the work file.

@AUTOSAVE Automatic saving

@AUTOSAVE is used to activate or deactivate the automatic saving of unsaved work files.

Operation	Operands	F mode / L mode
@AUTOSAVE	{ [ID=name] [[,] TIME=n] [ON] } OFF	

- name** Freely selectable identifier for the autosave files which does not exceed eight characters in length
Default: EDT
- n** Time interval in minutes between a manual or automatic save and the next automatic save. The minimum value is 0, and the maximum value is 255.
Default: 5
If TIME=0 is set, a save is performed after each dialog step.
- ON** Switches automatic saving on. A save operation writes the contents of all updated and otherwise unsaved virtual files into separate ISAM files. The line numbers are saved as an ISAM key.
- OFF** Switches off the AUTOSAVE function and deletes the autosave files.

The @AUTOSAVE statement can be issued in all work modes except batch mode, and can even be issued in an EDTSTART procedure. If issued in batch mode, the statement is ignored and no message is output. Only in interactive mode are autosaves executed.

An autosave saves all work files which have been updated since the last save operation. Autosaves are always executed after a dialog step, i.e. before the next input request, when the following conditions are satisfied:

- The AUTOSAVE function has been switched on.
- The defined time interval since the last save operation has elapsed.
- A work file has not been saved explicitly by the user since the last save operation.

At the beginning of an EDT session, the AUTOSAVE function is always deactivated.

Whenever AUTOSAVE is activated, it saves all work files which are not empty and which have been updated and not yet written back.

The names of the save files are formed as follows:

S.name.yyyy-mm-dd.hhmmss.SAVEnn

The yyyy-mm-dd.hhmmss specification is the point in time at which the AUTOSAVE function was switched on.

The nn specification is the number of the current work file.

An associated autosave file is deleted

- when the work file is empty (e.g. @DELETE)
- when the contents of the work file are saved as a file or library element. The possible statements are: @WRITE, @SAVE, @XWRITE and @CLOSE.

All autosave files are deleted

- when @AUTOSAVE OFF is issued
- when the EDT session is terminated by means of @HALT, @END, @EXEC or @LOAD
- when the user returns to the main program.

The autosave files are retained

- when EDT terminates abnormally
- when the user exits EDT by means of K2 or @SYSTEM without return.

ISAM files which are really opened are not autosaved.



Restoration of the individual work files can be initiated with:

@GET 'S.name.yyyy-mm-dd.hhmmss.SAVEnn' NORESEQ

@BLOCK Set or reset block mode

This statement switches the EDT blocked I/O mode (block mode) on or off. In block mode, the user can, with a single input:

- create several lines
- enter several statements, which are then processed sequentially.

Each line or statement must be terminated with the (terminal-specific) end-of-line character. The maximum number of characters which can be entered in one block is the number which can be displayed on one screen page. Each line in the block must not exceed 256 characters in length. For printer terminals, the maximum block size is 1020 characters.

If the entered block of statements contains an invalid statement, then this statement is output, together with the line number and an error message, at the time at which it would have been executed. The rest of the statement block is then executed.

Operation	Operands	F mode / L mode
@BLOCK @BK	{ [ON[,] [AUTOFORM]] OFF }	

ON This operand is mandatory only if AUTOFORM is to be used.

AUTOFORM Creates blank lines read from the keyboard in line mode or from a POSIX file by means of @XOPEN or @XCOPY and places one end-of-line character X'0D' in each blank line.
Data lines containing X'0D' are written into the file as lines of length 0 when written by means of @XWRITE and @CLOSE following @XOPEN.
If BLOCK mode is switched on without AUTOFORM, lines of length 0 are suppressed.

OFF Resets block mode and also the AUTOFORM function.

If an input entered in block mode contains the statement @BLOCK OFF, any subsequent statements or text lines in the block are ignored.

By default, block mode is on when EDT is started. This statement is ignored within EDT procedures.

Block mode on 816x Data Display Terminals

Block mode may be used on 816x Data Display Terminals only if a freely selectable character 'a' is defined as the end-of-line character by means of the command `MODIFY-TERMINAL-OPTIONS LINE-END-CHARACTER = C'a'`.

The default end-of-line character on these terminals is C'\'. `[LZE]` on an 8160 Data Display Terminal can be used as the end-of-line character if this terminal is actually generated as an 8160.

If the user specifies, by means of `MODIFY-TERMINAL-OPTIONS LINE-END-CHARACTER = NONE`, that no end-of-line character is to be defined, errors may occur.

@CHECK Check lines

This statement causes each line in a virtual file or a file opened by means of @OPEN which is created or updated by an EDT statement to be logged. Each affected line is displayed on the screen. @CHECK can also be used to control the checking of line lengths.

Operation	Operands	L mode
@CHECK	{ [ON] [OFF] } [,] [cl]	

ON Activates check mode. After this, any line in the virtual file or in a file opened by means of @OPEN which is created or updated by one of the following statements is displayed on the screen:

@COLUMN, @COPY, @CREATE, @MOVE, @ON, @PREFIX, @SUFFIX

OFF Deactivates check mode. This does not affect the checking of line lengths.

cl Specifies the line length for line length checking. EDT checks the length of each line which is entered or which is created by one of the following statements:

@+, @-, @IF, @LN, @SET, @UPDATE

If a line is longer than the specified length, it is still created, but EDT issues the message CHECK LINE LENGTH to inform the user that the defined line length has been exceeded. The default value for cl is 256 (the maximum permissible line length); the minimum value for cl is 1.

The value for cl can also be changed by means of @TABS.

@CHECK is effective only in L mode. Switching to F mode deactivates check mode.

If only ON or OFF is specified, the current value for cl is not changed.

If only cl is specified, the current check mode remains unchanged.

It is possible to change the current value for cl by means of @CHECK OFF,cl. If cl is not specified, its value remains unchanged, i.e. @CHECK OFF does not reset cl to its default value of 256. If the user wishes to do this, he/she must enter @CHECK OFF, 256.

@CLOSE Close and write file or library element

@CLOSE is used to

- close the ISAM file opened earlier
- write the current work file to disk or tape and close the library element
- close a POSIX file opened earlier with @XOPEN.

The work file is deleted.

Before @CLOSE is entered, a file or a library element must have been opened with @OPEN or @XOPEN.

Operation	Operands	F mode / L mode
@CLOSE	[NOWRITE]	

NOWRITE The work file is simply deleted (not written back to disk or tape). The previously opened file or library element is closed without changes. In the case of a file opened by means of @OPEN in work file 0 (see @OPEN, format 1), NOWRITE has no effect.

If specified without operands, @CLOSE has the following effect:

- real processing:
closes the file opened with @OPEN, deletes work file 0 and deletes the local entry for the file name;
- files and library elements opened with @OPEN format 2:
writes back and deletes the current work file and closes the file or library element that was opened;
- POSIX files opened with @XOPEN:
writes the work file back into the POSIX file system with the same code as it was read in with.

Any secondary keys of a NKISAM file opened with @OPEN format 2 are re-defined after the file is closed. If the fields of a secondary key have been modified inconsistently in the data area, this key is not set, and an error message is issued.

A file opened by means of @OPEN is also closed correctly (implicit @CLOSE) if, instead of @CLOSE, a @HALT, @LOAD, @EXEC statement or another @OPEN is entered.

After @CLOSE, EDT releases all memory space which is no longer in use.

Output of a new file version number after @CLOSE

If a file version number (ver) is specified in @OPEN, format 1, but "AS 'file'" is not specified, the new version number is 1 higher than the old version number.



After execution of @CLOSE, both the current line number and the increment value are set to 1. Any entries which existed in the three-level EDT stack (cf. @) are deleted.

@CODE Convert character codes

@CODE permits the user to define replacement codes for characters which cannot be displayed on certain terminals. With the aid of a code table, he/she can specify which output code is to be used for the character which is entered.

This code conversion applies only to output in the data window and in the statement line. The contents of the mark column are not converted.

A standard form of the code table exists in the module CODTAB in EDT's dynamically loadable library (see also [chapter "Installation notes" on page 593ff](#)). This standard code table can be output on the screen or modified with @CODE. The code table does not take effect until the code functions have been switched on with one of the three formats of @CODE.

Format	Statement	Meaning
1	@CODE In,SHOW	Display the code table in a screen line. Transfer the code table from a screen line to module CODTAB. Display and modify the code table. Activate code conversion.
2	@CODE In	Activate code conversion, using the code table in the specified record (In).
3	@CODE SHOW ON OFF	Display the code table. Activate code conversion. Deactivate code conversion.

The default code table:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					␣	&	-									0
1							/		a	j			A	J		1
2									b	k	s		B	K	S	2
3									c	l	t		C	L	T	3
4									d	m	u		D	M	U	4
5									e	n	v		E	N	V	5
6									f	o	w		F	O	W	6
7								~	g	p	x		G	P	X	7
8									h	q	y		H	Q	Y	8
9									i	r	z		I	R	Z	9
A					'	!	^	:								
B					.	\$,	#	[{					
C					<	*	%	@	\							
D					()	_	']		}					
E					+	;	>	=								
F							?	"								

The code positions for characters which cannot be displayed are set to X'07' and are displayed on the screen as smudge characters (on the 3270 Data Display Terminal: X'41'). The code table should be unique, i.e. a code other than X'07' should not appear more than once. It is the user's responsibility to make sure that the code table is not ambiguous. EDT does not check the table for ambiguous codes.

The default code table in module CODTAB carries out the following code conversions:

Keyboard	Character	File
X'FB'	(ä) {	X'AB'
X'4F'	(ö)	X'AC'
X'FD'	(ü) }	X'AD'
X'BB'	(Ä) [X'8B'
X'BC'	(Ö) \	X'8C'
X'BD'	(Ü)]	X'8D'
X'FF'	(ß) -	X'67'

Using format 2, the user can create a code table without having to know the internal coding for a given character in the computer.

Effect when specified together with LOWER OFF

Output (file → screen):

The record is converted on the basis of the code table. Lowercase letters are converted to smudge characters for screen output. The record is displayed on the screen.

Input (screen → file):

Any lowercase letters which are entered are converted to uppercase letters. Code conversion is carried out on the basis of the current code table.



The code for the tab character must not be converted.

@CODE and extended character sets

It is not a good idea to use both XHCS and @CODE. The @CODE statement in, however, still supported and co-exists with XHCS.

The @CODE statement does not change the coded character set (CCS) selected in EDT. For this reason, all files including library elements are supplied with the coded character set name (CCSN) as a code attribute. The selected CCSN is also used for input/output via WRTRD, WROUT and RDATA.

@CODE (format 1) Display code table and activate code function

A record containing a code table is displayed in a suitable format on the screen and can be modified. Code conversion is activated.

Operation	Operands	F mode / L mode
@CODE	In, SHOW	

In The line number of a record, which must be 256 bytes long. The line number can also be specified as a line number variable or as a symbolic line number.
If a record with the specified line number exists, it is transferred to module CODTAB as the default code table.
If there is no record with the specified line number, a record with this number and a length of 256 bytes is created and the default code table from module CODTAB is transferred to this record.

SHOW The code table is displayed in a suitable format on the screen, screen can be modified. Code conversion is activated.

Note, that the code table must remain unambiguous, i.e. each character other than NULL may appear only in the code position. NULL may appear in several code positions and causes the code X'07' (smudge character on the screen) to be placed in each position

The modified code table is then included in the work file as a record with line number "In".

@CODE (format 2) Deactivate code function

Code conversion is activated, using the record with the specified line number as the code table.

Operation	Operands	F mode / L mode
@CODE	In	

In The line number of the record containing the code table to be used. This record must exist and must be 256 bytes long. The line number can also be specified as a line number variable or as a symbolic line number.

@CODE (format 3) Activate or deactivate code function

This format activates or deactivates code conversion or permits the user to display and modify the code table.

Operation	Operands	F mode / L mode
@CODE	[ON] SHOW OFF	

ON Activates code conversion with the current code table (initially the code table from module CODTAB). If the code table is switched by means of @CODE SHOW, the newly selected table becomes the current table after the next @CODE ON.

SHOW Displays the current code table, which can then be modified as in format 1. Code conversion is activated only by a subsequent @CODE ON. The modified code table is deleted if EDT is terminated or if @CODE OFF is entered.

OFF Deactivates code conversion and deletes the current code table.

When EDT is started, the default setting is OFF.

Example

```

1.00 MIT DIESER  BUNG M CHTEN WIR IHNEN ERKL REN, WIE SIE.....
DCE4CCCECD48CEDC4D8CCECD4ECD4CCDCD4CDD8DCD64ECC4ECC
49304952590D245704C38355069909855505923B955B06950295
-----1-----2-----3-----4-----5-----6-----7--
2.00 DIE CODE-ANWEISUNG BEN TZEN K NNEN. ....
CCC4CDCC6CDECCEEDC4CCD8EECD4D8DDCD44
4950364501565924570255D395502C5555B0
-----1-----2-----3-----4-----5-----6-----7--
3.00 .....
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
-----1-----2-----3-----4-----5-----6-----7--

code show ; code on.....0001.00:001(0)
```

The text in lines 1.00 and 2.00 contains some German “umlaut” characters, which are displayed as smudges. Hexadecimal mode is active.

@CODE SHOW is used to display the current code table on the screen. This is to be modified and code conversion is to be activated.

```

***                C O D E - M O D E
*****
  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0   .      † & -          0
1           /   a j      A J  1
2           b k s     B K S 2
3           c l t     C L T 3
4           d m u     D M U 4
5           e n v     E N V 5
6           f o w     F O W 6
7           ~   g p x   G P X 7
8           h q y     H Q Y 8
9           i r z     I R Z 9
A           ! ^ :
B           . $ . # [ {
C           < * % @ \   }
D           ( ) '     ]
E           + ; > =
F           ? "

*****
PRESS K1 OR DUE FOR RETURN

```

Position 8C of the code table is now set to the \ character, which means that the umlauts are displayed as follows:

Ä → X'8B' → [
Ö → X'8C' → \
Û → X'8D' →]

```

1.00 MIT DIESER JBUNG M\CHTEN WIR IHNEN ERKLAREN, WIE SIE.....
DCE4CCCECD48CEDC4D8CCECD4ECD4CCDCD4CDDD8DCD64ECC4ECC
49304952590D245704C38355069909855505923B955B06950295

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7---
2.00 DIE CODE-ANWEISUNG BENJTZEN K\NNEN.....
CCC4CDCC6CDECCEDC4CCD8EECD4D8DDCD
4950364501565924570255D395502C5555
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7---
3.00 .....
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7---

```

Since code conversion was activated by means of @CODE ON, the umlauts are now represented by the characters [, \ and].

Printout of the two converted lines:

```
1.00    MIT DIESER ÜBUNG MÖCHTEN WIR IHNEN ERKLÄREN, WIE SIE  
2.00    DIE CODE-ANWEISUNG BENÜTZEN KÖNNEN.
```

Translation of German text:

This exercise shows you how to use the @CODE statement.

@CODENAME Switch explicitly to different CCSN

This statement selects the desired coded character set (CCS) for the EDT session.

In systems in which the XHCS subsystem is not installed, @CODENAME is rejected with an error message.

Operation	Operands	F mode / L mode
@CODENAME	[name]	

name Name of the coded character set.

If name is omitted, a switch is made to the coded character set EDF03IRV.

The following conditions must be met before a switchover to the desired coded character set can be made:

- The coded character set name (CCSN) must be included in the list of CCSNs valid for the data display terminal.
- The EDT work files must not contain data with a different CCSN (i.e. all EDT work files are empty).
- The coded character set (CCS) must not be an ISO code and must not be a 7-bit code other than EDF03IRV.

If these conditions are not met, @CODENAME is rejected and the currently selected CCSN remains valid.

A statement specifying a switchover to the currently valid CCSN is ignored.

@CODENAME may not be specified in @INPUT and @DO procedures.

@COLUMN Insert text or delete blanks at end of line

@COLUMN inserts a character string into existing lines, starting at a specified column. The user can specify whether or not the new string is to overwrite any existing text in the lines.

This statement also searches the lines from right to left, starting at column 256, and deletes all blanks it finds. The search is terminated when the first non-blank character is found. If a line contains only blanks, it is deleted by this statement (see note below).

Operation	Operands	F mode / L mode
@COLUMN	cl ON range { [CHANGE] INSERT } [:] string	

- cl The column at which the insert or overwrite function is to begin.
- range The line range, specified as:
 - one or more line numbers separated by commas (e.g. 4,6,15)
 - one or more line ranges separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30)

A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- CHANGE Specifies that the text beginning at column cl is to be overwritten by the new character string.
- INSERT Specifies that the new character string is to be inserted in the existing text, starting at column cl.
- :
- string The character string which is to overwrite the existing text or is to be inserted.

This may be specified either

 - explicitly, enclosed in single quotes, or
 - implicitly, in the form of a line number, a line number variable or a string variable (in each case with a domain if desired).

If the column at which the text is to be inserted is to the right of the current end-of-line, the intervening columns are filled with blanks.

- i – This statement can be used to delete empty lines or to delete blanks at the end of existing lines. If, for example, the user knows that the lines can never be longer than 80 characters, he/she could delete all blanks at the end of each line by means of @COLUMN 81 ON & ' '. This would first insert a blank in column 81 of each line, but the subsequent “search and delete blanks” operation would then delete it, together with any blanks to the right of it.
- Shifting or inserting whole blocks of columns is not supported by @COLUMN. However, the problem can be solved with the aid of a simple EDT procedure (see the examples in [chapter “EDT procedures” on page 141ff](#)).

Example

```

1.00 126790.....
2.00 348.....
3.00 .....

column 3 on 1:2 .....0001.00:001(0)

```

The contents of line 2.00 are to overwrite the contents of line 1.00, starting at column 3.

```

1.00 123480.....
2.00 348.....
3.00 .....

column 5 on 1 insert '567' .....0001.00:001(0)

```

The character string '567' is to be inserted in line 1.00, starting at column 5.
No characters in line 1.00 are overwritten.

```

1.00 123456780.....
2.00 348.....
3.00 .....

```

@COMPARE Compare work files line-by-line

@COMPARE, which has two formats, permits the user to compare all or part of the contents of two work files with each other.

@COMPARE (format 1) Compare two work files

@COMPARE causes EDT to compare all or part of the contents of two work files with each other.

The user can specify that the results are to be

- displayed on the screen,
- placed in a work file, or
- printed on SYSLST.

Operation	Operands	F mode / L mode
@COMPARE	[procno1] :rng*1 WITH [procno2] :rng*2 [,[int1] [(int2)] [LIST [ln [(inc)]]]]	

procno1, procno2

The numbers (0 to 22) of the two work files which are to be compared. If one of these files is work file 0 and contains a file opened by means of @OPEN, format 1, @COMPARE is rejected with an error message. If procno1 or procno2 is omitted, the current work file is used by default.

rng*1

The line range in work file 1 (procno1), specified as:

- a single line (e.g. 6) or
- several consecutive lines (e.g. 8-20).

The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.

rng*2

The line range in work file 2 (procno2), specified as:

- a single line (e.g. 6) or
- several consecutive lines (e.g. 8-20).

The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.

int1	<p>Specifies how many lines EDT is to examine in order to find a line range which is identical in both files. If EDT examines int1 lines without finding at least int2 consecutive lines which are identical in both files, it aborts the compare operation.</p> <p>The following applies to int1 and int2: $int2 \leq int1 \leq 65535$. The default value for int1 is 10.</p>
int2	<p>Specifies how many consecutive lines in both files must be identical before EDT regards the ranges formed by these lines as identical.</p> <p>The following applies to int1 and int2: $int2 \leq int1 \leq 65535$. The default value for int2 is 1.</p>
LIST	<p>If In is not specified, EDT prints the results of the comparison on SYSLST. For each line where no match is found, EDT prints the line number and the first 51 characters of the line.</p> <p>If In is specified, EDT writes the result into the current work file, providing this file is not one of the files being compared.</p> <p>If LIST is omitted, EDT displays the results on the screen.</p> <p>If both LIST and In are omitted, EDT simply displays the numbers of the lines for which it finds no match. The first 51 characters of each such line are not displayed.</p>
In	<p>EDT is to write the results of the comparison into the current work file (see the LIST operand). In specifies the number of the line which is to contain the first line of the result. EDT writes the result into the file in the same format as it uses for screen display.</p>
inc	<p>The increment value for the line numbers which follow line In. If this operand is omitted, EDT uses the increment value implied by the value specified for In.</p>

EDT starts the comparison at the beginning of the specified line ranges. If it finds a pair of non-matching lines, it skips one or more lines in one or both files; the number of lines skipped may be different in the two files. If EDT then finds int2 consecutive lines which are identical in both files, it aligns the two files on these line pairs for subsequent comparison.

1. A line pair is identical if the contents and the lengths of the two lines are the same. The line numbers are ignored by the compare function.
2. If the same value is specified for int1 and int2, EDT will find no matching line ranges in the two files if at least one line pair in the files is not identical.
3. EDT informs the user of the results of the comparison by means of the following messages:

- Messages for intermediate results; further messages follow

EXTRA LINES IN 1ST FILE

```

ln
.
.
.
ln

```

In file 1 EDT has skipped the lines whose numbers are listed. It has not skipped any lines in file 2. The lines skipped in file 1 are followed by int2 lines which are identical with int2 consecutive lines in file 2.

$1 \leq \text{number of line numbers listed} \leq \text{int1} - \text{int2}$

EXTRA LINES IN 2ND FILE

```

ln
.
.
.
ln

```

In file 2 EDT has skipped the lines whose numbers are listed. It has not skipped any lines in file 1. The lines skipped in file 2 are followed by int2 lines which are identical with int2 consecutive lines in file 1.

$1 \leq \text{number of line numbers listed} \leq \text{int1} - \text{int2}$

NON-MATCHING LINES

```

ln
.
.
.
ln

```

In file 1 EDT has skipped the lines whose numbers are listed in column 1 and in file 2 the lines whose numbers are listed in column 2. The lines skipped in the two files are followed by int2 pairs of identical lines.

$1 \leq \text{number of line numbers listed in each column} \leq \text{int1} - \text{int2}$.

- Messages upon completion of the comparison

```
EXTRA LINES IN 1ST FILE
 1n
 .
 .
 .
 1n
REACHED LIMIT ON BOTH FILES
```

EDT has found int2 consecutive lines in file 1 which match the last int2 lines in file 2. It has listed the numbers of the lines which were “left over” in file 1 at the end of the comparison.

$1 \leq \text{number of line numbers listed} \leq \text{int1}$

```
EXTRA LINES IN 1ST FILE
 1n
 .
 .
 .
 1n
REACHED 2ND FILE LIMIT
```

EDT has found int2 consecutive lines in file 1 which match the last int2 lines in file 2. At the end of the comparison, there were more than int1 lines left in file 1. EDT has listed the numbers of the first int1 lines which were “left over” at the end of the comparison.

```
EXTRA LINES IN 2ND FILE
 1n
 .
 .
 .
 1n
REACHED LIMIT ON BOTH FILES
```

EDT has found int2 consecutive lines in file 2 which match the last int2 lines in file 1. It has listed the numbers of the lines which were “left over” in file 2 at the end of the comparison.

$1 \leq \text{number of line numbers listed} \leq \text{int1}$

EXTRA LINES IN 2ND FILE

1n

.

.

.

1n

REACHED 1ST FILE LIMIT

EDT has found int2 consecutive lines in file 2 which match the last int2 lines in file 1. At the end of the comparison, there were more than int1 lines left in file 2. EDT has listed the numbers of the first int1 lines which were "left over" at the end of the comparison.

NON-MATCHING LINES

1n 1n

.

.

.

1n 1n

NOTHING SEEMS TO MATCH

EDT has listed the numbers of the last int1 lines which it has examined in the two files. Since there are not at least int2 identical line pairs in this range, has EDT aborted the comparison. In neither of the two files has EDT reached the end of the line range specified for the comparison.

NON-MATCHING LINES

1n 1n

.

.

.

1n 1n

REACHED LIMIT ON BOTH FILES

EDT has reached the end of the line ranges specified for comparison in the two files. Column 1 contains the numbers of the last lines in file 1, column 2 the numbers of the last lines in file 2. There are not at least int2 consecutive lines which are identical in both files.

1 ≤ number of line numbers listed in each column ≤ int1

NON-MATCHING LINES

```

ln      ln
.
.
.
ln      ln

```

REACHED 1ST FILE LIMIT

EDT has reached the end of the line range in file 1 specified for the comparison. Column 1 contains the numbers of the last lines in the range in file 1. Within these lines, there are not at least int2 consecutive lines which match int2 consecutive lines in file 2 (relative to the int1 lines in file 2 whose numbers are listed in column 2).

NON-MATCHING LINES

```

ln      ln
.
.
.
ln      ln

```

REACHED 2ND FILE LIMIT

EDT has reached the end of the line range in file 2 specified for the comparison. Column 2 contains the numbers of the last lines in the range in file 2. Within these lines, there are not at least int2 consecutive lines which match int2 consecutive lines in file 1 (relative to the int1 lines in file 1 whose numbers are listed in column 1).

$1 \leq$ number of line numbers listed in column 2 \leq int1

REACHED LIMIT ON BOTH FILES AT SAME TIME

EDT has reached the end of the ranges specified for comparison in both files. The last int2 lines in both files are identical.

Example

```
1.      @PROC 1
1.      @READ 'PROC-FILE.1' ----- (01)
7.      @PRINT
1.0000 AAAAAA
2.0000 BBBBBB
3.0000 CCCCCC
4.0000 UUUUUU
5.0000 VVVVVV
6.0000 WWWWWW
7.      @END
1.      @PROC 2
1.      @READ 'PROC-FILE.2' ----- (02)
8.      @PRINT
1.0000 AAAAAA
2.0000 BBBBBB
3.0000 ZZZZZZ
4.0000 AAAAAA
5.0000 BBBBBB
6.0000 CCCCCC
7.0000 UUUUUU
8.      @END
1.      @COMPARE 1:1-6 WITH 2:1-7, 5(2) ----- (03)
EXTRA LINES IN 2ND FILE
      3.0000
      4.0000
      5.0000
EXTRA LINES IN 1ST FILE
5.0000
6.0000
REACHED LIMIT ON BOTH FILES
1.      @COMPARE 1:1-6 WITH 2:1-7, 5(3) ----- (04)
NON-MATCHING LINES
1.0000      1.0000
2.0000      2.0000
3.0000      3.0000
4.0000      4.0000
5.0000      5.0000
NOTHING SEEMS TO MATCH
1.      @COMPARE 1:1-6 WITH 2:1-7, 6(3) ----- (05)
EXTRA LINES IN 2ND FILE
      1.0000
      2.0000
      3.0000
EXTRA LINES IN 1ST FILE
5.0000
```


6.0000
REACHED LIMIT ON BOTH FILES
1.

- (01) The SAM file PROC-FILE.1 is read into work file 1.
- (02) The SAM file PROC-FILE.2 is read into work file 2.
- (03) If examination of five lines in each file does not result in at least two consecutive matching line pairs, the comparison is to be aborted. All lines in both files are compared.
- (04) The @COMPARE statement used in step (03) is modified slightly and entered again. This time, at least three consecutive matching line pairs must be found. Since this does not occur, EDT aborts the comparison.
- (05) The @COMPARE statement used in step (04) is modified slightly and entered again. This time, the comparison is to be aborted only after six lines have been compared without success. The comparison is executed for all lines of both files.

@COMPARE (format 2) Compare two work files line by line

@COMPARE compares the contents of two work files line by line. EDT places the results of the comparison in a third work file, which is cleared before the results are stored.

The results returned by EDT are:

- a header line containing the file name. The header line may also display the following:
 - the name of a library element opened with @OPEN format 2 or of a file,
 - the name of a POSIX file opened with @XOPEN or
 - a local @FILE entry, if one exists.
- the line number and content of each record which occurs in only one of the two files being compared. Records exceeding 239 characters in length are truncated. The position of a given record's line number in column 1 or in column 2 under the LINE#(adatnr) header indicates which of the two files being compared contains that record.
- the line numbers of records which are identical in the two files (e.g. 0001.00=0006.00). If several consecutive records in the two files are identical (range of identical lines), only the first and last pairs of line numbers of each range are shown (see example).

Operation	Operands	F mode / L mode
@COMPARE	[[procno1] [WITH]] procno2 [LIST [procno3]] [,procno4]	

- procno1 The number of the first work file to be compared.
If procno1 is omitted, the current work file is used by default.
- procno2 The number of the second work file to be compared. Specification of procno2 is mandatory.
At least one work file (procno1 or procno2) must be specified.
- LIST If LIST is specified, the results are placed in work file procno3; if procno3 is omitted, the results are output to SYSLST. If LIST is omitted, the results are placed in work file 9 in F mode, displayed on the screen in L mode, or output to SYSOUT for procedures.
- procno3 The work file which is to be used to hold the results of the comparison. This file is cleared before it is used.
- procno4 The work file which EDT is to use as a scratch file. This file is cleared before it is used. If procno4 is omitted, work file 10 is used by default.

Different work files must be specified for procno1, procno2, procno3 and procno4.

If all of the lines to be compared are equal or different, the following messages are issued:

```
% EDT0291 ALL LINES ARE EQUAL
% EDT0290 ALL LINES ARE DIFFERENT
```

If the results are output in procno3, the following message is issued:

```
% EDT0297 COMPARE RESULT IN WORK FILE (procno3)
```

If one of the two files being compared is the work file into which the results are to be written, the following message is output:

```
% EDT5350 COMPARE RESULT CANNOT BE SHOWN
```

The results of the comparison cannot be displayed because the file which was specified to receive the results is occupied.

If one of the files being compared is work file 0, @OPEN, format 1 must not be used to really open any ISAM file.

@COMPARE causes all record marks to be deleted.

Querying comparison results

To enable users to query the comparison results within procedures, EDT sets not only the messages % EDT0290 and % EDT0297, but also the EDT error (for information on querying error switches, see @IF formats 1 and 3).

Differentiation:

	EDT error switch	Work file procno3
% EDT0291	not set	empty
% EDT0290	set	empty
% EDT0297	set	not empty

Before @COMPARE is used to compare files, it is necessary to issue @RESET to reset the EDT error switch and to delete work file procno3.



If, in either of the files to be compared, the fourth digit after the decimal point is not 0 (e.g.: 0.0009), @COMPARE is aborted with the message % EDT5352 @COMPARE ABORTED - PLEASE RENUMBER. The reason for this is that this digit in the line number is reserved for internal use.

Example

```
1.00 X.....
2.00 Y.....
3.00 Z.....
4.00 G.....
5.00 H.....
6.00 A.....
7.00 B.....
8.00 C.....
9.00 J.....
10.00 K.....
11.00 D.....
12.00 E.....
13.00 .....
14.00 .....
15.00 .....
16.00 .....
17.00 .....
18.00 .....
19.00 .....
20.00 .....
21.00 .....
22.00 .....
23.00 .....
1 .....0001.00:001(2)
```

Switch from work file 2 to work file 1.

```
1.00 A.....
2.00 B.....
3.00 C.....
4.00 D.....
5.00 E.....
6.00 F.....
7.00 G.....
8.00 H.....
9.00 I.....
10.00 J.....
11.00 K.....
12.00 .....
13.00 .....
14.00 .....
15.00 .....
16.00 .....
17.00 .....
18.00 .....
19.00 .....
20.00 .....
21.00 .....
22.00 .....
23.00 .....
compare 2 with 1 liste 3; 3.....0001.00:001(1)
```

Compare work file 2 with work file 1 and place the results in work file 3. Then switch to work file 3.

```

0.10 LINE#( 1)          FILENAME:.....
0.20          LINE#( 2) FILENAME:.....
0.30          0001.00 X.....
0.40          0002.00 Y.....
0.50          0003.00 Z.....
0.60          0004.00 G.....
0.70          0005.00 H.....
0.80 0001.00=0006.00.....
0.90 0003.00=0008.00.....
1.00 0004.00          D.....
1.10 0005.00          E.....
1.20 0006.00          F.....
1.30 0007.00          G.....
1.40 0008.00          H.....
1.50 0009.00          I.....
1.60 0010.00=0009.00.....
1.70 0011.00=0010.00.....
1.80          0011.00 D.....
1.90          0012.00 E.....
2.90 .....
3.90 .....
4.90 .....
% EDT0297 RESULT OF COMPARE IN PROCFILE 3
.....0000.10:001(3)

```

Work file 3 contains the results of comparing work file 2 with work file 1.

@CONTINUE Define branch destination

@CONTINUE is used to create a line in an EDT procedure to which a branch can be executed by means of @GOTO. Execution of this statement does not cause any further processing action, which means that it can be used (like @NOTE) for inserting comment lines in EDT procedures.

Operation	Operands	L mode / @PROC
@CONTINUE	[comment]	

comment Any desired comment.

The main use for this statement is to define a last line in an EDT or INPUT procedure. Such a line is mandatory only if an EDT procedure is started as an external loop by means of a loop symbol (e.g. @DO 5,!=%,\$). In such cases, it is always necessary to branch to the end of the procedure in order to start the next pass.

Example

```

6.      @PRINT
1.0000 WITH EDT
2.0000 ANYONE WHO KNOWS
3.0000 THE STATEMENTS CAN
4.0000 WRITE HIS PROGRAM ONE
5.0000 PROCEDURE AT A TIME
6.      @PROC 1
1.      @1.00
1.00   @ @CON OBJECTIVE: IF A LINE CONTAINS 'W' ----- (01)
1.01   @ @CON          DISPLAY IT ON THE SCREEN
1.02   @ @ON * FIND 'M'
1.03   @ @IF .FALSE. GOTO 2
1.04   @ @PRINT *
1.05   @2.00
2.00   @ @CONTINUE ----- (02)
2.01   @END
6.      @DO 1,*=1,$ ----- (03)
1.0000 WITH EDT
2.0000 ANYONE WHO KNOWS
4.0000 WRITE HIS PROGRAM ONE
6.

```

- (01) Here, @CONTINUE is used for inserting comments.
- (02) Here, @CONTINUE is mandatory, since there must be a last line in the procedure to which a branch can be executed.
- (03) The procedure is started in work file 1 by means of @DO and a loop symbol.

@ CONVERT Convert data lines to uppercase/lowercase

This statement can be used to convert a line range to uppercase or lowercase letters, as appropriate. In contrast to the @LOWER statement, it is the existing contents of the current work file that are converted, and not the input.

Operation	Operands	F mode / L mode
@CONVERT	[[range] T[O]=] {UPPER LOWER}	

range Line range consisting of one or more lines. This line range can be specified either symbolically or by means of variables for the line numbers (#L0 to #L20). Furthermore, specification of character string variables (#S0 to #S20) is also possible.

All data lines within "range" in the current work file are converted. If nothing is specified for "range", the entire work file is converted.

TO= Determines the direction in which conversion takes place and thus also determines the conversion table. This keyword can be omitted if no line range is specified.

UPPER EDT converts the lowercase letters a, ..., z to the uppercase letters A, ..., Z. The conversion table used is the one used when @LOWER OFF is input. If XHCS is installed on the system, the conversion table associated with the coded character set (CCS) is used for conversion. If XHCS is not available, EDT uses a default table based on EBCDIC.DF.03. The German umlaut characters ä, ö and ü are not converted in this case.

LOWER The uppercase letters A, ..., Z are converted to the lowercase letters a, ..., z. The conversion table is created by inverting the table used for conversion in the other direction (lowercase to uppercase).

Either UPPER or LOWER must be issued.

Special characters, digits and arithmetic characters are not affected.

Interaction with @CODE

Activating the code function (@CODE) has no affect on data conversion.

@COPY Copy data

@COPY has two formats, which offer the following copy facilities:

- copying a line or a line range from any work file into the current work file (format 1)
- copying a program library element (format 2).

Unlike @MOVE, the @COPY statement leaves the line range being copied (the source) unchanged.

@COPY (format 1) Copy a line or range of lines

A line or a line range is copied from any work file into the current work file.

It is not possible to copy from a work file which is currently being executed as an EDT procedure (see @DO), i.e. which is an active work file.

Operation	Operands	F mode / L mode
@COPY	rng [(procno)] [TO ln1 [(inc)] [:] [ln2]] [...]	

If lines are to be copied from the current work file, the operands TO and ln1 must always be specified. If lines are to be copied from another work file and these operands are omitted, the copied lines retain the line numbers they had in the source file.

rng A line range, specified as:
 - a single line number (e.g. 6)
 - several consecutive line numbers (e.g. 8-20)

The line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.

The symbolic line numbers are based on the current work file, i.e. the values of the symbolic line numbers correspond to the line numbers in the current work file and not in the work file from which the copy is being made.

procno The number (0-22) of the work file from which the lines are to be copied.

ln1 The number of the first line of the target range.
 EDT calculates the numbers of the subsequent lines in the target range by incrementing this line number by the increment value specified for this range. The minimum value is 0.0001, the maximum 9999.9999.
 If inc is not specified, EDT uses the increment value implied by the number of decimal places in the line number: for example, 5 implies an increment of 1 and 5.0 implies an increment value of 0.1.
 ln1 may also be specified as a line number variable or symbolically.

inc	The current increment value for the target range. The minimum value is 0.0001, the maximum 9999.9999.
:	This delimiter may be omitted if inc is specified, as this clearly separates the operands ln1 and ln2.
ln2	The number of the last line in the target range. @COPY copies line by line. When this upper limit is reached, the copy operation is terminated, even if there are still lines in the source range which have not been copied. The minimum value is 0.0001, the maximum 9999.9999. ln2 may also be specified as a line number variable or symbolically. If ln2 is not specified, the copy operation may overwrite lines which the user wanted to keep.

Duplicating line ranges

@COPY can be used to duplicate a range of lines if the source and target ranges overlap (see example 2).

If inc is set too large, or if ln2 is not specified, lines in the target range may inadvertently be overwritten.

Transferring without changing the line numbers

When transferring lines from other work files into the current one, the line numbers are retained as long as "TO ln1 ..." is not specified.

When transferring from the current work file, "TO ln1" must always be specified.

Current increment value and line number

@COPY does not change the current increment value. The operand inc simply determines the increment used between the copied records. It does not refer to the current increment value.

The current line number is changed in L mode only if a line with a number greater than the currently highest line number is created.

Example 1

```

1.00 NOW.....
2.00     WE CAN.....
3.00         COPY.....
4.00 .....

copy 1 to 7 ; copy 2 to 5 ; copy 1-3 to 30.1 (5).....0001.00:001(0)

```

The three @COPY statements are to copy as follows:

- line 1 → line 7
- line 2 → line 5 and
- line range 1 - 3
- a line range starting at line 30.1 with the explicit increment value 5

```

1.00 NOW.....
2.00     WE CAN.....
3.00         COPY.....
5.00     WE CAN.....
7.00 NOW.....
30.10 NOW.....
35.10     WE CAN.....
40.10         COPY.....
41.10 .....

```

Example 2

@COPY can be used to duplicate line ranges if the source and target ranges overlap. In this example, line 1 is to be duplicated.

```

1.00 111.....
2.00     222.....
3.00         333.....
4.00             444.....
5.00                 555.....
6.00 .....

copy 1-2 to 1.5.....0001.00:001(0)

```

This statement copies the line range comprising lines 1 and 2 into the range starting at line number 1.5 with the implicit increment value 0.1. EDT first copies line 1 into line 1.5. This new line lies within the specified source range, and

is therefore copied into line 1.6 (the implicit increment value is 0.1). Similarly, line 1.6 is copied into line 1.7, ... , 1.9 into 2.0 (overwriting the contents of line 2). Finally, line 2.0 is copied into line 2.1.

```

1.00 111.....
1.50 111.....
1.60 111.....
1.70 111.....
1.80 111.....
1.90 111.....
2.00 111.....
2.10 111.....
3.00      333.....
4.00      444.....
5.00      555.....
6.00 .....
7.00 .....
8.00 .....
9.00 .....
10.00 .....
11.00 .....
12.00 .....
13.00 .....
14.00 .....
15.00 .....
16.00 .....
17.00 .....
copy 3-5 to 4.1 : 5.....0001.00:001(0)

```

The line range 3 to 5 is to be copied into the line range 4.1 to 5 with the implicit increment value 0.1.

EDT first copies line 3 into line 4.1 and line 4 into line 4.2. Both of these lines are within the specified source range, which means that line 4.1 is copied into line 4.3, line 4.2 into 4.4, ... , 4.8 into 5.0 (overwriting the contents of line 5). Lines 4.9 and 5.0 are not copied, as the upper limit of the target range has been reached.

```

1.00 111.....
1.50 111.....
1.60 111.....
1.70 111.....
1.80 111.....
1.90 111.....
2.00 111.....
2.10 111.....
3.00      333.....
4.00      444.....
4.10      333.....
4.20      444.....
4.30      333.....
4.40      444.....
4.50      333.....
4.60      444.....
4.70      333.....
4.80      444.....
4.90      333.....
5.00      444.....
6.00 .....
7.00 .....
8.00 .....
.....0001.00:001(0)

```

@COPY (format 2) Copy a library element or file

This format of the @COPY statement copies a complete library element or file into the current work file and then closes the library element or file.

Operation	Operands	F mode / L mode
@COPY	$\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT]=elemname [(vers)][,elemtyp])} \\ \text{ELEMENT=elemname [(vers)][,elemtyp]} \\ \text{FILE=path2} \end{array} \right\}$ $[\left. \begin{array}{l} \text{BEFORE} \\ \text{AFTER} \end{array} \right\} \text{In}]$	

LIBRARY = path1 ([ELEMENT]=elemname [(vers)][,elemtyp])

The name of the library and of the desired element.

ELEMENT = elemname [(vers)][,elemtyp]

The name of the desired element, without a library name.

In this case, the library name must have been preset by means of @PAR.

path1 The library name. path1 may also be specified by means of a string variable. If path1 is omitted, the default library specified by means of @PAR LIBRARY is used.

elemname The element name. elemname may also be specified by means of a string variable.

vers The version number of the desired element (see the "LMS" manual [14]). If vers is not specified or if *STD is specified, the highest available version of the element is selected.

elemtyp The element type. elemtyp may also be specified by means of a string variable. Permissible type entries: S, M, P, J, D, X, *STD or a user-defined type names with appropriate base type. If no type is specified, the value preset in @PAR ELEMENT-TYPE will be used.

Users who specify a user-defined type name are responsible for ensuring that its associated base type corresponds to one of the permissible types S, M, P, J, D or X.

Type	Contents
S	Source programs
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format

*STD Default value

Type S is the default value when EDT is started. Any other valid type specification can be defined as the default value by means of @PAR.

FILE = path2	This operand is used to copy a BS2000 file.
path2	Name of the file to be copied. path2 may also be specified by means of a string variable.
BEFORE	The library element or file is inserted <i>before</i> the specified line number. If @PAR RENUMBER=ON is specified, any existing line numbers are renumbered as necessary. If @PAR RENUMBER=OFF is specified, it is not possible to copy an element before line number 0.01.
AFTER	The library element or file is inserted <i>after</i> the specified line number. It is not possible to copy an element after line number 9999.99.
In	The number of the first line in the target range. EDT calculates the numbers of the following lines in the target range by incrementing this line number by the current increment value for this range. The minimum value is 0.0001, the maximum 9999.9999. In can also be specified as a line number variable or symbolically. If In is omitted, the element is copied at the end of the current work file.

Calculation of line numbers

As they are inserted, the records are numbered in one of three ways:

1. Standard numbering with standard increment 1.0000 (e.g. 21.0000, 22.0000, 23.0000 ... 99.0000) or
2. Numbering with a preset increment as defined in @PAR INCREMENT or
3. Automatic numbering and renumbering, if the selected increment is too large to permit inclusion of the records to be copied. EDT then selects an increment which is smaller, by a factor of 10, than the standard (case 1) or specified (case 2) increment, and attempts to number the copied records with this increment.
This is repeated until the copied records can be included successfully or until EDT selects the minimum increment of 0.01.

Renumbering if @PAR RENUMBER=ON is specified:

If the copied records cannot be included with the minimum increment of 0.01, EDT automatically renumbers the lines following the target range with the increment value 0.01.

If EDT cannot find sufficient space, no records are inserted into the work file and an error message is issued. When copying into an empty work file, EDT calculates the line number for the first line by adding the standard increment or the specified increment (@PAR INCREMENT) to an initial line number of 0.

If @PAR INCREMENT is entered with an increment < 0.01, it should be noted that the line numbers of lines which have been read in, copied or inserted are not shown fully in F mode (6-digit line number display).

If these incomplete line numbers are then used in @COPY statements, unpredictable results may be produced.

If a line is created with a line number greater than the previously highest line number, the current line number is changed.

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of the file or library element is taken into account in a @COPY statement.

The @COPY statement is only executed if the CCSN of the file (library element) is the same as the CCSN currently selected in EDT or all work files are empty and the coded character set can be displayed on the data display terminal.

Example

COPY L = MACLIB (E=XYZ,M) AFTER 12.3

Element XYZ of macro library MACLIB is copied completely into the current work file after line 0012.3000.

COPY E = PERSONNEL (@), D

The element PERSONNEL with the highest version number (see the “LMS” manual [14]) is copied from a program library previously assigned by means of PAR L=libname into the current work file. It is placed at the end of this file. PERSONNEL is a library element of type D.

@CREATE Create text lines

@CREATE is used to write a freely selectable character string into any line or string variable.

The character string can be

- included in the statement (format 1) or
- entered from the screen (format 2).

@ CREATE (format 1) Create lines

This format of the statement writes the specified character string into a line or a string variable. If appropriate, the existing contents of the line or of the string variable are overwritten. Unlike @SET, format 6 the @CREATE statement does not change the current line number, even if it is used to create a line with a higher number.

Operation	Operands	F mode / L mode
@CREATE	line [:] [string[,...]	

- line The number of the line into which the string is to be written.
 A string variable or a line number variable may also be specified.
- :
- This delimiter needs to be specified only if "ln" cannot be clearly distinguished from "string".
- string[,...]
- The string to be written into the line or string variable.
 This may be specified either
- explicitly, enclosed in single quotes, or
 - implicitly, in the form of a line number, a line number variable or a string variable (in each case with a domain if desired).
- Any combination of the above possibilities may be entered.
- If "string" is not specified, a line consisting of one blank is created.
 If several strings are specified, they are chained together in the order in which they are specified.

@CREATE cannot process tab characters.

Example

```

1.00 THIS IS THE FIRST LINE.....
2.00 THIS IS THE SECOND LINE.....
3.00.....

create 3 'create line 3 using @create'.....0001.00:001(0)

```

Line 3 is created by means of the @CREATE statement.

```

1.00 THIS IS THE FIRST LINE.....
2.00 THIS IS THE SECOND LINE.....
3.00 CREATE LINE 3 USING @CREATE.....
4.00 .....

create 3:3, ' and make it longer'.....0001.00:001(0)

```

Line 3 is created again with the old contents of line 3, chained to the new string 'AND MAKE IT LONGER'.

```

1.00 THIS IS THE FIRST LINE.....
2.00 THIS IS THE SECOND LINE.....
3.00 CREATE LINE 3 USING @CREATE AND MAKE IT LONGER.....
4.00 .....

create 4:1, ' chained with ',,2,1; edit long on.....0001.00:001(0)

```

Line 4 is created by chaining line 1, the string ' CHAINED WITH ', line 2 and line 1, in this order.

In order to display the whole of line 4 in the data window, EDIT LONG ON is entered.

```

THIS IS THE FIRST LINE.....
THIS IS THE SECOND LINE.....
CREATE LINE 3 USING @CREATE AND MAKE IT LONGER.....
THIS IS THE FIRST LINE CHAINED WITH THIS IS THE SECOND LINETHIS IS THE FIRST LI
NE.....

create 4:1:1-12:,'fourth',2:19-23: ; index on .....0001.00:001(0)

```

Line 4 is created again; it contains columns 1 to 12 of line 1, the word 'FOURTH' and columns 19 to 23 of line 2, chained together in this order.

After this, the standard format of the work window is activated.

```

1.00 THIS IS THE FIRST LINE.....
2.00 THIS IS THE SECOND LINE.....
3.00 CREATE LINE 3 USING @CREATE AND MAKE IT LONGER.....
4.00 THIS IS THE FOURTH LINE.....
5.00 .....

```

@CREATE (format 2) Read character strings

This format of @CREATE transfers a character string from the screen to a line or a string variable.

This statement is intended solely for EDT procedures (@DO and @INPUT procedures).

Operation	Operands	L mode / @PROC
@CREATE	line READ [string [,...]]	

- line The number of the line to which the string from the screen is to be transferred. A character string variable or a line number variable may also be specified.
- string[,...] The string to be written into the line or string variable.
This may be specified either
- explicitly, enclosed in single quotes, or
 - implicitly, in the form of a line number, a line number variable or a string variable (in each case with a domain if desired).

When @CREATE is executed, a message is displayed, requesting the user to enter a character string. If the “string” operand was specified, this is displayed as the request for input; otherwise, an asterisk (*) is displayed.

The character string entered by the user is placed in the specified line or string variable.

Tab characters are not processed by this statement.



- If the “string” operand is specified, format 2 of the @CREATE statement calls the Executive macro WRTRD.
- If “string” is omitted or if @CREATE-READ is issued as part of a batch task, the RDATA macro is used.

Example 1

```

6.      @PRINT
1.0000 HELLO
2.0000 JUST WATCH
3.0000 HOW THIS WORKS
4.0000 LINE
5.0000 IS TO BE
6.      SET #S1 = ' DISPLAYED *** '
6.      @PROC 1

```

```

1.      @ @CREATE #S2 READ '*** WHICH ',4,5,#S1 ----- (01)
2.      @ @SET #L2 = SUBSTR #S2 ----- (02)
3.      @ @PRINT #L2
4.      @END
6.      @DO 1
*** WHICH LINE IS TO BE DISPLAYED *** 2 ----- (03)
2.0000 JUST WATCH
6.

```

- (01) @CREATE-READ is to create the string variable #S2. First, it displays the text, which consists of *** WHICH, the contents of lines 4 and 5 and the string variable #S1.
- (02) The line number variable #L2 is created from the contents of string variable #S2.
- (03) The user replies to the question on the screen.

Example 2

```

1.      @ @CREATE #S0 READ '*** WHICH PROCEDURE FILE IS TO ----- (01)
DISPLAY THE TIME ? ***'
2.      @ @SET #I0 = SUBSTR #S0
3.      @ @PROC #I0
4.      @ @@SET #S1 = TIME
5.      @ @@CREATE #S2: '*** IT IS NOW ',#S1,' ***' ----- (02)
6.      @ @@PRINT #S2 N
7.      @ @END
8.      @ @DO #I0
9.      @SAVE 'TEST.CREATE-READ'
9.      @INPUT 'TEST.CREATE-READ'
*** WHICH PROCEDURE FILE IS TO DISPLAY THE TIME ? *** 1 ----- (03)
*** IT IS NOW 110939 ***
9.

```

- (01) The text @CREATE #S0.... is entered. This text is interpreted as a statement only in step (03).
- (02) This creates the procedure line which is to record the current time in #S1.
- (03) By means of @INPUT, the statements saved in TEST.CREATE-READ are executed. This shows how @CREATE..READ can be used outside work files.

@DELETE Delete work files, library elements and record marks

@DELETE has three formats, which provide the following deletion facilities:

- delete all or part of a work file (format 1)
- delete a program library element or a POSIX file (format 2)
- delete record marks (format 3).

If an ISAM file has been opened by means of @OPEN, all or part of this file can be deleted on the disk (format 1). Its catalog entry is not deleted.

@DELETE (format 1) Delete a work file

This format can be used in a work file to delete:

- the entire work file
- single lines and/or line ranges in the file
- domains (column ranges).

Operation	Operands	F mode / L mode
@DELETE	[rng [:domain]] [,...]	

If D is entered with no operands in the F mode statement line, it is rejected with an error message. This is done to prevent the user from inadvertently deleting the entire work file by entering the short form of the statement.

rng The line range in the work file, specified as:

- a single line (e.g. 6) or
- several consecutive lines (e.g. 8-20).

A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.

If rng is omitted, the entire work file is deleted.

domain A domain, specified as:

- a single column number (e.g. 10-10) or
- a range of consecutive column numbers (e.g. 15-25).

If only one column number is specified, the remainder of the line after this column is deleted.

If the first column number is greater than the line length, the line is not changed.

The second column number

- must not be less than the first column number
- may be greater than the actual line length.

If no domain is specified, the entire line is deleted.

If the work file is not empty, @DELETE also deletes the local entry for the file name (see @FILE, @GET, @READ).

If a work file is completely deleted, both the current line number and the increment are set to 1, and the entries in EDT's three-level stack are deleted.

Any save file which may have been created will also be deleted.

Example

```

1.00 111.....
1.50 111.....
1.60 111.....
1.70 111.....
1.80 111.....
1.90 111.....
2.00 111.....
2.10 111.....
3.00      333.....
4.00      444.....
4.10      333.....
4.20      444.....
4.30      333.....
4.40      444.....
4.50      333.....
4.60      444.....
4.70      333.....
4.80      444.....
4.90      333.....
5.00      444.....
6.00 123456789012.....
7.00 .....
8.00 .....
delete 1-2 .....0001.00:001(0)

```

The range from line number 1 to line number 2 is deleted from the work file.

```

2.10 111.....
3.00      333.....
4.00      444.....
4.10      333.....
4.20      444.....
4.30      333.....
4.40      444.....
4.50      333.....
4.60      444.....
4.70      333.....
4.80      444.....
4.90      333.....
5.00      444.....
6.00 123456789012.....
7.00 .....
8.00 .....
9.00 .....
10.00 .....
11.00 .....
12.00 .....
13.00 .....
14.00 .....
15.00 .....
delete & : 7-10.....0002.10:001(0)

```

Columns 7 to 10 (inclusive) are to be deleted from each record in the work file.

```

2.10 111.....
3.00 .....
4.00      44.....
4.10 .....
4.20      44.....
4.30 .....
4.40      44.....
4.50 .....
4.60      44.....
4.70 .....
4.80      44.....
4.90 .....
5.00      44.....
6.00 12345612.....
7.00 .....
.....0002.10:001(0)

```

@DELETE (format 2) Delete library elements

Delete an element from a program library or a file.

Operation	Operands	F mode / L mode
@DELETE	$\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)][,elemtyp])} \\ \text{FILE=path2} \end{array} \right\}$	

path1 The library name. path1 may also be specified by means of a string variable.

elemname The element name. elemname may also be specified by means of a string variable.

vers The version number of the desired element (see the “LMS” manual [14]). If vers is not specified or if *STD is specified, the highest available version of the element is selected.

elemtyp The element type. elemtype may also be specified by means of a string variable.
 Permissible type entries: S, M, P, J, D, X, R, C, H, L, U, F, *STD or user-defined type names with appropriate base type. If no type is specified, the value preset in @PAR ELEMENT-TYPE will be used.

Users who specify a user-defined type name are responsible for ensuring that its associated base type corresponds to one of the permissible types S, M, P, J, D, X, R, C, H, L, U or F.

Type	Contents
S	Source programs
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format
R	Object modules
C	Load modules
H	Created by ASSEMBH
L	Created by BINDER
U	Created by IFG
F	Created by IFG

*STD Default value

Type S is the default value when EDT is started. Any other valid type specification can be defined as the default value by means of @PAR ELEMENT-TYPE.

path2 Name of the BS2000 file (fully qualified file name) to be deleted.
path2 may also be specified by means of a string variable.

Example

```
DELETE LIBRARY = PROGLIB (ELEMENT = TESTOLD (2))
```

Version 2 of library element TESTOLD (type S) is to be deleted from library PROGLIB.

@DELETE (format 3) Delete record marks

This format of the @DELETE statement deletes record marks (see [section "Description of the record marks in F mode" on page 136](#)).

Operation	Operands	F mode / L mode
@DELETE	MARK [m, [...]]	

MARK Record marks in the current work file are to be deleted.

m The number(s) of the record mark(s) to be deleted, where

$$1 \leq m \leq 9$$

Record marks may also be specified via integer variables.

If m is omitted, all record marks (1 to 9) in the current work file are deleted.

Record marks with special functions (record marks 13, 14, 15) are not deleted (e.g. record mark 15 for write protection, which can be set as a subroutine in EDT).

@DELIMIT Define text delimiter characters

This statement permits the user to define a set of characters which are to act as delimiters when searching for a character string with the aid of @ON (see @ON).

Operation	Operands	F mode / L mode
@DELIMIT	= $\left. \begin{array}{l} \text{R} \\ \text{str1} \\ + - \text{str2} \end{array} \right\}$	

The “=” character must be specified in each case, since otherwise D will be interpreted as @DELETE.

- R Resets the text delimiter set back to the default set defined in EDT, namely the blank (X'40') and the characters +.!*());-/,?:'=" .
- str1 The new set of text delimiters.
- str2 A set of delimiter characters which is to be added to (+) or deleted from (-) the existing delimiter set.

If no operand is specified, the text delimiter set will be empty, which means that a subsequent search for delimiters using @ON will only find a hit if the record is absolutely identical with the string that is being sought.

@DIALOG Switch to F mode screen dialog

If RDATA input is being used (BS2000 procedures) or if EDT is called as a subroutine (see the description of the CMD function in the manual “EDT Subroutine Interfaces” [1]), @DIALOG switches to F mode screen dialog.

The screen dialog is terminated by means of @END, @HALT, @RETURN or **K1**, and the processing sequence interrupted by @DIALOG is resumed.

Operation	Operands	F mode / L mode
@DIALOG		

The @DIALOG statement is

- ignored in F mode or in batch operations
- rejected with an error message in the case of L mode in EDT procedures (@DO) or in an INPUT file (@INPUT) or with WRTRD input.

If the screen dialog is called from a BS2000 procedure, the statements @SYSTEM without operands and @EDIT ONLY are disabled.

The user can switch to the operating system only by means of **K2**.



After termination of the screen dialog, all current values which are still needed (work file, library) should be set again, since the user of the F mode dialog may have changed them.

Example

BS2000 procedure PROC.DIALOG

```

/BEGIN-PROCEDURE LOGGING=A,PARAMETERS=YES(-
/  PROCEDURE-PARAMETERS=(&FILE1=,&FILE2=),-
/  ESCAPE-CHARACTER='&')
/ASSIGN-SYSDTA TO-FILE=*SYSCMD
/MODIFY-JOB-SWITCHES ON=5----- (01)
/START-PROGRAM $EDT
@PROC 1 ----- (02)
@READ '&FILE1' ----- (03)
@PAR LOWER=ON,SCALE=ON ----- (04)
@DIALOG ----- (05)
@PROC 1 ----- (06)
@WRITE '&FILE2' ----- (07)
@HALT ----- (08)

```

```
/MODIFY-JOB-SWITCHES OFF=5  
/ASSIGN-SYSDTA TO-FILE=*PRIMARY  
/END-PROCEDURE
```

- (01) Task switch 5 is set before EDT is loaded, in order to select L mode. EDT reads the input from SYSDTA with the aid of RDATA.
- (02) EDT switches to work file 1.
- (03) A file, whose name is requested during execution of the procedure, is to be read.
- (04) Lowercase letters and the scale display are activated.
- (05) EDT switches to F mode screen dialog and displays the work window on the screen. All F mode and L mode statements may be entered during the dialog. The F mode screen dialog is terminated by means of @END, @HALT or @RETURN or by hitting **[K1]**, and the processing sequence which was interrupted by @DIALOG is resumed.
- (06) Work file 1 is again selected as the current work file. This is necessary because the user may have selected another work file during the F mode screen dialog.
- (07) The contents of work file 1 are written back into a SAM file, whose name is requested during execution of the procedure.
- (08) EDT is terminated.

```
/call-procedure name=proc.dialog  
%/PROCEDURE-A,(&FILE1=,&FILE2=),SUBDTA=&  
%/ASSIGN-SYSDTA TO-FILE=*SYSCMD  
%/MODIFY-JOB-SWITCHES ON=5  
%/START-PROGRAM $EDT  
% BLS0500 PROGRAM 'EDT', VERSION '16.5A' OF 'yy-mm-dd' LOADED.  
PROGRAM EDT/16.5A00 STARTED  
%PROC 1  
%@READ '&FILE1'  
%&FILE1=bsp.dialog
```

The procedure "PROC.DIALOG" is started and requests the name of the file to be read. EDT then switches to F mode screen dialog.

```

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----
1.00 If RDATA input is being used (BS2000 procedures) or if.....
2.00 EDT is called as a subroutine (see SMD function), @DIALOG.....
3.00 switches to F mode screen dialog.....
4.00 The work window is displayed on the screen. All F mode and.....
5.00 L mode statements may be entered during the dialog.....
6.00 .....

halt.....0001.00:001(1)

```

As specified in step 04, lowercase letters and the scale display are activated. The F mode screen dialog is terminated by means of @HALT and the procedure interrupted by @DIALOG is resumed.

Later in the procedure, the name of the file into which the work file is to be written is requested. Depending on the actions in the F mode screen dialog, other messages may also be displayed.

```

%@WRITE 'bsp.dialog'
%@HALT
%&FILE1=bsp.dialog1
% EDT8000 EDT NORMAL END
%/MODIFY-JOB-SWITCHES OFF=5
%/ASSIGN-SYSDTA TO-FILE=*PRIMARY
%/END-PROCEDURE
/

```

@DO Start EDT procedure

@DO can be used

- to start EDT procedures, i.e. to process the contents of a work file (1 - 22) line by line (format 1). The lines may contain text or EDT statements.
- to control which lines of a procedure are to be displayed on the screen before they are executed (format 2).

@DO (format 1) Start EDT procedures

@DO, format 1, starts an EDT procedure, i.e. causes the text lines and EDT statements stored in the specified work file (1 - 22) to be executed sequentially.

Operation	Operands	F mode / L mode / @PROC
@DO	procno [,] [(param [,...])] [spec] [=ln1,ln2 [, [-] ln3]] [PRINT]	

procno The number of the work file (1-22) whose contents are to be used by EDT as input. procno must be specified either as an integer ($1 \leq \text{procno} \leq 22$) or as an integer variable which contains the number of the work file to be executed.

param The parameters to be passed to the procedure which is to be executed. These parameters must be defined within the procedure by means of @PARAMS (see @PARAMS). They are separated from each other by a comma. If @PAR LOWER=ON is activated, lowercase letters can also be passed (no conversion into uppercase letters).

The positional parameters must be specified before the keyword parameters and entered precisely in the order in which they were defined in the @PARAMS statement. Keyword parameters may be entered in any order.

The number of parameters is limited only by the maximum length of an EDT statement, namely 256 characters.

spec Loop symbol. This can be used as an operand in EDT statements in the procedure wherever a line number is required. When the procedure is executed, EDT then uses the current value of this loop symbol.

The loop symbol must be a special character, otherwise the @DO statement is rejected with the error message: % EDT3952 INVALID SYMBOL

In order to avoid errors and unpredictable results, the following characters should not be used as the loop symbol:

% \$? * (: # + - . < = > ' ;

If the procedure is started in F mode, the semicolon (;) must not be used as the loop symbol.

Suitable loop symbols are:

! " { } [] | /

If the loop symbol is not specified, it is regarded as undefined. If the operand sequence ln1,ln2,[-]ln3 is not specified, the loop symbol has the value 1.

=ln1,ln2,[-]ln3

These operands control multiple execution of a procedure (see example 3).

Before the procedure is executed for the first time, the loop symbol is set by EDT to the starting value ln1. After each pass, EDT increments or decrements (minus sign before ln3) the value of the loop symbol by ln3. The default value for ln3 is 1. The loop is executed until the value of the loop symbol becomes greater than or less than ln2, in which case it is terminated.

The procedure is executed at least once, since the condition is checked after each pass (REPEAT UNTIL).

Line number symbols (e.g. %,\$) may also be specified for ln1, ln2 and ln3. EDT then uses the value which this symbol has when @DO is executed. The number of passes is thus not affected by changing the value of this symbol during execution of the procedure.

The default value for ln1,ln2 and ln3 is 1.

PRINT

Each line of the procedure is output before it is executed.

Specifying PRINT causes all error messages to be output and the EDT error switch to be set. Normally, error messages which do not affect execution of the procedure (such as % EDT0901 NO MATCH IN RANGE or % EDT4932 LINE NUMBER NOT FOUND) are not output and the EDT error switch is not set by such messages.

Rules for specifying EDT parameters

1. The value of a parameter is determined by all characters, including blanks, specified between the commas.
2. If a parameter value contains a single quote or closing parenthesis, the value must be enclosed in single quotes. Single quotes within the parameter value must be entered twice. If @QUOTE was used to assign the function of the single quote to a different character, it does not apply to the a single quotes enclosing the parameter value.
3. If no value is specified for a parameter, then this parameter contains an empty string.

Specifying no value means that:

- for positional parameters, there is no value specified between the delimiter characters (parenthesis and comma);
- for keyword parameters, the equals sign is followed by a comma or a parenthesis.

Aborting EDT procedures

EDT procedures can be interrupted at any time by means of `[K2]`. Control is passed to the operating system, where the user can

- resume execution of the procedure by means of RESUME-PROGRAM or
- return to EDT by means of SEND-MESSAGE TO=PROGRAM, thus aborting the procedure.

If a @RUN statement or user statement (see @USE) is being processed, the procedure cannot be aborted by means of SEND-MESSAGE TO=PROGRAM.

An invalid statement will not abort the procedure.

Example 1

```

1.      @SET #S0 = 'TEST OF PROCEDURE FILE 1'
1.      @PROC 1 ----- (01)
1.      @ @SET #S1 = #S0:1-4: ----- (02)
2.      @ @CREATE #S2: ' '*4,#S0:5-9:
3.      @ @CREATE #S3: ' '*9,#S0:10-24:
4.      @ @CREATE #S4: ' '*24
5.      @ @PRINT #S1.-#S4
6.      @ @PRINT #S0
7.      @END ----- (03)
1.      @DO 1 ----- (04)
#S01 TEST
#S02      OF
#S03      PROCEDURE FILE 1
#S04
#S00 TEST OF PROCEDURE FILE 1

```

1.

- (01) Switch to work file 1.
- (02) The EDT statements are written into work file 1. When the procedure is called using @DO, these statements will create the string variables #S1 to #S4 and then display them together with #S0.
- (03) @END switches back from work file 1.
- (04) The procedure in work file 1 is called.

Example 2

```

1.      @PROC 2 ----- (01)
1.      @ @PARAMS &STRING ----- (02)
2.      @ @SET #S1 = '+++++'
3.      @ @SET #S2 = &STRING ----- (03)
4.      @ @PRINT #S2
5.      @END
1.      @DO 2(#S1) PRINT ----- (04)
1.      @SET #S1 = '+++++'
1.      @SET #S2 = #S1
1.      @PRINT #S2
#S02 ++++++
1.      @DO 2('#S1') PRINT ----- (05)
1.      @SET #S1 = '+++++'
1.      @SET #S2 = #S1
1.      @PRINT #S2
#S02 ++++++
1.      @DO 2( '#S1' ) PRINT ----- (06)
1.      @SET #S1 = '+++++'
1.      @SET #S2 = '#S1'
1.      @PRINT #S2
#S02 #S1
1.

```

- (01) Switch to work file 2.
- (02) The first line in this file is a @PARAMS statement, which permits the positional parameter &STRING to be used several times within this file.
- (03) #S2 is to receive a value which is unknown when work file 2 is created, and is to be requested when the procedure is executed by means of @DO 2(...).
- (04) The value #S1 in parentheses causes &STRING to be replaced by the value #S1 wherever it occurs in the statements in work file 2. PRINT causes the statements to be output before they are executed.
- (05) The value '#S1' is now passed for the positional parameter &STRING. Since the first and last characters of this parameter value are single quotes, they are suppressed when the parameter value is replaced in work file 2. Due to the PRINT operand, this can be seen in the output. This, therefore, has the same effect as step (04).
- (06) The only difference from (05) is that the parameter value now has a leading blank and a trailing blank, but this is sufficient to cause the single quotes to be included as part of the parameter value.

Example 3

```

1.      *
2.      @PROC 3 ----- (01)
1.      @ @CREATE $+1: $,'*' ----- (02)
2.      @END ----- (03)
2.      @DO 3,! =1,15 ----- (04)
2.      @PRINT
1.0000 *
2.0000 **
3.0000 ***
4.0000 ****
5.0000 *****
6.0000 *******
7.0000 ********
8.0000 **********
9.0000 *********
10.0000 **********
11.0000 *********
12.0000 **********
13.0000 *********
14.0000 **********
15.0000 *********
16.0000 ********** ----- (05)
2.

```

- (01) Switch to work file 3.
- (02) A single EDT statement is written into work file 3.
- (03) Return to work file 0.
- (04) Work file 3 is now executed, using the exclamation mark (!) as the loop symbol. Work file 3 is executed 15 times. It is possible, but not mandatory, to refer to line numbers via the loop symbol; in this example, it is not done. The operand sequence !=1,15 is equivalent to entering @DO 3 15 times without this operand sequence.
- (05) The output shows that 15 new lines have been created.

Example 4

```

5.      @PRINT
1.0000 1111111
2.0000 2222222
3.0000 3333333
4.0000 4444444
5.      @SET #S4 = '-----'
5.      @PROC 4 ----- (01)
1.      @ @PRINT !.-.$ ----- (02)
2.      @ @PRINT #S4 N
3.      @END
5.      @DO 4, !=$, %, -1 ----- (03)
4.0000 4444444
-----
3.0000 3333333
4.0000 4444444
-----
2.0000 2222222
3.0000 3333333
4.0000 4444444
-----
1.0000 1111111
2.0000 2222222
3.0000 3333333
4.0000 4444444
-----
5.

```

- (01) Switch to work file 4.
- (02) A line number is addressed with the aid of the loop symbol !.
- (03) Work file 4 is started several times. In the first pass, the highest existing line number is used for the loop symbol ! and this value is decremented by 1 (operand ln3 is -1) in each subsequent pass until the loop symbol reaches the value of the lowest existing line number (%).

Example 5

```

1.      @PROC 4 ----- (01)
1.      @READ 'PROC-FILE.4' ----- (02)
6.      @PRINT
1.0000 @PARAMS &A
2.0000 ABC
3.0000 EFG
4.0000 @ON 1 CHANGE 'ABC' TO '&A'
5.0000 @5: &A
6.      @END ----- (03)
1.      @DO 4 ('A,'B') ----- (04)
INVALID VALUE
1.      @DO 4 ('A, 'B')
6.      @PRINT
1.0000 A,'B ----- (05)
2.0000 EFG
5.0000 A, 'B
6.

```

- (01) Switch to work file 4.
- (02) Read the SAM file 'PROC-FILE.4' into work file 4.
- (03) Return to work file 0.
- (04) EDT rejects a parameter value containing a single apostrophe.
- (05) During execution of the work file, lines are written into the main file. In line 1, EDT suppresses one of the two consecutive apostrophes; in line 3, it accepts the parameter value without changing it.

@DO (format 2) Activate or deactivate logging

With this format, the PRINT option of @DO, format 1, can be deactivated or activated anywhere within a procedure.

Operation	Operands	@PROC
@DO	N P	

N EDT no longer logs the lines of the procedure before they are executed.

P EDT logs all subsequent lines of the procedure before they are executed.

This statement is used primarily for debugging EDT procedures. A typical example of its use would be to check whether or not a specific part of a procedure is actually executed.

Example

```

1.    @PROC 5 ----- (01)
1.    @ @SET #S5 = 'A'
2.    @ @DO N ----- (02)
3.    @ @CREATE #S6: 'B'*6,#S5
4.    @ @CREATE #S7: #S6,'C',#S6
5.    @ @DO P
6.    @ @PRINT #S5.-#S7 ----- (03)
7.    @ @DELETE #S5.-#S7
8.    @END ----- (04)
1.    @DO 5 PRINT ----- (05)
1.    @SET #S5 = 'A'
1.    @DO N
1.    @PRINT #S5.-#S7
#S05 A
#S06 BBBBBA
#S07 BBBBACBBBBBA
1.    @DELETE #S5.-#S7
1.

```

- (01) Switch to work file 5.
- (02) EDT stops logging the procedure lines before they are executed.
- (03) EDT resumes logging of the procedure lines before execution.
- (04) Switch back to work file 0.
- (05) Start the procedure in work file 5, logging the lines before they are executed.

@DROP Delete work files

@DROP deletes some or all of the work files 1-22 and releases the virtual memory pages they occupied.

@DROP may be used only in work file 0, i.e. it must not be used in EDT procedures.

Operation	Operands	F mode / L mode
@DROP	{ procno [,...] }	

procno The number of the work file (1-22) which is to be deleted. Any number of work files may be specified.

ALL All work files (1-22) are to be deleted and the virtual memory pages they occupy are to be released.



- @DROP removes the local file name.
- Open library elements should be closed beforehand (see @CLOSE).

Example 1

```

1.      @PROC USED ----- (01)
<03>   1.0000 T0    3.0000
<05>   1.0000 T0    1.0000
<08>   1.0000 T0    1.0000
<10>   1.0000 T0    1.0000
<14>   1.0000 T0    1.0000
1.      @DROP 10 ----- (02)
1.      @PROC USED
<03>   1.0000 T0    3.0000
<05>   1.0000 T0    1.0000 ----- (03)
<08>   1.0000 T0    1.0000
<14>   1.0000 T0    1.0000
1.      @DROP 8,5 ----- (04)
1.      @PROC USED
<03>   1.0000 T0    3.0000 ----- (05)
<14>   1.0000 T0    1.0000
1.

```

- (01) Display the numbers of the work files currently in use. In this case, these are the work files 3, 5, 8, 10, 14.
- (02) Delete work file 10 and release its memory space.
- (03) @PROC USED shows that only work files 3, 5, 8, 14 are now still in use.
- (04) @DROP can also be used to delete and release several work files at once; in this example, work files 5 and 8 are deleted and their memory space released.
- (05) Only work files 3 and 14 are now still in use.

Example 2

```

1.      @PROC USED ----- (01)
<03>   1.0000 TO    3.0000
<14>   1.0000 TO    1.0000
1.      @DROP ALL ----- (02)
1.      @PROC USED
% EDT0907 NO PROCEDURE FILES DECLARED ----- (03)
1.
```

- (01) Display the numbers of all work files which are in use.
- (02) Delete all (1-22) work files and release their memory space.
- (03) None of the work files is now still in use.

@EDIT Switch edit mode

This statement is used

- to switch from L mode to F mode and vice versa (@EDIT FULL SCREEN, @EDIT ONLY)
- to suppress the output of the current line number
- to switch between reading from SYSDTA with RDATA and WRTRD (@EDIT ONLY, @EDIT)
- to display the contents of the current line before it is edited (@EDIT .. PRINT)
- to activate sequential mode (see @+, @-), (@EDIT .. SEQUENTIAL)
- to define the maximum line length (i.e. the right-hand margin) on printer terminals (@EDIT .. cl).

Operation	Operands	F mode / L mode
@EDIT	{ FULL SCREEN [ONLY] [PRINT] [SEQUENTIAL] [cl] }	

If no operands are specified, EDT switches to L mode.

FULL SCREEN

causes EDT to switch to F mode.

@EDIT FULL SCREEN is ignored in batch mode and in F mode.

If this statement is encountered within an EDT procedure (@DO) or within an INPUT file (@INPUT file), it is rejected with an error message.

If @EDIT FULL SCREEN is entered within a statement block (@BLOCK mode) in a dialog, any following statements in the block are ignored.

@DIALOG can also be used to switch to F mode (see @DIALOG).

ONLY

If in F mode, EDT switches to L mode. Output of the current line number is suppressed (* is output instead). EDT uses the macro RDATA instead of WRTRD. Whereas WRTRD reads only from the screen, RDATA reads data from the system file SYSDTA.

If @EDIT is entered without ONLY, the current line number is again displayed, and input is again performed via the WRTRD macro.

PRINT Causes the line number and the contents of the line to be displayed before the line number or * is displayed.

SEQUENTIAL

Normally, the current line number is incremented by the current increment value when text is entered in a line or @+ is entered (or decremented if @- is entered).

This may result in existing lines, namely the lines between the old and the new current line number, being skipped without being noticed by the user.

If SEQUENTIAL is specified, the current line number is incremented or decremented as described above only if there are no intervening lines. Otherwise, the number of the first intervening line becomes the current line number.

cl The maximum line length for a printer terminal. The value specified for cl must be at least 50 and must not exceed 256.

EDT takes the default value of 72 for the line length from the system setting (which can be changed via the LINE-LENGTH operand of the MODIFY-TERMINAL-OPTIONS command).



PRINT can be particularly useful in batch mode. The contents of each line are logged before and after it is processed if

- @EDIT PRINT and @LOG ALL are specified,
- the file is positioned to the desired line number by means of a statement before text input and
- line contents are not updated by statements, but only by the input of new line contents.

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of SYSDTA is checked with @EDIT ONLY when a switch is to be made to SYSDTA. Switchover to SYSDTA via @EDIT ONLY is not possible unless the CCSN of SYSDTA matches the current CCSN. If there is no match, an error message is issued in interactive mode and reading continues with WRTRD (@EDIT without ONLY); in batch mode, EDT terminates.

@ELIM Delete ISAM file

This statement deletes all or part of an ISAM file on disk. If the entire file is deleted, the file name remains in the catalog (unlike deleting with @UNSAVE). It is also possible to delete the virtual file and the disk file simultaneously.

The file is open only during execution of @ELIM.

Operation	Operands	F mode / L mode
@ELIM	['file'] [(ver)] range* [BOTH]	

file	<p>The name of the file to be deleted.</p> <p>If this is omitted, the local @FILE entry is used as the file name, if it exists; if not, the global @FILE entry is used (see also @FILE).</p>
ver	<p>The version number of the file.</p> <p>This may consist of up to three digits or an asterisk (*). * designates the current version number. If a number is entered, @ELIM is executed only if this is the current version number; otherwise, the current version number is simply displayed.</p>
range*	<p>A line range, specified as:</p> <ul style="list-style-type: none"> – one or more line numbers, separated by commas (e.g. 4,6,15) – one or more line ranges, separated by commas (e.g. 5-10,17-19) – a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30). <p>The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.</p> <p>The values of the symbolic line numbers do not refer to the file specified by "file", but to the current work file.</p>
BOTH	<p>The specified line range is to be deleted in both the ISAM file and the virtual file.</p>

The most important differences between @ELIM and @UNSAVE are:

- @ELIM does not delete the catalog entry for the file
- @ELIM can be used only for ISAM files.

Example

```

23.00 .....
get 'bsp.elim' noresize .....0001.00:001(0)

```

The ISAM file 'XMPL.ELIM' is read into work file 0, using the ISAM keys as the line numbers.

```

1.00 ONE.....
2.00 TWO.....
3.00 THREE.....
4.00 FOUR.....
5.00 FIVE.....
6.00 SIX.....
7.00 SEVEN.....
8.00 EIGHT.....
9.00.....

elim 'bsp.elim' 5-7 both .....0001.00:001(0)

```

Line range 5-7 is to be deleted from the ISAM file 'XMPL.ELIM' and also from work file 0.

If the ISAM file is read in by means of @GET *without* NORESEQ, the line ranges to be deleted from the ISAM file and from the work file will not necessarily be the same.

```

1.00 ONE.....
2.00 TWO.....
3.00 THREE.....
4.00 FOUR.....
8.00 EIGHT.....
9.00 .....

delete ; get 'bsp.elim' noresize.....0001.00:001(0)

```

Work file 0 is cleared and the ISAM file 'XMPL.ELIM' is read in again.

```

1.00 ONE.....
2.00 TWO.....
3.00 THREE.....
4.00 FOUR.....
8.00 EIGHT.....
9.00 .....

```

This shows that the line range from 5 to 7 has also been deleted from the ISAM file.

@END Terminate processing of current work file

@END terminates processing of the current work file. EDT switches back to the work file in which processing of the current work file was initiated by means of the @PROC statement.

Operation	Operands	F mode / L mode / @PROC
@END	[comment]	

comment A comment of the user's choice.
 A comment may be specified only in L mode.

Response in L mode

If @END is entered in a work file other than work file 0, the program returns to the work file from which processing with @PROC was initiated.

Entering @END in interactive mode in work file 0 causes EDT to display first the message % EDT4939 @END WITHOUT @PROC, and then confirmation queries % EDT0900 and % EDT0904, or if there are no work files to be saved, only the % EDT0904 query.

The EDT0904 query is not suppressed by activating task switch 4 prior to the EDT session.

Response in F mode or in a screen dialog following @DIALOG

If @END is entered in a work file,

- the EDT session is terminated,
- the program returns to a system procedure and
- continues with a subroutine call.

If there are any work files which have not yet been saved, EDT displays the message % EDT0900 EDITED FILE(S) NOT SAVED! , followed by the query % EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO).

If you enter Y, the unsaved virtual files are lost, and EDT is terminated.

The confirmation query is not displayed if F mode was called with @DIALOG.

Example 1

```

1.      @PROC 1 ----- (01)
1.      @ @SET #S1 = DATE
2.      @ @SET #S2 = TIME ----- (02)
3.      @ @PRINT #S1.-#S2 N
4.      @END ----- (03)

```

- (01) Switch to work file 1.
- (02) An EDT procedure is entered in work file 1.
- (03) Return to work file 0. The procedure in work file 1 can now be called by means of @DO 1.

Example 2

```

1.      @PROC 7 ----- (01)
1.      @PROC ----- (02)
<07>
1.      @ @SET #S7 = 'THIS IS PROC 7'
2.      @ @PRINT #S7
3.      @PROC 8 ----- (03)
1.      @ @SET #S8 = 'THIS IS PROC 8'
2.      @PROC USED
<01> 1.0000 TO 3.0000 ----- (04)
<07> 1.0000 TO 2.0000
2.      @END ----- (05)
3.      @PROC ----- (06)
<07>
3.      @END ----- (07)
1.      @PROC ----- (08)
<00>

```

- (01) Switch to work file 7.
- (02) Display the number of the current work file.
- (03) Switch to work file 8.
- (04) When the work files currently in use are displayed, work file 8 is not included in the output, since it has not yet been terminated by means of @END.
- (05) EDT returns to work file 7 (which is where the branch to work file 8 was executed by means of @PROC).
- (06) Displaying the current work file number confirms that EDT is now in work file 7.
- (07) Return to work file 0.
- (08) Displaying the current work file number again shows that EDT has returned to work file 0, i.e. the main file.

@ERAJV Delete job variables

@ERAJV deletes job variable entries from the catalog. If the subsystem “job variable support” is not installed, this statement is rejected with an error message.

Operation	Operands	F mode / L mode
@ERAJV	str [ALL]	

- str** Selects the job variables to be deleted. All specifications are permitted which can be entered in the BS2000 command DELETE-JV as long as a length of 54 is not exceeded.
- The job variable name can also be partially qualified or the job variable can be addressed via its link name. '*str*' can also be specified. EDT then selects the names itself according to the wildcard syntax (analogous to the BS2000 command SHOW-FILE-ATTRIBUTES).
- Otherwise, the operand is not checked by EDT, i.e. it is transferred to the system unchanged.
- If more than one job variable name fulfills the condition and the ALL parameter has not been specified, EDT issues an additional query in interactive mode asking how processing is to continue. In batch mode, the statement is not executed in this case.
- ALL** If ALL is specified, all job variables to which the name applies are removed from the catalog without a request for confirmation.
- If no job variable with the appropriate name is found or if a DELETE-JV command is rejected by the system, EDT reports an error and sets the EDT switch for DMS errors. DMS errors can be queried in EDT procedures using @IF, format 1.

@EXEC Start program

The @EXEC statement

- terminates the EDT session and
- loads and starts the specified program.

@EXEC is one of the EDT statements that is relevant to security (see [section "Data protection" on page 71](#)). In uninterruptible system procedures in interactive mode and in the case of input from a file, the statement will be rejected (unless it is read from SYSDTA=SYSCMD).

Operation	Operands	F mode / L mode / @PROC
@EXEC	string	

string A character string specifying the name of the program to be loaded and started.
 The string may be specified:

- explicitly, enclosed in single quotes, or
- implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

If there are still work files which have not been saved, the numbers of these files are displayed after the message:

```
% EDT0900 EDITED FILE(S) NOT SAVED!
```

This is accompanied by one of the following items, if available:

- a local @FILE entry
 - defined explicitly by @FILE LOCAL, or
 - defined implicitly by @READ, @GET, @OPEN (format 1),
- the library and element name of
 - a library element opened by means of @OPEN (format 2)
- or the file name of
 - a SAM or ISAM file opened with @OPEN (format 2) or
 - a POSIX file opened with @XOPEN.

The user then receives the following query:

```
% EDT0904 TERMINATE EDT? REPLY (Y=YES, N=NO)
```


- N: In F mode the work window is displayed again. The user can close any files with unsaved data and write them back.
- Y: Virtual files with unsaved data are lost. EDT is terminated and the specified program started.

If a file was opened for real processing via @OPEN, this query is suppressed. EDT closes the file by means of an implicit @CLOSE.

The save query can be suppressed by setting task switch 4 before EDT is called.

Example

```

1.00 The @EXEC statement.....
2.00 - terminates the EDT session and.....
3.00 - loads and starts the specified program.....
4.00 .....

exec '$lms'.....0001.00:001(0)

```

EDT is to be terminated and LMS is to be loaded and started.

```

% EDT0900 EDITED FILE(S) NOT SAVED!
LOCAL FILE ( 0 ) :
% EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)?y
% BLS0500 PROGRAM 'LMS', VERSION 'V3.0A' OF 'yy-mm-dd' LOADED.
LMS0310 LMS VERSION V03.0A00 LOADED
CTL=(CMD) PRT=(OUT)
$

```

Since the work file has not been saved, EDT asks (as for @HALT) whether it is really to be terminated. This is done, and LMS loaded and started, only if the user replies to the message with Y.

@FILE Preset file name

@FILE is used to preset a file name for @GET, @READ, @WRITE, @SAVE, @OPEN and @ELIM.

There is

- a local @FILE entry, which is specific to one work file,
- a global @FILE entry, which is valid for all work files.

If no file name is specified in a @GET, @READ, @WRITE, @SAVE or @ELIM statement, the local @FILE entry is used, if one exists; otherwise, the global @FILE entry is used as the file name.

Only the global @FILE entry is evaluated for the @OPEN statement (format 1).

Operation	Operands	F mode / L mode
@FILE	[string [(ver)]] [LOCAL]	

string A character string specifying the file name.

The string may be specified:

- explicitly, enclosed in single quotes, or
- implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

ver The version number of the file.

This may consist of up to three digits or an asterisk (*). * designates the current version number. If LOCAL is specified, no version number may be specified.

LOCAL The specified file name is registered as the file name specific to the current work file (explicit local entry). If no explicit local entry exists when @READ 'file' or @GET 'file' is executed, the file name specified in this statement becomes the local file name (implicit local entry).

If LOCAL is not specified, the specified file name becomes the global @FILE entry.

The local @FILE entry is deleted by

- entering @FILE LOCAL without "string"
- completely deleting the work file by means of @DELETE, format 1
- using @CLOSE to close a file which was opened by means of @OPEN.

The global @FILE entry is deleted by entering @FILE without any operands.

Example

```

23.00 .....
file 'xmpl.file' (*) ; get

```

For the following @GET and @SAVE statements, the file name 'XMPL.FILE', including the asterisk as the version number, is preset as the file name.

The file 'XMPL.FILE' is then read by means of @GET.

```

1.00 ONE.....
2.00 TWO.....
3.00 THREE.....
4.00 FOUR.....
5.00 FIVE.....
6.00 .....

```

```

% EDT0902 FILE 'XMPL.FILE' VERSION 002
delete 1-2 ; save .....0001.00:001(0)

```

Lines 1 and 2 are deleted from the work file and the contents of the work file are then written back into the file 'XMPL.FILE' by means of @SAVE.

```

3.00 THREE.....
4.00 FOUR.....
5.00 FIVE.....
6.00 .....

```

```

% EDT0903 FILE 'XMPL.FILE' IS IN THE CATALOG, FCBTYPE = ISAM
y EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO) .....0001.00:001(0)

```

```

3.00 THREE.....
4.00 FOUR.....
5.00 EIGHT.....
6.00 .....

```

```

% EDT0902 FILE 'XMPL.FILE' VERSION 003
.....0003.00:001(0)

```

@FSTAT Display catalog information

With the aid of @FSTAT, the user can determine which files exist under a specific user ID and query the attributes of these files.

The information can be

- displayed on the screen or
- written into a work file.

The list is sorted alphabetically.

Operation	Operands	F mode / L mode
@FSTAT	$\left\{ \begin{array}{l} \text{['pfile']} \\ \text{str-var} \end{array} \right\} [\text{[TO] In [(inc)] }] \left[\left\{ \begin{array}{l} \text{SHORT} \\ \text{LONG [ISO4]} \end{array} \right\} \right]$	

The keyword TO can be omitted only if 'pfile' is specified.

pfile Selects the files to be displayed. "pfile" is equivalent to the operand "pathname" in the BS2000 command SHOW-FILE-ATTRIBUTES. EDT does not check this operand, but passes it to the operating system without modification. Here it is possible to specify any or all of the entries that can be specified for the FILE-NAME operand of the system command.

If no file which matches the path specification is found, EDT reports an error. In EDT procedures, the EDT error switch can be interrogated by means of @IF, format 1.

If a CATID is specified in the "file" operand, the list of file names is output with the catalog and user IDs (see SHORT).

lstr-var Selection of the files to be output can also be specified via a string variable (#S1-#S20).

In The line number at which the file names (catalog information) are to be written into the current work file. The minimum value is 0.0001, the maximum 9999.9999. In may also be specified by means of a line number variable (#L0 to #L20) or symbolically (e.g. %,\$).

If In is not specified, the information

- is displayed on the screen in L mode
- is output to SYSOUT in batch mode
- is written into work file 9, which is cleared beforehand, in F mode.

- inc** The increment value to be used for generating subsequent line numbers. If inc is omitted, the implicit increment value is used.
- SHORT** Only a list of file names with the catalog and user IDs is output.
If the specified path name is fully qualified, the file name is output in the form in which it was entered.
- LONG** In addition to the file names, further catalog information is output.
If In was not specified and @PAR INFORMATION=ON is activated, a header line describing the catalog information is displayed in F mode.

Column	Header	Meaning
1-7	SIZE	Number of PAM pages
8	P	File on private or public volume
9-62	FILENAME	File name with catalog ID and user ID
63-69	LAST PP	Last page
71-78	CR-DATE	Date created (YY-MM-DD)
80	S	SHARE attribute (Y/N)
81	A	ACCESS attribute (W/R)
83-86	FCB	FCB type (SAM/ISAM/PAM/BTAM)
88	R	READ-PASS attribute (Y/N)
89	W	WRITE-PASS attribute (Y/N)

ISO4 The creation date (CR-DATE) is specified in the form YYYY-MM-DD.

If neither **SHORT** nor **LONG** is specified in @FSTAT, the catalog information is output analogously to the input form. For example: the input @FSTAT '\$USERID.file' results in output of the file name with the user ID. If the file name is specified in partially qualified form, @FSTAT outputs a list of file names without catalog ID and user ID (for reasons of compatibility).

If **LONG** is specified in F mode, the length of the output line for each file is 89 characters, thus exceeding the maximum work window width of 80 characters (if @PAR INDEX=OFF applies).



If In is specified and a line is created with a line number greater than the previously highest line number, the current line number is changed.

Example

```

23.00 .....
fstat '*xmpl*' long ; edit long

```

This requests full information about all files whose names include the string 'XMPL'. EDIT LONG is specified so that all the information can be seen in the data window.

```

0000003 :N:$USER.XMPL.FSTAT                0000003 94-01-12 N
W ISAM NN.....
0000021 :N:$USER.XMPL.FSTAT.1              0000021 89-01-11 N
W SAM NN.....
0000015 :N:$USER.XMPL.FSTAT.XMPL          0000015 94-01-10 N
W SAM NN.....
0000006 :N:$USER.PROG.XMPL                0000006 94-01-12 Y
R PAM NY.....

.....0001.00:001(9)

```

The information is placed in work file 9.

@GET Read ISAM file

@GET reads or copies all or part of an ISAM file from disk or tape into the current work file.

The file is physically open only during execution of @GET. Processing is executed on the internal copy of the original ISAM file.

By default, EDT assumes that the ISAM file contains variable-length records (for information on reading a file with fixed-length records, see [section “Processing ISAM files with nonstandard attributes” on page 50ff](#)).

Operation	Operands	F mode / L mode
@GET	['file'] [(ver)] [range*] [:col:] [NORESEQ]	

If the ISAM key is to be interpreted as a line number, it is imperative that NORESEQ be specified.

file The name of the desired file.
 If there is no local @FILE entry for this file name, then the specified file name is stored as this entry. If the operand “file” is omitted, the local @FILE entry is used as the file name, if one exists; otherwise, the global @FILE entry is used (see @FILE). If the specified ISAM file does not exist, @GET is rejected with an error message.

If the file link name EDTISAM has been assigned to a file, it is sufficient to enter '/' in order to read this file (see [section “File processing” on page 49ff](#)).

ver The version number of the file.
 This may consist of up to three digits or an asterisk (*). * designates the current version number. If * is specified, the current version number is displayed before the read operation. If an incorrect version number is specified, the correct version number is displayed and this file is then read.

range* A line range, specified as:

- one or more line numbers, separated by commas (e.g. 4,6,15)
- one or more line ranges, separated by commas (e.g. 5-10,17-19)
- a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).

The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %, \$) or via line number variables. String variables must not be used.

If range* is omitted, all lines of the file are read.

- col A column range, specified as:
- one or more columns, separated by commas (e.g. 10,15,8)
 - one or more column ranges, separated by commas (e.g. 15-25,18-23)
 - a combination of columns and column ranges (e.g. 10,14-29,23-50,17).
- If no column range is specified, the entire contents of each line are read.
- Repetitions and overlapping columns and column ranges are permitted.
- NORESEQ The line numbers are formed from the ISAM keys of the ISAM file being read. This may result in existing lines being overwritten.
- The line numbers in the F mode work window are only 6 digits long. For this reason, only the first 6 positions of the ISAM keys are displayed.

After processing, the file can be written back to disk or tape by means of @SAVE.



If the specified file is a SAM file, a @READ statement is executed internally for this file. This is indicated by a message. range* and NORESEQ are ignored.

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of the file is taken into account in a @GET statement for BS2000/OSD V1.0 and higher.

The @GET statement is only executed if the CCSN of the file is the same as the CCSN currently selected in EDT or all work files are empty and the coded character set can be displayed on the data display terminal.

@GETJV Read value of job variable

Using @GETJV, the value of a job variable can be:

- displayed on the screen
- written to a work file
- assigned to a character string variable.

If the subsystem “job variable support” is not installed, the statement is rejected with the error message % EDT5259 JVS NOT IN SYSTEM.

Operation	Operands	F mode / L mode
@GETJV	[string] [=line]	

string Character string specifying a fully qualified job variable name.

“string” can be specified as follows:

- explicitly as a character string in single quotes
- implicitly via a line number, a line number variable or a string variable, always with the appropriate column range.

If “string” is not specified, the job variable with the link name *EDTLINK is addressed.

line Number of the line to which the value of the job variable is to be written. “line” can also be specified via line number variables (#L0-#L20), via symbolic line numbers (e.g. %,\$), or as a string variable (#S0-#S20). If “line” is omitted, the value of the job variable is displayed on the screen.

If line is specified and a line is created with a line number greater than the previously highest line number, the current line number is changed.

Example

```
1.      @GETJV '$SYSJV.DATE' = 1 ----- (01)
1.      @GETJV '$SYSJV.DATE' = #S01 ----- (02)
1.      @CREATE 2 : 'TODAY IS THE '
1.      @CREATE 2 : 2,#S01:7-8:,'.',#S01:4-5:,'.',#S01:1-2: ----- (03)
1.      @PRINT
1.0000 90-09-24267
2.0000 TODAY IS THE 24.09.90
```

- (01) The current date is written in line 1.
- (02) The current date is stored in the string variable #S01.
- (03) Line 2 is created by means of @CREATE.

@GETLIST Read elements of list variable

@GETLIST is used to read some or all of the elements of a list variable into the current work file.

In systems where the SDF-P subsystem is not installed, @GETLIST is rejected and an error message is issued.

Operation	Operands	F mode / L mode
@GETLIST	string [range*] [:col:]	

string A character string specifying the name of the list variable.
string may be entered

- explicitly, i.e. as a character string enclosed in single quotes, or
- implicitly, i.e. by means of a line number, a line-number variable or a string variable (each can be specified with a column range).

The elements of the list variable must be of the STRING type; if they are not, the statement will be aborted and an error message issued.

range* A range of lines consisting of:

- one or more line numbers separated in each case by a comma (e.g. 4,6,15)
- one or more line ranges separated in each case by a comma (e.g. 5-10,17-19)
- a combination of individual lines and line ranges (e.g. 4,7-23,8,15-30)

A range of lines can also be specified by means of the current line-range symbol (see @RANGE), symbolic line numbers (e.g. %,\$) or line-number variables.
It is not permissible to specify string variables.

range* gives users the possibility of reading in only some of the elements of a list variable. In this case, a one-to-one correspondence between element numbers and line numbers is to be imagined, in which 0.0001 stands for the 1st element in the list, 0.0002 stands for the 2nd, and so forth. The elements which are read in are appended to the contents of the work file or to file opened with @OPEN. There, the elements are given the line number yielded by the current line number and the current increment.
range* refers to the element numbers. For example, entering 0.0001-0.0005 reads in the first 5 elements of the list.

If no entry is specified for range*, all of the elements are read in.

col columns or column ranges
 If no column range is specified, the full length of the element is read in.
 If the content of the element exceeds 256 characters in length, only the first
 256 characters are read in, and a warning is issued.
 Column and column-range entries may overlap and/or occur multiple times.
 If the list element is empty or comes to an end before the first column
 specified, no line is created for it in the work file.

If the maximum line number is reached, the statement is aborted and an error message issued.

Assignment of line numbers

The current line number and the current increment determine the line numbers which are assigned. If the work file is empty, the current line number and the current increment are 1 by default.

In SDF-P V2.0, EDT needs an additional buffer of 8 memory pages, which it requests from the system.

@GETVAR Read S variable

Using @GETVAR, the contents of an S variable of the type STRING can be

- displayed on the screen
- assigned to a string variable as a value (max. 256 characters).

In systems in which the SDF-P subsystem is not installed, @GETVAR is rejected with an error message.

Operation	Operands	F mode / L mode
@GETVAR	$\left. \begin{array}{l} \text{string [=line =int-var] } \\ \text{SYSEDT} \end{array} \right\}$	

string	<p>Character string specifying the name of a simple S variable. “string” can be specified as follows:</p> <ul style="list-style-type: none"> – explicitly, enclosed in single quotes, or – implicitly via a line number, a line number variable or a string variable (in each case with a column range, if required).
line	<p>Number of the line into which the value of the STRING-type S variable is to be written. line can also be specified by means of line-number variables (#L00 through #L20), symbolically (e.g. %,\$) or as a string variable (#S00 through #S20).</p>
int-var	<p>Integer variable (#I0 through #I20) which is to receive the contents of the S variable. If the S variable is not of the INTEGER type, processing of the statement is aborted and an error message issued.</p>

If no entry is specified for int-var or for line, the contents of the S variable will be output on the screen if in interactive mode, or to SYSLST if in batch mode.

If a line is created with a line number greater than the previously highest line number, the current line number is changed.

SYSEDT	<p>The string variables #S00 to #S20 are assigned the contents of the S variables SYSEDT-S00 to SYSEDT-S20. If the SYSEDT operand is specified, no messages as to whether this assignment was successful or unsuccessful are issued, and no EDT5341 message is generated if the contents of the S variables are longer than 256 characters. No error switches are set.</p>
--------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

@GOTO Branch to line number in procedure

@GOTO is used within a procedure to branch to a specified line within this procedure.

Operation	Operands	@PROC
@GOTO	In	

In The target line number (e.g. 5).
 The minimum value is 0.0001, the maximum 9999.9999.
 In may also be specified as a line number variable (#L0 to #L20) or symbol-
 ically (e.g. %, \$).

The line to which @GOTO refers must exist in the associated procedure. If this is not the case, error message % EDT4974 LINE NOT IN PROCEDURE FILE is issued.

If @GOTO is to be used to branch to lines in an EDT procedure, it is advisable always to specify the line numbers of these lines by means of @In.

Example

```

1.      @SET #I3 = 1 ----- (01)
1.      @PROC 3
1.      @1 ----- (02)
1.      @ @IF #I3 > 5 RETURN ----- (03)
2.      @ @STATUS = #I3
3.      @ @SET #I3 = #I3+1
4.      @ @GOTO 1
5.      @END
1.      @DO 3 ----- (04)
#I03= 0000000001
#I03= 0000000002
#I03= 0000000003
#I03= 0000000004
#I03= 0000000005
1.
```

- (01) The integer variable #I3 is set to 1.
- (02) Specification of the line number of the line which is to be the destination of a @GOTO branch.
- (03) When the procedure in work file 3 is executed, the value of integer variable #I3 is to be incremented by 1 and displayed until this value is greater than 5. This is implemented by means of a loop in which the last statement is a @GOTO to return to the beginning of the loop.
- (04) The procedure in work file 3 is executed.

@HALT Terminate EDT

@HALT terminates

- the EDT run
- the screen dialog after @DIALOG
- EDT as a subroutine, with or without passing a text.

@HALT causes:

- termination of the EDT run
 - in a screen dialog if this was initiated by means of the command START-EDT, START-PROGRAM \$EDT or START-PROGRAM *MOD(\$EDTLIB,EDTC)
 - in BS2000 system procedures in which EDT was called
 - in the case of a subroutine call if it is entered in the CMD function (see the section on “IEDTCMD” in the manual “EDT Subroutine Interfaces” [1]) while processing a statement sequence
- termination of the screen dialog after @DIALOG
 - with a return to a system procedure
 - with continuation of the subroutine call.

Operation	Operands	F mode / L mode
@HALT	[ABNORMAL] [message]	

ABNORMAL If EDT was called as a main program, it is terminated abnormally. In procedures, processing resumes at the next JOB-STEP or in an ERROR-BLOCK.

If EDT was called as a subroutine, the character string is passed as part of the message to the calling program and a special return code is set.

message A character string which is passed to the calling program if EDT was called as a subroutine. This begins with the first non-blank character after @HALT and extends to the end of the statement.

In F mode, EDT also detects the end of the statement by a semicolon (;) in “message”.

There must be at least one blank between @HALT and “message”.

The length of “message” in F mode is limited by the statement length in the statement line (maximum of two continuation lines).

However, no more than 80 characters can be passed to the calling program.

This operand may be specified only if EDT is called as a subroutine; otherwise, @HALT message will be rejected with an error message.

If there are still work files which have not been saved, the numbers of these files are displayed after the message: % EDT0900 EDITED FILE(S) NOT SAVED!

This is accompanied by one of the following items, if available:

- a local @FILE entry
 - defined explicitly by @FILE LOCAL, or
 - defined implicitly by @READ, @GET, @OPEN (format 1),
- the library and element name of
 - a library element opened by means of @OPEN (format 2) or
- the file name of
 - a SAM or ISAM file opened with @OPEN (format 2) or
 - a POSIX file opened with @XOPEN.

The user then receives the following query:

```
% EDT0904 TERMINATE EDT? REPLY (Y=YES, N=NO)
```

N: In F mode the work window is displayed again. The user can close any files with unsaved data and write them back.

Y: Virtual files with unsaved data are lost. EDT is terminated.

The save query can be suppressed by setting task switch 4 before EDT is called. The message is likewise not issued if F mode was called by means of @DIALOG.

Example

```
% EDT0900 EDITED FILE(S) NOT SAVED!  
LOCAL FILE ( 0 ) :  
LOCAL FILE ( 1 ) :  
LOCAL FILE ( 4 ) : L= EDT164  
                  E= HALT,X  
% EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)?
```


@IF Query strings, line numbers, integers and switches

@IF can be used to

- query whether an EDT or DMS error occurred in a preceding (format 1)
- compare strings, line numbers or integers with each other (format 2)
- determine whether a previous @ON statement has found a hit (format 3)
- query the settings (on or off) of the 32 task switches and 32 user switches (format 4).

@IF (format 1) Query error switches

This format of @IF checks whether EDT or DMS errors have occurred. Depending on the result, a specified input string is processed or ignored.

EDT errors occur as the result of, for example, the entry of an incorrect EDT statement. The DMS error switch can either be set for statements involving files accesses (e.g. @WRITE format 1) or indicate errors in system accesses.

Operation	Operands	L mode / @PROC
@IF	{ ERRORS NO ERRORS } : text { DMS ERRORS NO DMS ERRORS }	

text Any character string.

If the first non-blank character in this string is

1. not the statement symbol, then any blanks following the : are regarded as part of the text.

The following processing guidelines apply:

- “text” is placed in the current line;
- the current line number is incremented by the current increment value;
- any tab characters are interpreted.

2. the statement symbol, then any blanks following the : are ignored.
If the next character is
 - not the statement symbol, then “text” is interpreted as an EDT statement and executed immediately;
 - the statement symbol, then “text” is treated as a text line as described in 1), above.

If “text” is not specified, @IF has no effect.

ERRORS text is executed if the EDT error switch has been set.

NO ERRORS text is executed if the EDT error switch has not been set.

DMS ERRORS
text is executed if the DMS error switch has been set.

NO DMS ERRORS
text is executed if the DMS error switch has not been set.



- The error switches must be reset before the statement which is to be checked is executed (see @RESET).
If this is not done, @IF may return an incorrect result because earlier statements may have set the EDT or DMS error switch.
- Within EDT procedures (@DO), @IF ERRORS must not be used to check for hits after @ON. Instead, format 3 of @IF must be used.

@IF (format 2) Query strings, line numbers and numbers

This format of @IF compares

- line contents or string variables
- line numbers or line number variables
- integer variables.

If the result of the comparison is positive, @IF

- branches to a line within the procedure (GOTO In) or
- aborts execution of the current procedure (RETURN).

If the result of the comparison is negative, EDT continues execution of the procedure at the line following the @IF statement.

Operation	Operands	@PROC
@IF	$\left\{ \begin{array}{l} \text{[S] string1 rel string2} \\ \text{In1 rel In2} \\ \text{[I] int1 rel int2} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{GOTO In} \\ \text{RETURN} \end{array} \right\}$

S Mandatory only if string1 and string2 contain line numbers without column numbers. In such cases, EDT cannot determine whether the contents of the lines are to be compared or simply the line numbers.

@IF#L1 = #L2..., for example, compares the line numbers #L1 and #L2. If, in contrast, the user wishes to compare the contents of lines #L1 and #L2, he/she must specify @IF S #L1 = #L2.

string1
string2

The strings to be compared with each other.

The strings may be specified:

- explicitly, enclosed in single quotes, or
- implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

For example, the string 'HUGO' is permissible. If the character string variable #S18 contains the text 'ABCD456DEF', then specifying the string '456ABCE' is equivalent to specifying #S18:5-7,1-3,9:.

ln1
ln2 The line numbers to be compared with each other. The contents of the lines are not compared. The minimum value is 0.0001, the maximum 9999.9999. ln may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %,\$).

I Needs to be specified only if a number is entered for int1 (otherwise, EDT does not know whether this is a line number or an integer).

int1
int2 The integers to be compared with each other.

Each of the operands may be a (positive or negative) integer or an integer variable (#I0,...,#I20).

rel Relational operators:

Symbol			Meaning
GT	or	>	greater than
LT	or	<	less than
GE	or	>=	greater than or equal to
LE	or	<=	less than or equal to
EQ	or	=	equal to
NE	or	<>	not equal to

ln could be either %+3L or %+3.

If followed by the relational operator LE, this could lead to interpretation problems. It is therefore advisable to make a habit of using the mathematical symbols for the relational operators.

GOTO ln If the result of the comparison is positive, control is passed to the specified line number (ln) in the procedure.

RETURN If the result of the comparison is positive, the current procedure is aborted.

Comparison of strings

The way in which two strings are compared depends on the lengths of these strings. (Note that a string with a length of zero is permitted.)

Case 1: Both strings are the same length

The corresponding characters of the two strings are compared with each other, working from left to right.

The result of this will either be a pair of characters which do not match or the two strings are regarded as identical. If the characters in any one position of the two strings are not the same, then the two strings are not identical. EDT interprets characters as binary numbers on the basis of their EBCDIC codes and the string with the higher binary number is regarded as greater than the other string.

Case 2: The two strings are different lengths

Basically, the comparison is executed as in case 1. When EDT has compared all of the characters in the shorter string with the corresponding characters in the longer string and has found no non-matching pairs, the longer string is regarded as the greater. If a non-matching pair is found before the end of the shorter string is reached, EDT interprets the non-matching characters as binary numbers on the basis of their EBCDIC codes. The string whose character has the higher binary number is regarded as greater than the other string.

If the two strings are different lengths, they can never be equal.

Example 1

```

4.      @PRINT
1.0000 PLEASE DO NOT LAUGH
2.0000 AT THIS EXAMPLE
3.0000 PLEASE DO NOT LAUGH
4.      @SET #S0 = 'FIRST LINE = LAST LINE
4.      @SET #S1 = 'FIRST LINE NOT EQUAL TO LAST LINE ----- (01)
4.      @SET #I9 = 2
4.      @PROC 3
1.      @ @IF S%+#I9 = $-2L GOTO 4 ----- (02)
2.      @ @PRINT #S1 N
3.      @ @RETURN
4.      @ @PRINT #S0 N
5.      @END
4.      @DO 3 ----- (03)
FIRST LINE = LAST LINE
4.      @ON 1 DELETE 'NOT'
4.      @DO 3 ----- (04)
FIRST LINE NOT EQUAL TO LAST LINE
4.

```

- (01) The string variables #S0 and #S1 and the integer variable #I9 are set to certain values.
- (02) When the procedure in work file 3 is executed, the contents of the lines (not the line numbers) are compared with each other.
- (03) Execution of the procedure in work file 3 causes lines 3 (%+#I9) and 1 (\$-2L, i.e. 3-2) to be compared. Since the contents of these lines are identical, EDT branches to line 4 in work file 3.
- (04) Since the contents of line 1 have now been changed, EDT no longer branches to line 4 of work file 3.

Example 2

```

1.    @SET #S4 = 'M' ----- (01)
1.    @PROC 4
2.    @ @PRINT #S4 N ----- (02)
2.    @ @CREATE #S4: 'M',#S4
3.    @ @IF #S4 < 'M'*8 GOTO 1
4.    @END
1.    @DO 4

M
MM
MMM
MMMM
MMMMM
MMMMMM
MMMMMMM
MMMMMMMM
1.

```

(01) The string variable #S4 is set to the character 'M'.

(02) The following procedure is entered in work file 4: display the contents of #S4 and then insert the letter 'M' in front of the current contents of #S4.

If the contents of #S4 are less than 'MMMMMMMM', execute the loop again.

Example 3

```

9.      @PRINT
1.0000 ABC
2.0000 WHO
3.0000 ABC
4.0000 WANTS
5.0000 ABC
6.0000 TO TRY
7.0000 ABC
8.0000 HIS HAND?
9.      @PROC 6
1.      @ @IF ! <> 'ABC' GOTO 3 ----- (01)
2.      @ @CREATE !: '*' * 20
3.      @ @CONTINUE
4.      @END
9.      @DO 6,!=%,$ ----- (02)
9.      @PRINT
1.0000 *****
2.0000 WHO
3.0000 *****
4.0000 WANTS
5.0000 *****
6.0000 TO TRY
7.0000 *****
8.0000 HIS HAND?
9.

```

- (01) In work file 6, the line numbers are addressed via the loop symbol !. If the line addressed by ! does not contain 'ABC', this line is to be left unchanged. Otherwise, the line contents are to be replaced by '*****'.
- (02) Work file 6 is executed, addressing all lines of the current work file in consecutive order via the loop symbol !.

Example 4

```

4.      @PRINT
1.0000 PLEASE DO NOT LAUGH
2.0000 AT THIS EXAMPLE
3.0000 IT IS TOO SIMPLE
4.      @SET #S0 = 'RESULT POSITIVE'
4.      @SET #S1 = 'RESULT NEGATIVE' ----- (01)
4.      @SET #I9 = 1
4.      @PROC 1 ----- (02)
1.      @ @IF %+#I9 = $-1L GOTO 4 ----- (03)
2.      @ @PRINT #S1 N
3.      @ @RETURN
4.      @ @PRINT #S0 N
5.      @END
4.      @DO 1 ----- (04)
RESULT POSITIVE
4.      @SET #I9 = 2
4.      @DO 1 ----- (05)
RESULT NEGATIVE
4.

```

- (01) The string variables #S0 and #S1 are set to the desired values. Integer variable #I9 is set to 1.
- (02) Open work file 1.
- (03) After a subsequent @DO 1, this line compares the line numbers % + #I9 and \$ - 1L with each other.
- % addresses the first line number - in this case 1.
- \$ addresses the last line number - in this case 3.
- \$ - 1L addresses the last line number but one - in this case 2.
- (04) The procedure in work file 1 is executed. At this time, the specified relationship % + #I9 = \$ - 1L is true, since 1+1 = 3-1 is true.
- (05) At this time, the specified relationship % + #I9 = \$ - 1L is not true, since 1+2 = 3-1 is false.

Example 5

```

4.      @PRINT
1.0000 PLEASE DO NOT LAUGH
2.0000 AT THIS EXAMPLE
3.0000 IT IS TOO SIMPLE
4.      @SET #L3 = 5
4.      @PROC 2
1.      @ @IF %+6-#L3 <> $-* RETURN ----- (01)
2.      @ @CREATE $+1: 'OR PERHAPS NOT'
3.      @ @PRINT
4.      @END
4.      @$-1
2.      @DO 2 ----- (02)
2.      @1
1.      @DO 2 ----- (03)
1.0000 PLEASE DO NOT LAUGH
2.0000 AT THIS EXAMPLE
3.0000 IT IS TOO SIMPLE
4.0000 OR PERHAPS NOT
1.

```

- (01) If, when work file 2 is executed, the specified condition is not true (<> means not equal to), the procedure is aborted at this point.
- (02) Work file 2 is executed. Since $* = \$ - 1 = 2$, the expression $\% + 6 - \#L3 <> \$ - *$ is equivalent to $1+6-5 <> 3-2$ and thus true. Execution of the work file is therefore aborted.
- (03) At this time, $* = 1$, which means that the condition $\% + 6 - \#L3 <> \$ - *$ is false, since $1+6-5 = 3-1$. As a result, the remaining statements in work file 2 are executed.

Example 6

```

1.      @SET #I3 = 1 ----- (01)
1.      @PROC 7
1.      @ @IF #I3 > 5 RETURN
2.      @ @STATUS = #I3 ----- (02)
3.      @ @SET #I3 = #I3+1
4.      @ @GOTO 1
5.      @END
1.      @DO 7 ----- (03)
#I03= 0000000001
#I03= 0000000002
#I03= 0000000003
#I03= 0000000004
#I03= 0000000005
1.

```

- (01) Integer variable #I3 is set to 1.
- (02) The procedure in work file 7 is to display (@STATUS = #I3) and increment (#I3 + 1) the value of integer variable #I3 until this value exceeds 5 for the first time.
- (03) Work file 7 is executed.

@IF (format 3) Query @ON hits or empty work files

Format 3 of @IF checks whether EDT found a hit when @ON was last executed. Depending on the result of this check, EDT

- branches to the specified line in the procedure (GOTO In),
- aborts execution of the current procedure (RETURN), or
- continues execution of the procedure in the line following the @IF statement.

Operation	Operands	@PROC
@IF	$\left\{ \begin{array}{l} \text{.TRUE. [rel cl]} \\ \text{.FALSE.} \\ \text{.EMPTY.} \end{array} \right\} \left\{ \begin{array}{l} \text{GOTO In} \\ \text{RETURN} \end{array} \right\}$	

.TRUE. GOTO In or RETURN is executed if the last @ON statement executed detected a hit.

If rel and cl are specified, the branch is not executed immediately. First, EDT compares the number of the column in which the hit was detected with the column number specified for cl. If the result of this comparison is positive, GOTO In or RETURN is executed.

rel Relational operators:

Symbol	Meaning
GT or >	greater than
LT or <	less than
GE or >=	greater than or equal to
LE or <=	less than or equal to
EQ or =	equal to
NE or <>	not equal to

cl A column number (integer between 1 and 256 or an integer variable). This number is compared with the number of the column in which the last @ON statement found the first hit.

.FALSE. GOTO In or RETURN is executed if no hit was detected by the last @ON statement executed.

.EMPTY. GOTO In or RETURN is executed if the current work file is empty, i.e. contains no data lines.

In A line number (e.g. 5).
 The minimum value is 0.0001, the maximum 9999.9999.
 In may also be specified as a line number variable (#L0 to #L20) or symbol-
 ically (e.g. %,\$).

If the specified condition is not fulfilled, EDT continues execution of the procedure in the line following the @IF statement.

Example 1

```

5.      @PRINT
1.0000 WHICH
2.0000 WAY TO
3.0000 THE MAIN
4.0000 STATION?
5.      @PROC 8
1.      @ @ON ! FIND 'T'
2.      @ @IF .FALSE. GOTO 4 ----- (01)
3.      @ @CREATE !: '*'*20
4.      @ @CONTINUE
5.      @END
5.      @DO 8, !=%, $ ----- (02)
5.      @PRINT
1.0000 WHICH
2.0000 *****
3.0000 *****
4.0000 *****
5.

```

- (01) In work file 8, the line numbers are addressed via the loop symbol !. If one of the lines addressed in this manner does not contain the letter I, it is to remain unchanged. Otherwise, the contents of the line are to be changed to '*****'.
- (02) Work file 8 is executed, addressing all lines of the main file in sequential order with the aid of the loop symbol !.

Example 2

```

5.      @PRINT
1.0000 WHICH
2.0000 WAY TO
3.0000 THE MAIN
4.0000 STATION?
5.      @PROC 9
1.      @ @ON ! FIND 'AI'
2.      @ @IF .TRUE. = 6 GOTO 4
3.      @ @GOTO 5 ----- (01)
4.      @ @SUFFIX ! WITH ' RAILWAY'
5.      @ @CONTINUE
6.      @END
5.      @DO 9, !=%, $ ----- (02)
5.      @PRINT
1.0000 WHICH
2.0000 WAY TO
3.0000 THE MAIN RAILWAY
4.0000 STATION?
5.

```

- (01) In the procedure in work file 9, the line numbers are addressed via the loop symbol !. If one of the lines addressed in this manner contains the string AI in columns 6 and 7, then the string RAILWAY is to be appended to this line. Otherwise, the contents of the line are to remain unchanged.
- (02) The procedure in work file 9 is executed, addressing the lines of the main file in sequential order with the aid of the loop symbol !.

@IF (format 4) Query task and user switches

Format 4 of @IF checks which task or user switches are on or off (see @SETSW and section “Task switches” on page 69ff). Depending on the result of this check, EDT

- branches to the specified line in the procedure (GOTO In),
- aborts execution of the procedure (RETURN), or
- continues execution of the procedure in the line following the @IF statement.

Operation	Operands	@PROC
@IF	$\left. \begin{array}{l} \text{ON} \\ \text{OFF} \end{array} \right\} = [\text{U}] \text{ int} \left. \begin{array}{l} \text{GOTO In} \\ \text{RETURN} \end{array} \right\}$	

ON EDT checks whether the specified switch is on.

OFF EDT checks whether the specified switch is off.

U If “U” is specified, the user switches are queried; if not, the task switches are queried.

int Number of the switch to be checked. This must be specified as an integer between 0 and 31 (or as an integer variable). If the U parameter is specified before the switch number, the user switch int under the user’s own ID is checked instead of the specified task switch int.

In A line number (e.g. 5).
The minimum value is 0.0001, the maximum 9999.9999.
In may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %, \$).

Example

```

1.      @SET #S2 = 'SWITCH 15 IS OFF'
1.      @SET #S3 = 'SWITCH 15 IS ON'
1.      @PROC 8
1.      @ @IF ON = 15 GOTO 4
2.      @ @PRINT #S2 N
3.      @ @RETURN ----- (01)
4.      @ @PRINT #S3 N
5.      @END
1.      @SETSW OFF = 15 ----- (02)
1.      @DO 8 ----- (03)
SWITCH 15 IS OFF
1.      @SETSW ON = 15 ----- (04)
1.      @DO 8 ----- (05)
SWITCH 15 IS ON
1.

```

- (01) The procedure in work file 8 is to display string variable #S3 if task switch 15 is on or string variable #S2 if this switch is off.
- (02) This resets task switch 15.
- (03) The procedure in work file 8 is executed.
- (04) Switch 15 is set.
- (05) Work file 8 is executed.

@INPUT Define input mode or start procedure

@INPUT can be used to

- read in and process part or all of an @INPUT procedure from a SAM or ISAM file (format 1)
- start an @INPUT procedure from a file or library element (format 2)
- define the input mode for input in L mode (format 3).

@INPUT (Format 1) Start an @INPUT procedure from a SAM or ISAM file

This format starts an @INPUT procedure: the contents of a SAM or ISAM file are read into the user address space and the statements and text lines in this input are processed immediately.

Operation	Operands	F mode / L mode
@INPUT	'file' [(ver)] [range*] [:col:] [{ RECORDS } { KEY }]][PRINT]

file The name of the SAM or ISAM file to be read in and processed.

ver The version number of the file.
This may consist of up to three digits or an asterisk (*). * designates the current version number. If an incorrect version number is specified, the @INPUT procedure is still read in and processed.

range* The line range in the SAM or ISAM file which is to be processed. If range* is omitted, all lines in the file are processed.

ISAM file	file is a	
	SAM file	
range* may be: <ul style="list-style-type: none"> - a single line, e.g. 5 - a contiguous line range, e.g 6-12 - a string of lines and/or line ranges, separated by commas, e.g. 6-12, 5, 19. Lines or line ranges may be specified more than once, e.g. 5, 5, 8-10, 9. 	range* may be:	
	<ul style="list-style-type: none"> - a single line, e.g. 5 or .0005 - a contiguous line range, e.g. 6-12 or .0006-.0012 	
	Is RECORDS specified?	
	Yes	No
	The line range to be read is to be addressed with the aid of the logical line numbers. The logical line number of the first line is 0.0001, that of the second line 0.0002, etc.	Specification of range* is meaningful only if the INPUT file was created by means of @WRITE with the KEY operand. In such a file, the first eight characters in each line contain the number which the line had in the virtual file. range* refers to these numbers, i.e. EDT interprets the first eight characters in each line as a line number, not as part of the line contents. Line number 5, for example, refers to the line which starts with 00050000.

The line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %, \$) or via line number variables. However, their values do not refer to the file specified by "file", but to the current virtual file or file opened by means of @OPEN.

col

A column range, specified as:

- one or more columns, separated by commas (e.g. 10,15,8)
- one or more column ranges, separated by commas (e.g. 15-25,18-23)
- a combination of columns and column ranges (e.g. 10,14-29,23-50,17).

If no column range is specified, the full length of each line is read in.

- KEY** Must be specified for SAM files written using @WRITE with the KEY operand. The first 8 characters in each line of such a file contain the number which this line had in the virtual file. If KEY is specified in @INPUT, these 8 characters are regarded as a line number, not as part of the line contents, when the file is read in. If KEY were not specified, EDT would regard all characters in each line of the INPUT file as text characters.
- RECORDS** Must be specified for SAM files if the user wishes to select a line range with the aid of the logical line numbers.
- PRINT** Causes each line read from the SAM or ISAM file to be displayed.

By specifying range* and col, it is possible to select parts of the @INPUT procedure.

@INPUT must not be used in @INPUT or @DO procedures.

Processing of an @INPUT procedure is terminated if

- a @RETURN statement is found,
- the results of a comparison in an @IF statement with the RETURN operand are positive or
- a DMS error occurs during execution of a statement.



Any keys contained in the file are not checked for errors. The line number is calculated from the first eight characters in each line of a SAM file. If the calculated line number is less than the specified line range, or if no key can be formed from these eight characters, this line is ignored. If the calculated line number is greater than the specified line range, this line and all succeeding lines are ignored.

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of the file is taken into account in an @INPUT statement.

The @INPUT statement is only executed if the CCSN of the file is the same as the CCSN currently selected in EDT or all work files are empty and the coded character set can be displayed on the data display terminal.

Example

```
6.      @PRINT
1.0000 @DELETE
2.0000 THIS IS LINE 1
3.0000 THIS IS THE SECOND LINE
4.0000 @PRINT 1
5.0000 @PRINT 2
6.      @WRITE 'SAM-INP' KEY ----- (01)
6.      @SAVE 'ISAM-INP' ----- (02)
6.      @INPUT 'SAM-INP' & ----- (03)
1.0000 THIS IS LINE 1
2.0000 THIS IS THE SECOND LINE
3.      @INPUT 'ISAM-INP' 1-3,5,4 ----- (04)
2.0000 THIS IS THE SECOND LINE
1.0000 THIS IS LINE 1
3.
```

- (01) The contents of the work file are written into a SAM file, placing a key formed from the line number at the beginning of each line.
- (02) The contents of the work file are written again, but this time as an ISAM file.
- (03) The entire file SAM-INP is to be read in and processed. Since this file was created using @WRITE with the KEY operand, either KEY or range* (in this case: &) must be specified. Otherwise, the stored keys will not be converted into line numbers.
- (04) Lines 1-3, 5, 4 (in this order) of file ISAM-INP are to be read and processed.

@INPUT (Format 2) Start an @INPUT procedure from a library or file

This format starts an @INPUT procedure from a library or from a file. A library element or a file is read into the user address space. The statements and text lines read in are processed immediately.

Operation	Operands	F mode / L mode
@INPUT	$\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT]=)elemname [(vers)][,elemtyp]} \\ \text{ELEMENT=elemname [(vers)][,elemtyp]} \\ \text{FILE=path2} \end{array} \right\}$ [PRINT]	

LIBRARY = path1 ([ELEMENT]=)elemname [(vers)][,elemtyp]

The name of the library and of the desired element.

ELEMENT = elemname [(vers)][,elemtyp]

The name of the desired element, without a library name. In this case, the library name must have been preset by means of @PAR.

path1 The library name.

path1 may also be specified by means of a string variable.

If path1 is omitted, the default library specified by means of @PAR LIBRARY is used.

elemname The element name. elemname may also be specified by means of a string variable.

vers The version number of the desired element (see the "LMS" manual [14]). If vers is not specified or if *STD is specified, the highest available version of the element is selected.

elemtyp The element type. elemtyp may also be specified by means of a string variable.

Permissible type entries: S, M, P, J, D, X, *STD or user-defined type names with appropriate base type. If no type is specified, the value preset in @PAR ELEMENT-TYPE will be used.

Users who specify a user-defined type name are responsible for ensuring that its associated base type corresponds to one of the permissible types S, M, P, J, D or X.

Type	Element contents
S	Source programs
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format

***STD**

Type S is the default value when EDT is started. Any other valid type specification can be set as the default value by means of @PAR.

FILE = path2 This operand is used to read in a BS2000 file.

path2 Name of the file to be read in as the @INPUT procedure. path2 may also be specified by means of a string variable.

PRINT If PRINT is specified, each line is displayed on the screen as it is read in.

@INPUT must not be used in @INPUT or @DO procedures.

Processing of an @INPUT procedure is terminated if

- a @RETURN statement is found,
- the results of a comparison in an @IF statement with the RETURN operand are positive or
- a DMS error occurs during execution of a statement.

Calculation of line numbers

As they are inserted, the records are numbered in one of three ways:

1. Standard numbering with standard increment 1.0000 (e.g. 21.0000, 22.0000, 23.0000 ... 99.0000)
2. Numbering with a preset increment as defined in @PAR INCREMENT
3. Automatic numbering and renumbering, if the selected increment is too large to permit inclusion of the records to be inserted. EDT then selects an increment which is smaller, by a factor of 10, than the standard (case 1) or specified (case 2) increment, and attempts to number the copied records with this increment. This is repeated until the copied records can be included successfully or until EDT selects the minimum increment of 0.01.

Renumbering if @PAR RENUMBER=ON is specified:

If the copied records cannot be included with the minimum increment of 0.01, EDT automatically renumbers the lines following the target range with the increment value 0.01.

If EDT cannot find sufficient space, no records are inserted into the work file and an error message is issued. When copying into an empty work file, EDT calculates the line number for the first record by adding the standard increment or the specified increment (@PAR INCREMENT) to an initial value of 0.



- If @PAR INCREMENT is entered with an increment < 0.01, it should be noted that the line numbers of lines which have been read in, copied or inserted are not shown fully in F mode (6-digit line number display). If these incomplete line numbers are then used in @INPUT statements, unpredictable results may be produced.
- If this statement is entered in L mode and results in the creation of a line with a number higher than that of the highest existing line number, the current line number is changed.

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of the file (library element) is taken into account in an @INPUT statement.

The @INPUT statement is only executed if the CCSN of the file (library element) is the same as the CCSN currently selected in EDT or all work files are empty and the coded character set can be displayed on the data display terminal.

@INPUT (Format 3) Define EDT input mode

By means of @INPUT, format 3, the user specifies that EDT is to interpret text input as:

- a sequence of printable characters,
- a sequence of hexadecimal characters (EBCDIC or ISO), or
- a sequence of binary characters.

Operation	Operands	L mode
@INPUT	{ [CHAR] HEXIX [ISO] BINARY }	

- CHAR** EDT is to interpret all text inputs as a sequence of printable characters.
- HEX,X** EDT is to interpret all text inputs as a sequence of hexadecimal characters. If an odd number of hexadecimal characters is entered, a zero is inserted on the left.
Since the maximum length of an input line is 256 (hexadecimal) characters, the maximum number of (text) characters in one line is 128.
- ISO** EDT is to interpret the hexadecimal input as ISO code (ASCII). The data itself is then added as EBCDIC to the work file.
The conversion table corresponds to the correlation of Coded Character Set EDF03IRV to ISO646 international 7-bit code (or EDF041 to ISO8859-1).
If ISO is not specified, EDT expects input in EBCDIC.
- BINARY** EDT is to interpret all text inputs as a sequence of binary characters. If the number of characters entered is not a multiple of 8, the appropriate number of zeros is inserted on the left.
Since the maximum length of an input line is 256 (binary) characters, the maximum number of (text) characters in one line is 32.

The default value when EDT is started is CHAR.



Statements must always be entered as a sequence of printable characters.

@LIMITS Display line numbers

Entering @LIMITS causes EDT to output the following information for the current work file:

- the lowest line number assigned,
- the highest line number assigned and
- the number of lines.

The output is directed to SYSOUT.

Operation	Operands	F mode / L mode
@LIMITS		

Example

```

4.      @PRINT
1.0000 A
2.0000 B
3.0000 C
4.      @LIMITS
1.0000 TO    3.0000          3 LINES ----- (01)

4.      @COPY 1-3 TO 99.01 ----- (02)

100.03  @LIMITS
1.0000 TO    99.0300        6 LINES ----- (03)

100.03

```

- (01) The lowest and highest line numbers are displayed.
- (02) Lines 1-3 are moved to 99.01, 99.02 and 99.03.
- (03) Now the lowest and highest line numbers are 1.0000 and 99.0300, respectively. The number of lines is 6.

@LIST Print contents of work file

With the aid of @LIST, any desired parts of a work file can be printed on a printer.

Operation	Operands	F mode / L mode
@LIST	[rng [:domain] [X] [N] [C [int] P int] [I] [S]] [,...]	

- rng A line range, specified as:
- a single line (e.g. 6)
 - several contiguous lines (e.g. 8-20).
- The line range may also be specified by means of the current range symbol (see @RANGE), via symbolic line numbers (e.g. %,\$) or using line number variables. String variables (#S0 to #S20) may also be used.
- If rng is omitted, the entire file is printed. rng must be specified if X, N, C int, P int, I or S is used.
- domain A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, each line is printed from this column to the end of the line. If the first column number is greater than the line length, this line is ignored.
- The second column number
- must not be less than the first column number
 - may be greater than the actual line length.
- If no domain is specified, the entire line is printed.
- X The lines are to be printed in hexadecimal form. A maximum of 128 characters in each line are printed.
- N Line numbers are to be suppressed in the output.
- C The first character in each line controls the printing but is not itself printed.

Contents of the first character	Control function
X'C1'	Form feed before printing
X'40' or X'00'	1 line feed before printing
X'41' or X'01'	2 line feeds before printing
X'42' or X'02'	3 line feeds before printing
X'4F' or X'0F'	16 line feeds before printing

- P** Controls the form feed function.
- int** Number between 0 and 256
- C int** If a number between 1 and 256 is specified, EDT inserts a form feed after printing int lines. If 0 is specified for int, no check is performed. If int is specified without C, a form feed is inserted after exactly int lines, or if int=0, no form feed is inserted.
- P int** A form feed is inserted after exactly int lines. If 0 is specified for int, no form feed is inserted.
If P is used, rng must be specified.
- I** Printing starts immediately.
I is permitted in interactive mode only.
If I is not specified, output is directed to SYSLST and, if SYSLST is not assigned to a file, printing does not begin until after LOGOFF.
- S** Suppresses the empty lines which normally precede the first printed line.

Example

```

6.      @PRINT
1.0000 THE @LIST STATEMENT
2.0000 PERMITS THE CONTENTS
3.0000 OF A WORK FILE TO BE
4.0000 TRANSFERRED TO PAPER
5.0000 IN ANY DESIRED FORM.
6.      @LIST ----- (01)
6.      @LIST 4-5 N ----- (02)
6.      @LIST & X ----- (03)
6.      @LIST & I ----- (04)
% SCP0810 SPOOLOUT OF FILE
':A:$USER0001.S.SPS.5660.01.23.89023.115714'ACCEPTED
: TSN: '5666', PNAME: 'NAME'
6.

```

- (01) The entire contents of the work file are to be printed. Printing is to be started after / LOGOFF.

Printed output

1.0000 THE @LIST STATEMENT
2.0000 PERMITS THE CONTENTS
3.0000 OF A WORK FILE TO BE
4.0000 TRANSFERRED TO PAPER
5.0000 IN ANY DESIRED FORM.

- (02) Lines 4 and 5 are to be printed after LOGOFF, with the line numbers suppressed.

Printed output

TRANSFERRED TO PAPER
IN ANY DESIRED FORM.

- (03) All lines are to be printed in hexadecimal format after /LOGOFF.

Printed output

1.0000 D4C9E340C4C5D4407CD3C9E2E360D2D6D4D4C1D5C4D6
2.0000 E6C9D9C440C4C5D940C9D5C8C1D3E3
3.0000 C5C9D5C5D940C1D9C2C5C9E3E2C4C1E3C5C9
4.0000 C9D540D1C5C4C5D940C7C5E6E4C5D5E2C3C8E3C5D5
5.0000 C6D6D9D440E9E440D7C1D7C9C5D940C7C5C2D9C1C3C8E34B

- (04) All lines are to be printed immediately.

Printed output

1.0000 THE @LIST STATEMENT
2.0000 PERMITS THE CONTENTS
3.0000 OF A WORK FILE TO BE
4.0000 TRANSFERRED TO PAPER
5.0000 IN ANY DESIRED FORM.

@LOAD Load program

The @LOAD statement

- terminates the EDT session and
- loads the specified program.

@LOAD is one of the EDT statements that is relevant to security (see [section “Data protection” on page 71](#)). In uninterruptible system procedures in interactive mode and in the case of input from a file, the statement will be rejected (unless it is read from SYSDTA=SYSCMD).

Operation	Operands	F mode / L mode
@LOAD	string	

string A character string specifying the name of the program to be loaded. The string may be specified:

- explicitly, enclosed in single quotes, or
- implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

If there are still work files which have not been saved, the numbers of these files are displayed after the message: % EDT0900 EDITED FILE(S) NOT SAVED!

This is accompanied by one of the following items, if available:

- a local @FILE entry
 - defined explicitly by @FILE LOCAL, or
 - defined implicitly by @READ, @GET, @OPEN (format 1),
- the library and element name of
 - a library element opened by means of @OPEN (format 2)
- or the file name of
 - a SAM or ISAM file opened with @OPEN (format 2) or
 - a POSIX file opened with @XOPEN.

The user then receives the following query:

% EDT0904 TERMINATE EDT? REPLY (Y=YES, N=NO)?

N: In F mode the work window is displayed again. The user can close any files with unsaved data and write them back.

Y: Virtual files with unsaved data are lost. EDT is terminated and the specified program started.

If a file was opened in real mode by means of @OPEN, this query is not displayed, since EDT closes the file by means of an implicit @CLOSE statement.

The save query can be suppressed by setting task switch 4 before EDT is called.

Example

```

1.00 The @LOAD statement.....
2.00 - terminates the EDT session and.....
3.00 - loads the specified program.....
4.00 .....

load '$lms'.....0001.00:001(0)

```

EDT is to be terminated and LMS loaded.

```

% EDT0900 EDITED FILE(S) NOT SAVED!
  LOCAL FILE ( 0 ) :
% EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)?y
% BLS0500 PROGRAM 'LMS', VERSION 'V3.0A' OF 'yy-mm-dd' LOADED.
/resume-program
% LMS0310 LMS VERSION V03.0A00 LOADED
CTL=(CMD) PRT=(OUT)
$

```

Since the work file has not been saved, EDT asks, just as for @HALT, whether it is really to be terminated.

Since @LOAD, rather than @EXEC, was specified, the slash is used to indicate that further system commands are expected. LMS is started only when the RESUME-PROGRAM command is entered.

@LOG Control logging in batch mode

@LOG controls the logging of inputs made in batch operation and in interactive mode.

The output may be directed to:

- SYSLST (a high-speed printer),
- SYSLSTnn or a file assigned to SYSLSTnn or
- a list variable.

Operation	Operands	F mode / L mode
@LOG	[{ ALL COMMANDS NONE }] [{ SYSLST SYSLST nn LIST-VAR=chars }]	

ALL EDT is to log all L-mode inputs (text and statements) which are entered via RDATA or at the display terminal. Inputs made in an F-mode dialog are logged in the statement line (in the case of statement sequences, separated into individual statements).

COMMANDS Only statements are to be logged.

NONE Nothing is to be logged.

When EDT is called in batch mode, the default depends on task switch 4:

- If SETSW ON=4, @LOG NONE is set.
- If SETSW OFF=4, @LOG COMMANDS is set.

If EDT is called in a mode other than batch mode, @LOG NONE is set.

SYSLST Directs the log to SYSLST. This is the default.

SYSLST nn nn = 1,...99
Directs the log to the file assigned to SYSLSTnn.

LIST-VAR Directs the log to a list variable.

chars A string specifying the name of a list variable. This S variable must have been defined beforehand, e.g. with DECLARE-VARIABLE chars, MULTI-ELEMENT=LIST. The individual elements of the list must be of the ANY or the STRING type. The individual log lines are appended to the list.

If none of the SYSLST, SYSLSTnn or VAR operands is specified, the output destination remains unchanged.

This statement is also executed in test mode.

@LOWER Specify uppercase and lowercase display

@LOWER is used to specify whether or not EDT is to convert lowercase letters in the input into uppercase letters.

The setting selected by means of @LOWER is valid for all work files, regardless of the work window in which the statement was entered.

Operation	Operands	F mode / L mode
@LOWER	[ON] <u>OFF</u>	

- ON EDT makes a distinction between uppercase and lowercase letters. All character strings are processed exactly as they are entered.
- OFF EDT converts the lowercase letters a, ..., z in the input into the uppercase letters A, ..., Z and displays them on the screen. This also applies to strings entered as operands in EDT statements. The German umlaut characters ä, ö and ü are not converted; they are reproduced in the work file in lowercase form.

In F mode, any lowercase letters in work files are displayed as smudge characters when the file is displayed. In L mode, lowercase letters are displayed in printable form.

The default value when EDT is started is OFF. If @LOWER is entered without an operand, @LOWER ON is set.

If @LOWER is used within an input block (see @BLOCK), the conversion mode is changed only when all statements in the block have been processed. @LOWER should therefore always be specified at the end of such an input block.

If XHCS is installed in the system, the conversion table associated with the coded character set (CCS) is used for converting lowercase letters into uppercase letters.

Processing of lowercase letters in EDT statements

- If @LOWER OFF is specified, screen entries are converted from lowercase letters to uppercase letters in all processing modes.
In the case of inputs from SYSDTA files, @INPUT files or procedure files or inputs read via the subroutine interface, EDT expects statements in uppercase letters.
- If @LOWER ON is specified, only lowercase letters in the 'string' and 'xpath' operands are not converted to uppercase letters in F mode dialog. In all other processing modes (L mode, procedure mode, etc.) lowercase letters in the operands string, text, param and xpath are not converted.

@MOVE Move line ranges

@MOVE is used to move a line or a range of lines to a specified line range, deleting the original line(s) afterwards. The lines can be moved within the current work file or from any other work file to the current work file.

It is not possible to move lines from a work file which is currently being executed as an EDT procedure, i.e. from an active work file (see @DO).

Operation	Operands	F mode / L mode
@MOVE	rng [(procno)] [TO ln1 [(inc)] [:] [ln2]] [,...]	

The operands TO and ln1 must always be specified if the lines to be moved are in the current work file. If lines are to be moved from another work file to the current work file, omission of TO and ln1 causes the lines to retain their line numbers.

rng A line range, specified as:
 – a single line (e.g. 6)
 – several contiguous lines (e.g. 8-20).

A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.

The symbolic line numbers refer to the current work file, i.e. the values of the symbolic line numbers correspond to the line numbers of the current work file and not of the work file from which data is transferred.

procno The number (0-22) of the work file from which the lines are to be moved. If procno is omitted, the lines to be moved are assumed to be in the current work file.

ln1 The number of the first line of the target range.
 EDT calculates the numbers of the following lines in the target range by incrementing this line number by the increment value specified for this range. The minimum value is 0.0001, the maximum 9999.9999.
 If inc is not specified, EDT uses the increment value implied by the number of decimal places in the line number: for example, 5 implies an increment of 1 and 5.0 implies an increment value of 0.1.

ln1 may also be specified as a line number variable or symbolically.

- inc The increment value for the target range.
 The minimum value is 0.0001, the maximum 9999.9999.
- :
- In2 The number of the last line in the target range.

 @MOVE moves line by line. When this upper limit is reached, the move operation is terminated, even if there are still lines in the source range which have not been moved. The minimum value is 0.0001, the maximum 9999.9999. In2 may also be specified as a line number variable or symbolically.
 If In2 is not specified, the move operation may overwrite lines which the user wanted to keep.

The entire source range is moved to the new location before it is deleted.
If the source and target ranges overlap, each line is moved and deleted separately.

Current increment value and line number

@MOVE does not change the current increment value. The operand inc simply determines the increment used between the moved records. It does not refer to the current increment value.

If the specified increment value (inc) is too large, the move operation may overwrite existing lines in the target range.

The current line number is changed in L mode only if a line with a number greater than the currently highest line number is created.

Example

```

1.00 THIS LINE IS NOT MOVED.....
2.00 LINE 2 AND LINE 3.....
3.00 AND LINE 4 ARE MOVED.....
4.00 SEVERAL TIMES.....
5.00 .....

set 90:this line is never overwritten.....0001.00:001(0)

```

This creates a new line 90.

```

1.00 THIS LINE IS NOT MOVED.....
2.00 LINE 2 AND LINE 3.....
3.00 AND LINE 4 ARE MOVED.....
4.00 SEVERAL TIMES.....
90.00 THIS LINE IS NEVER OVERWRITTEN.....
91.00 .....

move 2-4 to 20.....0001.00:001(0)

```

Lines 2 to 4 are to be moved to the range starting at line 20, using the implicit increment value of 1.

```

1.00 THIS LINE IS NOT MOVED.....
20.00 LINE 2 AND LINE 3.....
21.00 AND LINE 4 ARE MOVED.....
22.00 SEVERAL TIMES.....
90.00 THIS LINE IS NEVER OVERWRITTEN.....
91.00 .....

move 20-22 to 100 (5).....0001.00:001(0)

```

Lines 20, 21 and 22 have been created with the implicit increment value of 1 and lines 2, 3 and 4 have been deleted.

Lines 20-22 are now to be moved to lines 100, 105 and 110.

```

1.00 THIS LINE IS NOT MOVED.....
90.00 THIS LINE IS NEVER OVERWRITTEN.....
100.00 LINE 2 AND LINE 3.....
105.00 AND LINE 4 ARE MOVED.....
110.00 SEVERAL TIMES.....
111.00 .....

move 100-.$ to 82(5) : 89.....0001.00:001(0)

```

The range from line 100 to the end of the file (100-.\$) is to be moved to the range starting at line 82, using the explicit increment value of 5. Specifying the upper limit value of 89 ensures that line 90 is not overwritten.

```
1.00 THIS LINE IS NOT MOVED.....  
82.00 LINE 2 AND LINE 3.....  
87.00 AND LINE 4 ARE MOVED.....  
90.00 THIS LINE IS NEVER OVERWRITTEN.....  
110.00 SEVERAL TIMES.....  
111.00 .....
```

Line 110 was not moved, since the new line would have exceeded the specified limit value.

@NOTE Place comment in EDT procedure

@NOTE is used to place comments in EDT procedures; it does not cause any further action when executed.

Operation	Operands	L mode / @PROC
@NOTE	[comment]	

comment

Comment, freely selectable text.

@ON Process file with search string

@ON examines a specified line range for the presence of the specified search string. There are 11 formats of the @ON statement for processing the data lines in which the search string is found. The various formats execute the following actions if the string is found:

Output and marking

- Format 1: Outputs the line which contains the search string on the screen (in interactive mode) or on the printer (in batch mode); see [page 342ff.](#)
- Format 2: Displays, on the screen, the number of the column in which the search string begins or the length of each line in the specified line range; see [page 346ff.](#)
- Format 3: Checks if the search string exists in the specified range and, if so, records the number of the first line in which the string was found (the first hit); see [page 350ff.](#)
- Format 4: Marks the line containing the search string with a record mark (preparation for format 5); see [page 352ff.](#)

Copying with or without record marks

- Format 5: Copies the lines with the specified record mark; see [page 355ff.](#)
- Format 6: Copies the lines which contain the specified search string; see [page 358ff.](#)

Replacing and inserting strings

- Format 7: Replaces the string identified by the search string with another string; see [page 361ff.](#)
- Format 8: Replaces or inserts a string before or after the search string; see [page 364ff.](#)

Deleting strings

- Format 9: Deletes the specified search string from the text; see [page 368ff.](#)
- Format 10: Deletes the text between the beginning or end of the line and the specified search string; see [page 371ff.](#)
- Format 11: Deletes the line containing the specified search string; see [page 373ff.](#)

Specification of the search string in @ON

The search string may be specified in the @ON statement:

- directly, in the form of a string enclosed in single or double quotes, or
- indirectly, in the form of a line number, a line number variable or a string variable.

The simplest search string is a sequence of constants (e.g. 'ABC'). Any substring with the same text value found in the search object satisfies the search criterion.

Search string taking account of uppercase/lowercase notation

In the @SEARCH-OPTION statement it is possible to specify whether the search for specific strings is to distinguish between uppercase and lowercase letters. This setting for the search string "search" in @ON is valid for formats 1 to 4 and 6 to 8 (for a description of format 5 see [page 355ff](#)).

Example

@SEARCH-OPTION CASELESS-SEARCH	sets the default value OFF for search type CASELESS-SEARCH, i.e. a distinction is made between uppercase and lowercase notation; the search finds every word beginning with s or S.
@ON & FIND PATTERN "s*"	
@SEARCH-OPTION CASELESS-SEARCH	sets the default value OFF for search type CASELESS-SEARCH, i.e. a distinction is made between uppercase and lowercase notation.
@ON & C'SEEK' TO 'SEEK'	All strings 'seek' in all possible uppercase and lowercase variations are converted to SEEK.

Use of wildcards in the search string

In addition to constants, variables (known as wildcards) can be specified. The following two wildcards are used:

- asterisk (Default value *); this replaces a string of any length, even an empty one. The search is satisfied by the shortest possible substring in the line that is searched. Two or more adjacent asterisks are handled in the same way as a single asterisk, e.g.: 'ABC**F' is equivalent to 'ABC*F'.
- slash (Default value /); this replaces precisely one character.

If the keyword PATTERN is specified, the wildcards are interpreted as variable characters and pattern matching is performed (see the example for @ON, format 6).

If there is no keyword PATTERN, the wildcards are treated as simple constants.

Example

on & print 'AB*C'	displays all the lines containing exactly the string AB*C.
on & print pattern 'AB*C'	displays those lines containing the strings ABC, ABXC, ABCDEFG, ABXXXXXXC etc.

More than one wildcard may be used in each search string. A search string consisting exclusively of wildcards is also permitted. The wildcards can also be redefined by means of @SYMBOLS.

Negative search

If the keyword NOT is specified, those records which do not contain the search string are selected (see the example for @ON, format 1).

Indirect specification of the search string

- The search string is stored in a line whose line number must be specified.
- The search string is stored in a line whose number is stored in a line number variable (#L0 to #L20). This line number variable is then specified in the @ON statement.
- The search string (e.g. 'ABC') is assigned (see @SET) to a string variable (#S1 to #S20). This string variable is then specified in the @ON statement.

```
@SET #S0 = 'AB*C//D'
@ON & PRINT PATTERN #S0
```

When specified indirectly, the search string is always treated as if it were enclosed in single quotes. Double quotes cannot be used.

Significance of the search string delimiters

The left-hand delimiter of the search string is			
Single quotes		Double quotes	
The right-hand delimiter of the search string is		The right-hand delimiter of the search string is	
Single quotes	Double quotes	Single quotes	Double quotes
EDT detects a hit if	EDT detects a hit if	EDT detects a hit if	EDT detects a hit if
<ul style="list-style-type: none"> the search string exists 	<ul style="list-style-type: none"> the search string exists and if is followed by a text delimiter or is at the end of the line 	<ul style="list-style-type: none"> the search string exists and it is preceded by a text delimiter or begins at the beginning of the line 	<ul style="list-style-type: none"> the search string exists and it is preceded by a text delimiter or begins at the beginning of the line if is followed by a text delimiter or is at the end of the line
<p>Example:</p> <ol style="list-style-type: none"> ABCD A,BCD ABC,D A,BC,D <p>@ON &... 'BC' ... detects hits in all 4 lines.</p>	<p>Example:</p> <ol style="list-style-type: none"> ABCD A,BCD ABC,D A,BC,D <p>@ON &... 'BC'... detects hits only in lines 3 and 4.</p>	<p>Example:</p> <ol style="list-style-type: none"> ABCD A,BCD ABC,D A,BC,D <p>@ON &... "BC" ... detects hits only in lines 2 and 4.</p>	<p>Example:</p> <ol style="list-style-type: none"> ABCD A,BCD ABC,D A,BC,D <p>@ON &... "BC"... detects a hit only in line 4.</p>

If @QUOTE is used to define some other character to be used instead of single quotes, the above rules apply to this new character.

When an EDT session is started, the following text delimiters are available:

Blank (X'40') and +.!*());-/,?:'="

The set of text delimiters can be modified by means of @DELIMIT.

Use of the delimiters in a search string with wildcards

The search string can be enclosed between:

single quotes	single quotes and double quotes	double quotes
@ON & P PATTERN 'ABC*'	@ON & P PATTERN 'ABC*' @ON & P PATTERN "ABC*"	@ON & P PATTERN "ABC*"

If the wildcard asterisk occurs next to a double quote in the search string, the hit string extends to the next text delimiter. If there is no text delimiter, the hit string contains the rest of the line.

If the wildcard asterisk is next to a single quote, the hit string contains the shortest possible character string.

Example

Line contains xxx_abcd_yyy

Search string	Hit string
'abc*'	abcd (because _ is the next text delimiter)
'abc*'	abc (because the shortest possible substring is used)

If a search string is to contain the text delimiters, " must be entered for ', and "" must be entered for " .

Example

'This is a ""random"" string.'
(The following string is sought: This is a "random" string.)

How EDT searches for the specified string

The order in which EDT searches the lines for a string and the type of search executed depend on the following operands:

range	The line range (or several subranges) to be searched
domain	The column range to be searched
F	EDT stops searching after the first hit
ALL	EDT finds all hits in a line
int	EDT searches for the “int”th hit in a line
R	EDT searches the lines from right to left (reverse)

By default, the lines are searched from left to right. EDT searches the lines in the order in which they are specified in the “range” operand. The following diagrams show how the actions of EDT depend on the operands F, range, int and ALL.

Is operand F specified?		
Yes	No	
Are there several line ranges (or line numbers) separated by commas, specified for range?		Each line of the specified range(s) is searched.
Yes	No	
If EDT finds the specified search string, it stops searching the current line range: the remaining lines in this range are not searched. EDT continues the search in the next range specified (if any).	If EDT finds the specified search string, it stops searching. The remaining lines are not searched.	

Is operand int specified?			
Yes		No	
EDT signals a hit only when it finds the "int"th occurrence of the search string in the line being examined. Example: 1. AAAAA @ON 1 'AA',3 will detect a hit in columns 3 and 4.		Is operand ALL specified?	
		Yes	No
Is operand ALL specified?		When EDT finds the search string, it executes the action specified by @ON. After this, it continues the search at the column which originally followed this search string.	EDT stops examining a line as soon as it finds the first occurrence of the search string.
Yes	No		
When EDT finds the search string, it executes the action specified by @ON. After this, it continues the search at the column which originally followed this search string. Any further occurrence of the search string which EDT finds in the line is regarded as a hit.	EDT stops examining a line as soon as it finds the first occurrence of the search string.		

Recording a hit

EDT records whether or not it has found a hit:

Checking for hits via @IF

@IF (see @IF, format 3) can be used to check if the last @ON statement executed has found a hit.

The message NO MATCH IN RANGE can be interrogated only if the procedure was called by means of @DO...PRINT, since the EDT error switch is set only if the message is actually displayed on the screen.

Recording the hit location

- The number of the line in which EDT found the first hit is recorded in line number variable #L0 and under the line number symbol ?. If the hit is found in a string variable, the current values of #L0 and ? remain unchanged.
- The number of the column in which the first occurrence of the search string begins is placed in integer variable #I0 and the number of the column in which it ends is placed in integer variable #I1. This also applies to a hit detected within a string variable.

Example

The line contains XXX_ABCD_YYY

Search string	Hit string	Contents of I0 or #I1
'ABC*'	ABCD	#I0 = 5; #I1 = 8
'ABC*'	ABC	#I0 = 5; #I1 = 7
'*BCD'	ABCD	#I0 = 5; #I1 = 8
'*BCD'	BCD	#I0 = 6; #I1 = 8

In the case of a hit in L mode, the hit can be displayed on the screen by means of @PRINT #L0:#I0-#I1:. If no hit is found, the values of ?, #L0, #I0 and #I1 remain unchanged.

Recording a hit in a negative search

For the first record in which the search string does not occur, the start position of the checked column range is stored in integer variable #I0 and the end position is stored in integer variable #I1.

If the end position of the column range exceeds the record length, the record length is stored in #I1.

@ON (Format 1) Display lines containing the search string

This format of @ON causes EDT to output the contents of each line in which the search string was found. The output is sent to the screen (SYSOUT) in interactive mode or to the printer (SYSLST) in batch mode.

Operation	Operands	F mode / L mode
@ON	range [:domain] PRINT [ALL] [F] [R] [NOT] [PATTERN] search [,int] [S] [N] [E]	

- range A line range, specified as:
- one or more line numbers, separated by comma (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- domain A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
 If the first column number is greater than the line length, this line is ignored.
 The second column number
- must not be less than the first column number
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL Only effective if the E operand is specified. All hits in a line are highlighted.
- F Only the first hit line is output. If the E operand is specified, either the first or all hits in the first hit line are highlighted.
- R The lines are searched from right to left. Normally, they are searched from left to right.
 If R is specified, PRINT must be specified at least as PR.
- NOT A hit is recognized if the search string is not contained in the specified column range of a line (negative search).

PATTERN	The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.
search	The search string, specified either: <ul style="list-style-type: none"> – directly, in the form of a string enclosed in single quotes, or – indirectly, in the form of a line number, a line number variable or a string variable (in each case with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
int	The “int”th occurrence of the specified search string in a line is to be regarded as a hit.
S	Suppresses the empty line which otherwise precedes the first line to be output.
N	The hit lines are to be output without line numbers.
E	Highlights the search string in the output to the screen. If the ALL operand is also specified, all hits in the hit lines are highlighted, otherwise only the first occurrence or the int-th (if the int operand is specified) from the left or from the right (if the R operand is specified).

Example

```

1.00 ALL LINES IN.....
2.00 WHICH A HIT OCCURS.....
3.00 ARE TO BE OUTPUT;.....
4.00 IF THERE IS NO.....
5.00 HIT IN THE LINE RANGE.....
6.00 NOTHING IS OUTPUT.....
7.00 .....

```

```
on & print f 'hit'.....0001.00:001(1)
```

The first line containing the string 'HIT' is to be output.

```

2.0000 WHICH A HIT OCCURS
PLEASE ACKNOWLEDGE

```

```

1.00 ALL LINES IN.....
2.00 WHICH A HIT OCCURS.....
3.00 ARE TO BE OUTPUT;.....
4.00 IF THERE IS NO.....
5.00 HIT IN THE LINE RANGE.....
6.00 NOTHING IS OUTPUT.....
7.00 .....

on & print 'hit'.....0001.00:001(1)

```

All lines which contain the string 'HIT' are to be output.

```

2.0000 WHICH A HIT OCCURS
5.0000 HIT IN THE LINERANGE
PLEASE ACKNOWLEDGE

create 100: #10 :#i0-#i1:.....0001.00:001(1)

```

If there are several hits, the values of line number variable #L0 and of integer variables #I0 and #I1 apply to the first hit which was found, i.e. to 'HIT' in line 2.

```

1.00 ALL LINES IN.....
2.00 WHICH A HIT OCCURS.....
3.00 ARE TO BE OUTPUT;.....
4.00 IF THERE IS NO.....
5.00 HIT IN THE LINE RANGE.....
6.00 NOTHING IS OUTPUT.....
100.00 HIT.....
101.00 .....

on &:2 print 'hit'.....0001.00:001(1)

```

All lines in which the string 'HIT' occurs in or after column 2 are to be output.


```

2.0000 WHICH A HIT OCCURS
PLEASE ACKNOWLEDGE

```

```

1.00 ALL LINES IN.....
2.00 WHICH A HIT OCCURS.....
3.00 ARE TO BE OUTPUT;.....
4.00 IF THERE IS NO.....
5.00 HIT IN THE LINE RANGE.....
6.00 NOTHING IS OUTPUT.....
100.00 HIT.....
101.00 .....

```

```

on & print 'I',3.....0001.00:001(1)

```

The lines in which the string 'I' appears at least three times are to be output.

```

5.0000 HIT IN THE LINE RANGE,
PLEASE ACKNOWLEDGE

```

```

1.00 ALL LINES IN.....
2.00 WHICH A HIT OCCURS.....
3.00 ARE TO BE OUTPUT;.....
4.00 IF THERE IS NO.....
5.00 HIT IN THE LINE RANGE.....
6.00 NOTHING IS OUTPUT.....
100.00 HIT.....
101.00 .....

```

```

on & print not 'hit'.....0001.00:001(1)

```

All lines in which the string 'HIT' does not occur are to be output.

```

1.0000 ALL LINES IN
3.0000 ARE TO BE OUTPUT;
4.0000 IF THERE IS NO
6.0000 NOTHING IS OUTPUT.
PLEASE ACKNOWLEDGE

```

@ON (Format 2) Display the starting column of the search string

This format of @ON causes EDT to display, on the screen, the line numbers and numbers of the columns in which the occurrences of the search string begin. If “search” is omitted, EDT displays the line number and length of each line in the specified line range.

Operation	Operands	F mode / L mode
@ON	range [:domain] COLUMN [ALL] [F] [R] [PATTERN] [search [,int]]	

- range A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- domain A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
 If the first column number is greater than the line length, this line is ignored.
 The second column number
- must not be less than the first column number,
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL After each hit, EDT is to continue examining the remainder of the line.
- F Only the first hit of each specified line range is displayed.
- If neither ALL nor F is specified, the first hit in each line is displayed.
- R The lines are to be searched from right to left instead of in the default direction (left to right).
- PATTERN The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.

- search The search string, specified either:
- directly, in the form of a string enclosed in single quotes, or
 - indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
- int The “int”th occurrence of the specified search string in a line is to be regarded as a hit.

Example

```

1.00 HOW LONG IS LINE 1 ?.....
2.00 AND LINE 2 ?.....
3.00 WHO KNOWS THE LENGTH OF LINE 3 ?.....
4.00 .....

on & column.....0001.00:001(1)

```

The length of each line is to be displayed.

```

1.0000 020
2.0000 012
3.0000 032
PLEASE ACKNOWLEDGE

```

```

1.00 HOW LONG IS LINE 1 ?.....
2.00 AND LINE 2 ?.....
3.00 WHO KNOWS THE LENGTH OF LINE 3 ?.....
4.00 .....

on 3 column r 'e '.....0001.00:001(1)

```

The column number at which the string 'E' occurs for the first time in line 3 (searching from the right) is to be displayed.

```

1.0000 028
PLEASE ACKNOWLEDGE

```

```

1.00 HOW LONG IS LINE 1 ?.....
2.00 AND LINE 2 ?.....
3.00 WHO KNOWS THE LENGTH OF LINE 3 ?.....
4.00 .....

on 3 column all 'e '.....0001.00:001(1)

```

The numbers of all columns in which the string 'E' occurs in line 3 are to be displayed.

```

1.0000 013 028
PLEASE ACKNOWLEDGE

```

```

1.00 HOW LONG IS LINE 1 ?.....
2.00 AND LINE 2 ?.....
3.00 WHO KNOWS THE LENGTH OF LINE 3 ?.....

on & column 'o',2.....0001.00:001(1)

```

For all lines containing the string 'O' at least twice, the column number of the hit (second occurrence of the search criterion) is to be displayed.

```

1.0000 006
3.0000 007
PLEASE ACKNOWLEDGE

```

```

1.00 HOW LONG IS LINE 1 ?.....
2.00 AND LINE 2 ?.....
3.00 WHO KNOWS THE LENGTH OF LINE 3 ?.....
4.00 .....

```

```

on & column all 'e'.....0001.00:001(1)

```

The numbers of all lines and columns in which the string 'E' occurs are to be displayed.

```

1.0000 016
2.0000 008
3.0000 013 016 028
PLEASE ACKNOWLEDGE

```

@ON (Format 3) Find the line number of the first hit

This format of @ON determines whether the specified search string occurs within the specified range and, if so, where it occurs for the first time. The line number of the hit is placed in #L0 and the column numbers in #I0 and #I1.

Operation	Operands	L mode / @PROC
@ON	range [:domain] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int]	

- range A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- domain A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
 If the first column number is greater than the line length, this line is ignored.
 The second column number
- must not be less than the first column number,
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL After each hit, EDT is to continue examining the remainder of the line.
- F Examination of the line range is terminated after the first hit has been found. The remaining lines in the range are not examined.
- Specification of ALL or F is permitted in this format of @ON, but is meaningless, since only the first hit is recorded.
- R The lines are to be searched from right to left instead of in the default direction (left to right).
- NOT A hit is recognized if the search string is not contained in the specified column range of a line (negative search).

PATTERN	The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.
search	The search string, specified either: <ul style="list-style-type: none"> – directly, in the form of a string enclosed in single quotes, or – indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
int	The “nits occurrence of the specified search string in a line is to be regarded as a hit.

Example

```

1.      ***** ----- (01)
2.      E1**E2**E3**E4**E5**E6**
3.      @ON & FIND 'E',4 ----- (02)
3.      @PRINT #L0:#I0 ----- (03)
2.0000 E4**E5**E6**
3.
```

- (01) Two lines are created in the virtual file.
- (02) The fourth occurrence of 'E' in each line is to be found. EDT shows no external reaction; however, if there is a hit, it places the number of the hit line in #L0, the starting column of the hit in #I0 and the final column of the hit in #I1.
- (03) This prints the first hit line, starting at the first column of the hit.

@ON (Format 4) Mark records containing the search string

This format causes all records which contain the search string to be marked with the specified record mark (m). In F mode the work window is positioned to the first record containing a hit.

Existing record marks (such as those set by a previous @ON statement) remain unchanged. They can, if desired, be deleted by means of @DELETE MARK.

The resulting record marks can be used for copying (@ON format 5) or for positioning within the work file (see “+/- Position within work file”, page 120ff, format 2).

Operation	Operands	F mode / L mode
@ON	range* [:domain] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int] MARK [m]	

- range* A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.
- domain A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
 If the first column number is greater than the line length, this line is ignored.
 The second column number
- must not be less than the first column number
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL After each hit, EDT is to continue examining the remainder of the line. This operand may be specified, but is meaningless, since a record can be marked only once.
- F Examination of each line range is to be terminated after the first hit.

R	The lines are searched from right to left. Normally they are searched from left to right.
NOT	A hit is recognized if the search string is not contained in the specified column range of a line (negative search).
PATTERN	The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.
search	The search string, specified either: <ul style="list-style-type: none">– directly, in the form of a string enclosed in single quotes, or– indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
int	The “int”th occurrence of the specified search string in a line is to be regarded as a hit.
MARK m	The records are to be marked with record mark m (1, ..., 9). MARK m does not need to be specified in F mode: the records are then marked with mark number 1.



If the statement is issued for a file opened by means of @OPEN, no records are marked; the work window is simply positioned to the first record containing a hit. Explicit specification of MARK m is rejected with an error message.

Example

```

1.00 BERGER THOMAS 10, HIGH RD. DURHAM.....
2.00 DUCK DONALD 8, WALT RD. DISNEYLAND.....
3.00 GREEN JENNIFER 16, LOW ST. POUGHKEEPSIE.....
4.00 STUBBS LARRY P.O. BOX 99 MUNICH.....
5.00 HOPPER MANUELA 3, POST ST. GRANTHAM.....
6.00 .....

on & find 'str.' mark 2.....0001.00:001(1)

```

The records containing the string 'ST' are to be marked with mark number 2. EDT will automatically position the file to the first record containing a hit.

```

3.00 GREEN JENNIFER 16, LOW ST. POUGHKEEPSIE.....
4.00 STUBBS LARRY P.O. BOX 99 MUNICH.....
5.00 HOPPER MANUELA 3, POST ST. GRANTHAM.....
6.00 .....

+(2).....0003.00:001(1)

```

The work window is positioned to line 3, since this contains the first hit.

+(2) scrolls to the next record marked with 2.

```

4.00 STUBBS LARRY P.O. BOX 99 MUNICH.....
5.00 HOPPER MANUELA 3, POST ST. GRANTHAM.....
6.00 .....

```

@ON (Format 5) Copy marked records

All records with the specified record mark are copied into the specified work file.

A work file currently being executed as an EDT procedure (see @DO), i.e. an active work file, cannot be used as the output file.

Operation	Operands	F mode / L mode
@ON	range* [:domain] FIND [ALL] [F] [NOT] MARK m [COPY TO] (procno) [KEEP] [OLD]	

Only the marked records are checked to see if they contain the mark m.

range*	<p>A line range, specified as:</p> <ul style="list-style-type: none"> – one or more line numbers, separated by commas (e.g. 4,6,15) – one or more line ranges, separated by commas (e.g. 5-10,17-19) – a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30). <p>The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.</p>
domain	<p>Column range.</p> <p>This operand may be specified, but it is meaningless since EDT searches only for record marks.</p>
ALL	<p>After it has found a hit, EDT is to examine the remainder of the line.</p> <p>This operand is permitted, but it is meaningless, since a line is copied only once.</p>
F	<p>Only the first record containing the specified mark in each specified line range is copied.</p> <p>If F is specified together with NOT, the first marked record not containing the mark m in each specified line range is copied.</p>
NOT	<p>All marked lines not containing the mark m are copied.</p>
MARK m	<p>The number (1, ..., 9) of the record mark.</p>
procno	<p>The number (0-22) of the work file into which the records are to be copied. procno must not be the current work file. If hits are found and OLD is not specified, the contents of procno are deleted before the new lines are copied into it. If no hits are found, the contents of procno remain unchanged. An active work file (see @DO) cannot be specified as the output file.</p>

- KEEP The marked lines are to retain their line numbers as they are copied. If KEEP is omitted, EDT creates the lines in the output file, starting with the current line and using the current increment value.
- OLD The contents of the target file procno are not deleted before copying. Any existing lines in procno with the same line numbers are overwritten. If hits are found and OLD is omitted, the contents of procno are deleted before the new lines are copied to it.

If no record is marked or no record with the specified mark is found, EDT issues the message: % EDT0901 NO MATCH IN RANGE



If this format of @ON is entered in work file 0 and an ISAM file has been opened in real mode by means of @OPEN, the statement is rejected with the error message: % EDT4935 MAIN FILE OPENED REAL.

Example 1

```

1.00 BERGER THOMAS 10, HIGH RD. DONCASTER.....
2.00 DUCK DONALD 8, WALT ST. DISNEYLAND.....
3.00 GREEN JENNIFER 16, LOW ST. POUGHKEEPSIE.....
4.00 HOPPER LARRY P.O. BOX 99 MUNICH.....
5.00 STUBBS MANUELA 3, POST ST. GRANTHAM.....
6.00 .....

on 1-5 find mark 2 copy to (3) keep ; 3.....0001.00:001(1)

```

Line range 1 to 5 is to be searched for record mark 2 and all records with this mark are to be copied into work file 3, retaining their original line numbers. EDT is then to switch to work file 3.

```

3.00 GREEN   JENNIFER 16, LOW ST.   POUGHKEEPSIE.....
4.00 HOPPER  LARRY     P.O. BOX 99   MUNICH.....
5.00 .....
.....0003.00:001(3)

```

Example 2

All the lines which are not marked are to be copied. As a first step, all lines must be marked with a record mark (e.g. 9) that has not yet been assigned (@ON, format 4). Then all those lines which do not have the record mark (e.g. 1 to 3) are copied (@ON, format 5).

```

@ON & FIND PATTERN '*' MARK 9
@ON & FIND NOT MARK 1 COPY TO (1) KEEP
@PROC 1
@ON & FIND NOT MARK 2 COPY TO (2) KEEP
@PROC 2
@ON & FIND NOT MARK 3 COPY TO (3) KEEP
@PROC 3

```

Work file 1 contains all lines which do not have record mark 1.

Work file 2 contains all lines which do not have record mark 1 or 2.

Work file 3 contains all lines which do not have record mark 1 or 2 or 3.

@ON (Format 6) Copy records containing the search string

All records with the specified search string are copied into the specified work file. If hits are found, the contents of the target file procno are deleted before copying by default. If no hits are found or OLD is specified, the contents remain unchanged.

A work file currently being executed as an EDT procedure (see @DO), i.e. an active work file, cannot be used as the output file.

Operation	Operands	F mode / L mode
@ON	range* [:domain] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int] [COPY [TO]] (procno) [KEEP] [OLD]	

- range* A line range, specified as:

 - one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).

The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.
- domain A column range, specified as:

 - a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).

If only one column number is specified, the line is searched from this column to the end of the line.
If the first column number is greater than the line length, this line is ignored.
The second column number

 - must not be less than the first column number,
 - may be greater than the actual line length.

If no column range is specified, the entire line is searched.
- ALL After each hit, EDT is to continue examining the remainder of the line. This operand may be specified, but it is meaningless, since a record is copied only once.
- F In each specified line range, only the first search string found is copied.
- R The lines are to be searched from right to left. Normally, they are searched from left to right.

NOT	A hit is recognized if the search string is not contained in the specified line range of a line (negative search).
PATTERN	The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.
search	The search string, specified either: <ul style="list-style-type: none"> – directly, in the form of a string enclosed in single quotes, or – indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
int	The “int”th occurrence of the specified search string in a line is to be regarded as a hit.
procno	The number (0-22) of the work file into which the records are to be copied. procno must not be the current work file. If hits are found and OLD is not specified, the contents of procno are deleted before the new lines are copied into it. If no hits are found, the contents of procno remain unchanged. An active work file (see @DO) cannot be specified as the output file.
KEEP	The marked lines are to retain their line numbers as they are copied. If KEEP is omitted, EDT creates the lines in the output file, starting with the current line and using the current increment value.
OLD	The contents of the target file procno are not deleted before copying. Any existing lines in procno with the same line numbers are overwritten. If hits are found and OLD is omitted, the contents of procno are deleted before the new lines are copied to it.

Example

```

1.00 BERGER THOMAS 10, HIGH RD. DERBY.....
2.00 DEACON DENNIS 8, MOON RD. DERBY.....
3.00 GREEN JENNIFER 16, LOW ST. POUGHKEEPSIE.....
4.00 HOPPER LARRY 44, HOPE ST. BRADFORD.....
5.00 SMITH DORIS 3, POST RD. DERBY.....
6.00 .....

on & find 'str.' copy to (5) ; 5.....0001.00:001(1)

```

All records containing the string 'ST.' are to be copied into work file 5 and renumbered there. EDT is then to switch to work file 5.

```

1.00 GREEN   JENNIFER  16, LOW ST.   POUGHKEEPSIE.....
2.00 HOPPER  LARRY     44, HOPE ST.  BRADFORD.....
3.00 .....
.....
1.....0001.00:001(5)

```

EDT switches to work file 1.

```

1.00 BERGER  THOMAS    10, HIGH RD.  DERBY.....
2.00 DEACON  DENNIS    8, MOON RD.   DERBY.....
3.00 GREEN   JENNIFER  16, LOW ST.   POUGHKEEPSIE.....
4.00 HOPPER  LARRY     44, HOPE ST.  BRADFORD.....
5.00 SMITH   DORIS     3, POST RD.   DERBY.....
6.00 .....
.....
on & :10-20: find pattern 'm*a' copy to (6) ; 6.....0001.00:001(1)

```

All records of people whose first names begin with “D” and end with “S” are copied into work file 6. EDT then switches to work file 6.

```

1.00 DEACON  DENNIS    8, MOON RD.   DERBY.....
2.00 SMITH   DORIS     3, POST RD.   DERBY.....
3.00 .....
.....
.....0001.00:001(6)

```


@ON (Format 7) Replace the search string

This format of @ON causes the search string to be replaced by another string in the event of a hit. The number of hits and the number of hit lines can be written into integer variables (V switch).

Operation	Operands	F mode / L mode
@ON	range [:domain] CHANGE [ALL] [F] [R] [PATTERN] search [,int] [TO] string [V]	

- range A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- domain A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
If the first column number is greater than the line length, this line is ignored.
The second column number
- must not be less than the first column number
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL After each hit, EDT is to continue examining the remainder of the line after replacing the search string with the new string.
- F The search and replace of each specified line range is terminated after the first hit line.
If neither ALL nor F is specified, the first occurrence of the search string in each line will be replaced.
- R The lines are to be searched from right to left instead of in the default direction (left to right).
- PATTERN The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.

- search The search string, specified either:
- directly, in the form of a string enclosed in single quotes, or
 - indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
- int The “int”th occurrence of the specified search string in a line is to be regarded as a hit. A value between 1 and 256 may be specified for int; the default value is 1.
- string A character string, which may be specified:
- explicitly, enclosed in single quotes, or
 - implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).
- “string” replaces the search string. If an empty string (”) is specified, the search string is deleted.
- V Only effective if the ALL operand is also specified. The number of hit lines is written into integer variable #I2 and the number of hits into integer variable #I3.
#I2 and #I3 remain unchanged if ALL or V is not specified.

Example

```

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7-----
1.00 ABCDEFGHIJKLMNOPQRSTUVWXYZ.....
2.00 WHO IS AS IBBORN AS A MULE ?.....
3.00 .....

on 2 change 'i',3 to 1:19-21 ; on 2 change 'is' to ''.....0001.00:001(1)

```

The first @ON is to search for the third occurrence of the character 'I' in line 2 and replace it with the string from columns 19 to 21 of line 1, i.e. 'STU'.

The second @ON is to search for the first occurrence of the string 'IS' in line 2 and replace it with the empty string, i.e. to delete it (this is an alternative to @ON, format 9).

```

1.00 ABCDEFGHIJKLMNOPQRSTUVWXYZ.....
2.00 WHO IS AS STUBBORN AS A MULE ?.....
3.00 .....

```

@ON (Format 8) Replace or insert before or after the search string

This format of @ON provides two possibilities for modifying the contents of a line. The specified new string

- either CHANGES the text between the beginning of the line and the search string (PREFIX) or the text between the search string and the end of the line (SUFFIX), or
- is INSERTed before (PREFIX) or after (SUFFIX) the search string.

Operation	Operands	F mode / L mode
@ON	range [:domain] FIND [ALL] [F] [R] [PATTERN] search [,int] { CHANGE } { PREFIX } string { INSERT } { SUFFIX }	

- range A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- domain A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
 If the first column number is greater than the line length, this line is ignored.
 The second column number
- must not be less than the first column number,
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL After each hit, EDT is to continue examining the remainder of the line after replacing the text before or after the search string or inserting the new string before or after the search string, as appropriate.
- F Examination of each line range specified is to be terminated after the first hit.

If neither ALL nor F is specified, the first hit in each line is processed.

R	The lines are to be searched from right to left instead of in the default direction (left to right).
PATTERN	The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.
search	The search string, specified either: <ul style="list-style-type: none">– directly, in the form of a string enclosed in single quotes, or– indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
int	The “int”th occurrence of the specified search string in a line is to be regarded as a hit.
CHANGE	The text between the beginning of the line and the search string or between the search string and the end of the line is to be replaced by the specified new string.
INSERT	The specified new string is to be inserted before or after the search string.
PREFIX	The character string specified for “string” replaces the text between the beginning of the line and the search string or is inserted before the search string.
SUFFIX	The character string specified for “string” replaces the text between the search string and the end of the line or is inserted after the search string.
string	A character string, which may be specified: <ul style="list-style-type: none">– explicitly, enclosed in single quotes, or– implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required). “string” replaces the text before or after the search string or is inserted before or after the search string, as appropriate. If PREFIX or SUFFIX is written in full and a character string is specified for “string”, there must be a blank between PREFIX or SUFFIX and the single quote.

Example 1

```

23.00 .....
fstat '$user2.xml.' .....0001.00:001(1)

```

All shareable files under the user ID USER2 whose names begin with the partially qualified name 'XMPL.' are to be listed.

```

1.00 XMPL.1.....
2.00 XMPL.2.....
3.00 XMPL.3.....
4.00 XMPL.4.....
5.00 .....

on & find 'xml.' insert prefix '@read '$ser2.' .....0001.00:001(9)

```

The character string '@read '\$user2.' is to be prefixed to each partially qualified name 'XMPL.'.

```

1.00 @READ '$USER2.XMPL.1.....
2.00 @READ '$USER2.XMPL.2.....
3.00 @READ '$USER2.XMPL.3.....
4.00 @READ '$USER2.XMPL.4.....
5.00 .....

1 ; do 9.....0001.00:001(9)

```

The four files 'XMPL.1' to 'XMPL.4' are read into work file 1, one after the other.

Example 2

```

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7--
1.00 A11B11C11D11E11F11G11H11.....
2.00 A1111B1111C1111D1111E1111.....
3.00 .....

on 1-2 find r '11',4 insert suffix '++++'.....0001.00:001(1)

```

In the line range 1 to 2, a search from right to left for the fourth occurrence of the string '11' is to be effected and ++++ is to be inserted after the hit.

```

-----+-----1-----+-----2-----+-----3-----+-----4-----+-----5-----+-----6-----+-----7--
1.00 A11B11C11D11E11++++F11G11H11.....
2.00 A1111B1111C1111D1111++++E1111.....
3.00 .....

on &:4 find '111',3 change suffix '####'.....0001.00:001(1)

```

In line 1, the search string occurred for the fourth time (counted from the right) in columns 14-15. In line 2, the search string occurred for the first time in columns 24-25, for the second time in columns 23-24, for the third time in columns 22-23, and for the fourth time (hit) in columns 19-20. The string '++++' was inserted after the hit.

Now, a search is to be made in the entire work file, starting at column 4, for the third occurrence of the string '111'. If a hit occurs, the subsequent text is to be replaced with ####.

```

1.00 A11B11C11D11E11++++F11G11H11.....
2.00 A1111B1111C111####.....
3.00 .....

```

In line 1, the search string was not found.

In line 2, the search string occurred for the first time (counting from column 4) in columns 7-9, for the second time in columns 8-10, and for the third time (hit) in columns 12-14. The line portion after the hit was replaced with the string '####'.

@ON (Format 9) Delete the search string

In the case of a hit, this format of @ON deletes the search string, leaving the remainder of the line unchanged.

Operation	Operands	F mode / L mode
@ON	range [:domain] DELETE [ALL] [F] [R] [PATTERN] search [,int]	

- range** A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- domain** A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
 If the first column number is greater than the line length, this line is ignored.
 The second column number
- must not be less than the first column number
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL** After each hit, EDT is to continue examining the remainder of the line after deleting the search string it has found.
- F** Examination of each line range specified is to be terminated after the first hit.
- If neither ALL nor F is specified, the first hit in each line is deleted.
- R** The lines are to be searched from right to left instead of in the default direction (left to right).
- PATTERN** The characters currently specified in "search" for * (asterisk) and / (slash) are interpreted as wildcards.

- search The search string, specified either:
- directly, in the form of a string enclosed in single quotes, or
 - indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
- int The “int”th occurrence of the specified search string in a line is to be regarded as a hit.

Example

```

1.00 XXXYYYYZZ *** XXXYYYYZZ ### XXXYYYYZZ %%%.
2.00 AAA XXXYYYYZZ BBB XXXYYYYZZ CCC XXXYYYYZZ.
3.00 .....

on & delete f r 'xxxxyyzzz'.....0001.00:001(1)

```

Each line in the work file is to be searched from right to left for the search string 'XXXXYYZZZ'. The first occurrence of this string is to be deleted and the search is then to be terminated.

```

1.00 XXXYYYYZZ *** XXXYYYYZZ ### %%.
2.00 AAA XXXYYYYZZ BBB XXXYYYYZZ CCC XXXYYYYZZ.
3.00 .....

on & delete all 'xxxxyyzzz'.....0001.00:001(1)

```

In line 1, the first occurrence of the search string from the right began in column 29. This string was deleted.

Now, the entire work file is to be searched for the search string 'XXXYYYZZZ' and this string is to be deleted wherever it occurs.

```
1.00 *** ### %%.....  
2.00 AAA BBB CCC.....  
3.00 .....
```

@ON (Format 10) Delete the line contents before or after the search string

In the case of a hit, this format of @ON deletes all text between the beginning of the line and the search string (PREFIX) or between the search string and the end of the line (SUFFIX).

Operation	Operands	F mode / L mode
@ON	range [:domain] FIND [ALL] [F] [R] [PATTERN] search [,int] DELETE { PREFIX } { SUFFIX }	

range

A line range, specified as:

- one or more line numbers, separated by commas (e.g. 4,6,15)
- one or more line ranges, separated by commas (e.g. 5-10,17-19)
- a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).

A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.

domain

A column range, specified as:

- a single column (e.g. 10-10)
- a contiguous column range (e.g. 15-25).

If only one column number is specified, the line is searched from this column to the end of the line.

If the first column number is greater than the line length, this line is ignored. The second column number

- must not be less than the first column number
- may be greater than the actual line length.

If no column range is specified, the entire line is searched.

ALL

After each hit, EDT is to continue examining the remainder of the line after deleting the text before or after the search string, as appropriate.

F

Examination of each line range specified is to be terminated after the first hit.

R

The lines are to be searched from right to left instead of in the default direction (left to right).

PATTERN

The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.

- search The search string, specified either:
 - directly, in the form of a string enclosed in single quotes, or
 - indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.

- int The “int”th occurrence of the specified search string in a line is to be regarded as a hit.

- PREFIX All the text between the beginning of the line and the search string is deleted.

- SUFFIX All the text between the search string and the end of the line is deleted.

Example

```

1.00 ABABAB ABABAB ABABAB ABABAB.....
2.00 ABABABABABABABABABABABAB.....
3.00 .....

on %-.$ find 'ab'*3 , 4 delete prefix.....0001.00:001(1)

```

In a line range encompassing all the lines of the file (%-.\$), all text preceding the fourth occurrence of the string 'ABABAB'('AB'*3,4) in any line is to be deleted.

```

1.00 ABABAB.....
2.00 ABABABABABABABABABABABAB.....
3.00 .....

```

In line 1, the search string occurred for the fourth time at column 22. In line 2, the occurrences of the search string overlap: it was found for the first time at column 1, the second time at column 3, the third time at column 5, and the fourth time (the actual hit) at column 7.

@ON (Format 11) Delete the line containing the search string

This format of @ON deletes the entire line in which the search string is found.

Operation	Operands	F mode / L mode
@ON	range [:domain] FIND [ALL] [F] [R] [NOT] [PATTERN] search [,int] DELETE	

- range** A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- domain** A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is searched from this column to the end of the line.
If the first column number is greater than the line length, this line is ignored.
The second column number
- must not be less than the first column number,
 - may be greater than the actual line length.
- If no column range is specified, the entire line is searched.
- ALL** This may be specified, but it is meaningless for this format of @ON.
- F** Only the first record found in each specified line range is deleted.
- If neither ALL nor F is specified, the first hit in each line is deleted.
- R** The lines are to be searched from right to left instead of in the default direction (left to right).
- NOT** A hit is recognized if the search string is not contained in the specified column range of a line (negative search).
- PATTERN** The characters currently specified in “search” for * (asterisk) and / (slash) are interpreted as wildcards.

- search The search string, specified either:
- directly, in the form of a string enclosed in single quotes, or
 - indirectly, in the form of a line number, a line number variable or a string variable (in each case, with a column range if desired) (e.g. 5:2-6: or #L2 or #S5:2-3:). The line with the specified number or the specified variable must then contain the desired search string.
- int The "int"th occurrence of the specified search string in a line is to be regarded as a hit.

Example

```

1.00 1  ABC 2  ABC 3  ABC 4  ABC 5  ABC.....
2.00 1  ABC 2  ABC 3  ABC 4  ABC.....
3.00 1  ABC 2  ABC 3  ABC.....
4.00 1  ABC 2  ABC.....
5.00 1  ABC.....
6.00 1.....
7.00 .....

on & find 'abc',3 delete.....0001.00:001(1)

```

All lines in the entire work file which contain the search string 'ABC' at least three times are to be deleted.

```

4.00 1  ABC 2  ABC.....
5.00 1  ABC.....
6.00 1.....
7.00 .....

on & : 7 find 'a' delete.....0004.00:001(1)

```

Lines 1, 2 and 3 were deleted.

Now, all lines in the entire work file which contain the character 'A' in or after column 7 are to be deleted.

```
5.00 1 ABC.....  
6.00 1.....  
7.00 .....  
  
on & find ' '*2 delete.....0004.00:001(1)
```

Line 4 was deleted.

Next, all lines in the entire work file which contain two consecutive blanks (' ') are to be deleted.

```
6.00 1.....  
7.00 .....
```

Line 5 was deleted.

@OPEN Open and read file or library element

@OPEN has two formats with the following functions:

- Open and read an ISAM file for real processing (format 1).
- Copy a file into an ISAM file (format 1).
- Open and read a program library element or a file (format 2). The library element or file is read into the current work file, where it can then be processed.

@OPEN (Format 1) Processing an ISAM file on disk (real processing)

The ISAM file is not read into the memory area of EDT. Instead, only that part of the file which is currently being processed, i.e. the contents of one work window, is read into the current work file, which must be empty. Any changes to the file contents are written back to the disk file as soon as the DUE key is hit. Until closed by means of @CLOSE at the end of processing, the file remains physically open.

Copying files and real processing of SAM files

With the aid of @OPEN, a SAM or ISAM file which is to be processed ("file1") can be copied into an ISAM file. The copy of the file ("file2") is then opened.

A SAM file which is to be processed in real mode can thus be copied into an ISAM file and opened by means of @OPEN before it is processed. See also the section ["Real processing of SAM files" on page 378ff.](#) below.

Operation	Operands	F mode / L mode
@OPEN	['file1'] [(ver)] [[KEY] [AS 'file2' [OVERWRITE]]]	

- file1 A file name. If this file name does not yet exist, a file with this name is cataloged. file1 can be omitted if a file name has been previously defined by means of @FILE. If no file name has been previously defined by means of @FILE, file1 must be specified to prevent the @OPEN statement being rejected with an error message.
- ver The version number of the file.
This may consist of three digits or an asterisk (*), where * designates the current version number.
- KEY This is used only with SAM files created by means of @WRITE and KEY. KEY specifies that the keys stored in the SAM file are to be interpreted as line numbers, and not as part of the line contents.

- file2 A file name. If this file name does not yet exist, a file with this name is cataloged. If file2 is specified, file1 is copied into file2 and file2 is then opened.
- If OVERWRITE is not specified and file2 exists and is not empty, a warning message is issued. @OPEN is then executed only if
- the user responds to the warning message by entering Y,
 - file1 is not empty, and
 - file name file2 is not the same as file1.
- OVERWRITE Suppresses the warning message % EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO)? which is issued if file2 exists. Any file with this name is then overwritten. If file2 does not exist, OVERWRITE has no effect.

ISAM files can be processed in real mode only in work file 0. The work file must be empty. After an @OPEN statement, the statements @RUN, @RENUMBER and @COMPARE are rejected. Records of really opened files cannot be marked (e.g. @ON format 4) and are not autosaved (see @AUTOSAVE).

If an incorrect version number is specified in an @OPEN statement, the current version number is simply displayed; the file is not actually opened.

If the current version number or * is specified in @OPEN, this version number is incremented by 1 to form the new version number and displayed after @CLOSE.

Closing a file opened by means of @OPEN

- @CLOSE closes the file and deletes the work file and the entry for the local file name. Subsequent inputs thus refer to the work file (virtual file), not to the disk file.
- A second @OPEN implicitly initiates a @CLOSE: the first file is closed and the work file deleted before the second file is opened.

Real processing of ISAM files

The line numbers in the work file are taken from the ISAM keys in the ISAM file opened by means of @OPEN.

EDT does not check if there are any lines with more than 256 characters. When such a line is read in, the characters in columns 257 upwards are lost.

If file2 is specified, a copy of the original file is made by means of an implicit COPY-FILE command. The copy (file2) is then opened and processed; the original (file1) is not opened.

Real processing of ISAM files with an ISAM key length of less than 8 bytes

Before EDT is called, the key length must be specified in the following system command:

```
/SET-FILE-LINK LINK-NAME = EDTMAIN, FILE-NAME = filename, -  
/ ACCESS-METHOD = ISAM(KEY-LENGTH = keylength)
```

EDT can then be called and the file opened by means of @OPEN. In this case, it is not possible to specify ` instead of the file name.

It is advisable, after the file has been closed, to cancel the assignment of the file link name again, using the system command

```
/REMOVE-FILE-LINK LINK-NAME = EDTMAIN
```

By default, EDT expects or generates ISAM keys with a length of 8 bytes.

If a shorter key length is defined, any existing ISAM key is truncated from the left. If, for example, the key length is specified as 4, the line number 1234.5678 is interpreted as the ISAM key 5678. This means that duplicate ISAM keys may result: it is the user's responsibility to ensure that the keys are unique if he/she wants to use ISAM keys with a length of less than 8 characters.

Processing of ISAM files with fixed-length records (RECORD-FORMAT=FIXED) is not supported.

Real processing of SAM files

A SAM file file1 which is to be processed in real mode must be copied into an ISAM file file2 before it can be processed. This is done by specifying "AS 'file2'" in the @OPEN statement. EDT then copies SAM file file1 into ISAM file file2 and opens the latter.

Copying is actually executed by means of an implicit @READ for file1 followed by a @SAVE with the second file name. The file link name for the first file is EDTSAM, that for the second file EDTMAIN.

The ISAM key for the first line in file2 is the current line number which existed before @OPEN was executed. The ISAM keys of the following lines are formed by incrementing this key by the current increment value which existed before @OPEN was executed (see @In or @SET, format 6).

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of the file is taken into account in an @OPEN statement.

The @OPEN statement is only executed if the CCSN of the file is the same as the CCSN currently selected in EDT or all work files are empty and the coded character set can be displayed on the data display terminal.

@OPEN (format 2) Open and read into the current work file

@OPEN (format 2) is used to open a library element or a SAM or ISAM file and read it into the current work file.

Files with nonstandard file attributes can be opened without a SET-FILE-LINK command having been issued beforehand.

@OPEN (format 2) can be issued in work files 0 through 22.

Operation	Operands	F mode / L mode
@OPEN	$\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)][,elemtyp])} \\ \text{ELEMENT=elemname [(vers)][,elemtyp]} \\ \text{FILE=path2 [,TYPE=ISAM SAM CATALOG]} \end{array} \right\}$ [,MODE=ANY UPDATE NEW REPLACE]	

If more than one operand is specified, the operands must be separated by a blank or a comma.

LIBRARY = path1 ([ELEMENT]=]elemname [(vers)][,elemtyp])

The name of the library and of the desired library element.

ELEMENT = elemname [(vers)][,elemtyp]

The name of the desired library element, without a library name. The library name must have been defined previously by means of @PAR.

After successful execution of @OPEN, the library name is output.

path1 The library name.

path1 may also be specified by means of a string variable.

If path1 is omitted, the default library specified by means @PAR LIBRARY is used.

elemname The element name.

elemname may also be specified by means of a string variable.

vers The version number of the desired element (see the "LMS" manual [14]). If vers is not specified or if *STD is specified, the highest available version of the element is selected.

elemtyp The element type.

elemtyp may also be specified by means of a string variable.

Permissible type entries are: S, M, P, J, D, X, *STD or user-defined type names with appropriate base type. If no type is specified, the value preset in @PAR ELEMENT-TYPE will be used.

Users who specify a user-defined type name are responsible for ensuring that its associated base type corresponds to one of the permissible types S, M, P, J, D or X.

Type	Contents
S	Source programs
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format

*STD

Type S is the default value when EDT is started. Any other valid type specification can be defined as the default value by means of @PAR.

- FILE = path2** Opens and reads in a BS2000 file.
- path2** Fully qualified file name of the file to be opened.
path2 may also be specified by means of a string variable.
- TYPE** Defines the access method of the file.
- = SAM Default value. The file to be opened is a SAM file.
- = ISAM The file to be opened is an ISAM file.
- = CATALOG
The attributes are adopted from the existing file's catalog entry.
The access method is determined by the FCBTYPE attribute of the catalog entry.
Files with nonstandard attributes can be processed (this is equivalent to @READ and @WRITE following an assignment to the link name EDTSAM, or to @GET and @SAVE following an assignment to the link name EDTI-SAM). Files which can be read with SET-FILE-LINK LINK-NAME=EDTSAM, ACCESS-METHOD=ISAM cannot be opened with @OPEN.
- MODE** Defines the open mode for the library element or file.
- = ANY Default value. An existing or new library element or file can be opened.
- = UPDATE An existing library element or file is to be opened for processing.
- = NEW A new library element or file is to be created.
The specified element name must not already exist in the specified library or file.

= REPLACE

The contents of an existing element or file are to be replaced. The contents are not read into the work file.

If there is already a library element or file open in another work file, an error message is issued.

Calculation of line numbers when reading the file

The records are numbered in one of three ways as they are read in:

1. Default numbering with the default increment 1.0000
(e.g. 1.0000, 2.0000, 3.0000 ... 999.0000)
2. Numbering with a defined increment
as defined by means of @PAR INCREMENT
3. Automatic numbering if @PAR RENUMBER=ON is set:
This is done if the increment is too large to permit the entire file to be read in. EDT selects an increment which is smaller by a factor of 10 than the default (1.) or defined (2.) increment. Records which have already been read are renumbered with this smaller increment and the line numbers of further lines read in are calculated with this increment.
If necessary, the increment is reduced by further factors of 10 until the entire file can be read in or until EDT determines that the entire file cannot be read with the smallest permissible increment of 0.0001 (when the read operation is aborted with an error message).

If the increment is < 0.01, the following should be noted:

In F mode, the line numbers of records which are read in, copied or inserted are not displayed in full (since the line number display is only 6 positions long). If these incomplete line numbers are used in EDT statements (@COPY, etc.), errors may occur.

The current line number is set to the value for the last line read in plus the current increment value.



If the library containing the desired element is part of a file generation group, the library element type must be defined beforehand by means of @PAR ELEMENT-TYPE= elemtyp.

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of the file is taken into account in an @OPEN statement.

The @OPEN statement is only executed if the CCSN of the file (library element) is the same as the CCSN currently selected in EDT or all work files are empty and the coded character set can be displayed on the data display terminal.

Example

```
OPEN LIBRARY = PROGLIB (ELEMENT = TEST)
```

Element TEST of program library PROGLIB is opened and read into the current work file, which must be empty.

```
OPEN ELEMENT = PROC.EX(3), J
```

Before this statement is entered, the program library containing the element PROC.EX must be defined by means of @PAR LIBRARY = libname. Element PROC.EX, which contains a procedure (element type J), is opened and version 3 of this element is read into the current work file. The element remains open.

```
OPEN E=#S1
```

The element name is stored in the string variable #S1. The period preceding the variable name must be specified in order to prevent confusion between file names and element names.

@P-KEYS Define programmable keys

@P-KEYS is used to

- define the programmable keys on the keyboard (@P-KEYS)
- display the EDT default settings for the programmable keys (@P-KEYS SHOW).

Operation	Operands	F mode / L mode
@P-KEYS	[SHOW]	

SHOW The functions of the programmable keys defined by EDT using the @P-KEYS statement are displayed.
 This operand is supported in interactive mode on all data display terminals which are compatible with type 8160.

The programmable keys are defined as follows:

```

*** MEANING OF THE P-KEYS ***
P1 : position CURSOR to 1st command line
P2 : position CURSOR to 2nd command line
P3 :
P4 : skip to next page in first window
P5 : skip to previous page in first window
P6 : skip to next page in first window for corrections
P7 : skip to next page in second window
P8 : skip to previous page in second window
P9 : skip to next page in second window for corrections
P10: skip to the next mark in the first window
P11: skip to the previous mark in the first window
P12: position CURSOR eight characters to the right
P13: skip to the next mark in the second window
P14: skip to the previous mark in the second window
P15:
P16:
P17:
P18:
P19:
P20:
-----
                PRESS DUE1      FOR RETURN
    
```



- If the screen layout is changed by means of @PAR SPLIT, or the format is changed on a 9763 Data Display Terminal by means of @VDT, the @P-KEYS statement must be entered again.
- On the 3270 Data Display Terminal, the P-KEYS statement is rejected with an error message.

@PAGE Execute form feed

This statement executes a form feed on SYSLST.

Operation	Operands	F mode / L mode
@PAGE		

@LIST is used to set the maximum number of lines printed per page to a value between 1 and 256. @PAGE cancels this setting and resets to the default value of 65.

@PAR Enter default parameters

@PAR defines the default values for various file processing functions, namely:

Switching the following functions on and off:

- EDIT LONG mode (EDIT LONG)
- hexadecimal mode (HEX)
- lowercase or uppercase conversion (LOWER)
- overwrite mode (EDIT FULL)
- write protection on record level (PROTECTION) (only for the EDT program interface)
- column counter display (SCALE)
- information line display (INFORMATION)
- line number display (INDEX)
- screen optimization (OPTIMIZE)
- automatic renumbering (RENUMBER)
- two screen work windows (SPLIT)

Presetting values for:

- the work file to which @PAR applies (fwkfv/GLOBAL)
- the record separator character (SEPARATOR)
- the default code type (CODE)
- the default element type for a library element (ELEMENT TYPE)
- the increment value for line numbers (INCREMENT)
- the default library name (LIBRARY)
- the maximum record length in the F mode data window (LIMIT)
- the structure symbol for structure scrolling (STRUCTURE)
- the default for a program name (SDF-PROGRAM)

Operation	Operands	F mode / L mode
@PAR	[fwkfv GLOBAL] [,] [EDIT[[-] LONG] [=ON] =OFF] [,] [HEX [=ON] =OFF] [,] [LOWER [=ON] =OFF] [,] [EDIT[-]FULL [=ON] =OFF] [,] [PROTECTION [=ON] =OFF] [,] [SCALE [=ON] =OFF] [,] [INFORMATION [=ON] =OFF] [,] [INDEX [=ON] =OFF] [,] [OPTIMIZE [=ON] =OFF] [,] [RENUMBER [=ON] =OFF] [,] [SPLIT =n fwkfv =OFF] [,] [SEPARATOR ='char' =OFF] [,] [CODE =EBCDIC =ISO] [,] [[ELEMENT] [-] TYPE =elemtyp =*STD] [,] [INCREMENT =inc] [,] [LIBRARY =path] [,] [LIMIT =cl] [,] [STRUCTURE ='char'] [,] [SDF-PROGRAM =structured-name =*NONE] [,] [SDF-NAME-TYPE = INTERNAL = EXTERNAL]	

For the operands EDIT LONG, HEX, LOWER, EDIT FULL, PROTECTION, SCALE and INFORMATION, the preset value is not the same as the default value. The default value for these operands is ON.

If no operands are specified, all global and work file-specific values are given the same values as after calling EDT.

The work files for which the operands specified via @PAR are valid can be found in the table under [“Effects of the default values” on page 395](#).

The commas in front of the operands are only entered if 2 or more operands in a @PAR statement are to be specified.

fwkfv	<p>The work file variable (\$0-\$9) specifies which work file (0-9) @PAR refers.</p> <p>The OPTIMIZE operand always refers to all 10 work files (0-9).</p> <p>The SDF-PROGRAM and SEPARATOR operands always refer to all 23 work files (0-22).</p> <p>The SPLIT operand refers either to the current work file (if you are working in work files 0-9) or to the work file used before switching to L mode (if you are working in work files 10-22).</p> <p>The work file(s) to which @PAR refers if no work file variable is specified can be found in the table under “Effects of the default values” on page 395.</p> <p>If fwkfv is specified as an operand, it must be the first to be specified.</p>
GLOBAL	<p>GLOBAL specifies that @PAR is to refer to all work files.</p> <p>The operands CODE, ELEMENT-TYPE, INCREMENT, LIBRARY, LIMIT, LOWER, RENUMBER, SDF-PROGRAM and SEPARATOR refer to work files 0 to 22.</p> <p>The operands EDIT-FULL, EDIT-LONG, HEX, INDEX, INFORMATION, OPTIMIZE, PROTECTION, SCALE and STRUCTURE refer to work files 0 to 9, as they are only relevant in F mode.</p> <p>The SPLIT operand refers either to the current work file (if you are working in work files 0-9) or to the work file used before switching to L mode (if you are working in work files 10-22).</p> <p>The work file(s) to which @PAR refers if no work file variable is specified can be found in the table under “Effects of the default values” on page 395.</p> <p>If GLOBAL is specified as an operand, it must be the first to be specified.</p>
EDIT LONG	<p>Specifies whether records with more than 80 characters (3270 Data Display Terminal: 77 characters) are to be displayed in full in the work window.</p> <p>=ON The records are to be displayed in full (maximum length 256 characters).</p> <p>=OFF A maximum of 80 characters are displayed in the work window (3270 Data Display Terminal: 77 characters). This is the default value when EDT is started.</p>
HEX	<p>Switches hexadecimal mode on or off. In hexadecimal mode, each record is displayed in four lines on the screen. The first line contains the characters as they are normally displayed (or smudge characters if they cannot be displayed). Lines 2 and 3 contain the hexadecimal codes for the characters in line 1. The two digits of the hexadecimal code are displayed vertically below each character. Line four contains a scale (column counter) as a separator between the records.</p>

- =ON** Switches hexadecimal mode on.
If @PAR CODE=ISO, the hexadecimal value of the EDT data is output in ISO code (ASCII). Entries in the hexadecimal lines must be made in ISO code.
- =OFF** Switches hexadecimal mode off. This is the default value when EDT is started.
- LOWER** Specifies whether EDT is to convert lowercase letters entered from the keyboard into uppercase letters.
- =ON** EDT makes a distinction between uppercase and lowercase letters. All text and strings are processed exactly as they are entered.
- =OFF** EDT converts lowercase letters in the input into uppercase letters. Lowercase letters in a file are displayed on the screen as smudge characters.
This is the default value when EDT is started.
For further details, see @LOWER.
- EDIT FULL** Specifies whether the data window and the mark column are to be set to overwritable at the same time. Overwrite mode is only effective if the line number display is activated (@PAR INDEX=ON).
- =ON** The data window and the mark column are set to overwritable at the same time. It is possible to mark a line and to modify data in this line at the same time. Data can thus be copied by means of an O mark to a line which has not yet been created.

When the user changes to hexadecimal mode (@PAR HEX=ON) or to EDIT LONG mode (@PAR EDIT LONG=ON) or when he/she switches off the line number display (@PAR INDEX=OFF), this setting is not canceled, but merely deactivated.

As long as write protection (@PAR PROTECTION=ON) is set, entry of @PAR EDIT FULL=ON is ignored.
- =OFF** Default processing mode; data can be written to either the mark column or the data section of a screen line.
- PROTECTION** Write protection is activated at record level. This operand is only effective if used with the EDT subroutine interface. Records can then be write-protected or made overwritable from the user program by entering the appropriate mark (see the manual "EDT Subroutine Interfaces" [1]).
- =ON** Appropriately-marked records in F mode dialog are write-protected, while all others are automatically set to overwritable.
If EDIT FULL has been set, it is reset.

- =OFF The default value specified by the user program (write-protected or overwritable) has no effect.
The value OFF is preset when EDT is called.
- SCALE Displays a column counter (scale) in the data window (not valid in EDIT LONG mode).
- =ON The column counter appears as the first line after any information line present and displays the current column numbers of the work window (e.g. after horizontal shifting of the work window).

If a tab has been defined (see @TABS), a further screen line is displayed, in which the current positions of the tabs are indicated by “|”.
If a tab character has been defined (see @TABS), it is shown in the mark column position.
- =OFF Switches off the column counter and, if appropriate, the tab display scale.
The default value is OFF when EDT is called.
- INFORMATION Displays in the data window an information line containing one of the following (not valid in EDIT LONG mode):
a local @FILE entry
– explicitly defined by means of @FILE, or
– implicitly defined by means of @READ, @GET, @OPEN,
the library and element name of
– a library element opened with @OPEN (format 2),
the POSIX file name of
– a POSIX file opened with @XOPEN (if the file name is too long, its beginning is abbreviated with ...) or
a header line in work file 9 if
– @FSTAT LONG, @STAJV LONG or @SHOW is issued without a destination entry in the F-mode dialog and the content has not been changed.
- =ON The information line appears as the first line in the work window, i.e. before a scale, if one has been activated. If none of the above-mentioned names and no header line is defined, the name field remains empty.
- =OFF The information line is switched off.
The value OFF is preset when EDT is called.
- INDEX Switches the line number display on and off. This causes the screen format to change.

- =ON Switches the line number display on (default EDT format). Each line on the screen contains the 6-digit line number, one blank and 72 characters (3270 Data Display Terminal: 69 characters).
- =OFF Switches the line number display off. Each line on the screen contains 80 characters (3270 Data Display Terminal: 77 characters).
The first column in each line can be overwritten (mark column).

@PAR INDEX=ON deactivates of EDIT LONG mode.

In the case of the 3270 Data Display Terminal, the field separator characters occupy one space on the screen. As a result the section available for data is reduced by one space.
- OPTIMIZE Switches screen optimization on and off.
Before executing a screen output, EDT compares the contents of the new screen and the old screen. By default, it then outputs only that text which has been changed, in order to optimize the output performance; any unchanged text in the old screen contents is left unchanged.
- =ON Only the modified lines are to be output (default value).
- =OFF The complete contents of the work window are to be output.
- RENUMBER Switches automatic line renumbering on and off. This becomes effective after the statements @OPEN and @COPY (format2), @XCOPY, @XOPEN, @SDFTEST, @SEPARATOR, and after the statement codes C, M and R. See also "Calculation of line numbers" in the descriptions of the @COPY and @OPEN statements.
- =ON The lines of a work file are to be renumbered as required (default value). EDT does this if, when a file (or library element) is read in or when lines are copied or inserted, the increment is too large to permit correct execution of the statement.
- =OFF Switches off automatic line renumbering.

The line numbers of a file are not changed by EDT. If @OPEN or @COPY cannot be executed because the increment is too large to accommodate all the lines, EDT issues a corresponding message.
- SPLIT Specifies that a second work window is to be displayed on the screen. The file in which this statement is entered (the current work file) is displayed in the upper work window and the work file specified by fwkfv is displayed in the lower work window.
- =n fwkfv Specifies that a second work file is to be displayed, as follows:
n: the number ($2 \leq n \leq 22$) of lines in the lower work window.
fwkfv: a work file variable (\$0 to \$9) which specifies which work file is to be displayed in the lower work window.

- The file position (line and column number) remains unchanged when @PAR is entered.
During further processing with EDT, the positions of the two work files can be changed independently.
- =OFF** Removes one work window from the screen and expands the other window to the full screen size of 24 lines (default work window).
The work window in whose statement line this statement is entered is expanded to fill the screen.
If @PAR SPLIT = OFF is entered in the upper work window when the lower work window contains statements, @PAR SPLIT is rejected with an error message.
- SEPARATOR** Defines the record separator character:
– for "splitting a record" (see [page 119](#)) or
– to be used as the default value for the @SEPARATE statement
- = 'char'** A freely selectable alphanumeric or special character which is to be used as the record separator. It must be enclosed in single quotes. The single quotes can be redefined by means of @QUOTE.
Different characters must be defined for the record separator and the tab character.

The defined record separator character can be output by means of the statement @STATUS=SYMBOLS.
- =OFF** Resets the record separator definition. This is the default value, i.e. no record separator character is defined.
- CODE** Default code type.
Defines the default for the CODE operand in the statements @XOPEN, @XCOPY and @XWRITE. In hexadecimal mode (@PAR HEX=ON or HEX ON), this default determines how EDT data is displayed in the data window. When EDT is called up, the value is preset to EBCDIC.
- =EBCDIC** For hexadecimal output, the hexadecimal value of the EDT data is output as EBCDIC code in the data window.
This defines EBCDIC as the default value for the CODE operand in the statements @XOPEN, @XCOPY and @XWRITE.
- =ISO** For hexadecimal output, the hexadecimal value of the EDT data is output as ISO code in the data window. This code corresponds to ASCII code. The display shows how the data is stored in POSIX files if it is written back with @XWRITE (CODE=ISO).
This defines ISO as the default value for the CODE operand in the statements @XOPEN, @XCOPY and @XWRITE.

ELEMENT-TYPE

Specifies the default element type.

This element type is assumed if a @COPY, @DELETE, @OPEN, @INPUT or @WRITE statement is entered without an element type.

=elemtyp Permissible type entries are: S, M, P, J, D, X, R, C, H, L, U, F, *STD or user-defined type names with appropriate base type. elemtyp may also be specified in the form .str-var.

Type	Contents
S	Source programs
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format
R	Object modules
C	Load modules
H	Created by ASSEMBH
L	Created by BINDER
U	Created by IFG
F	Created by IFG

*STD Type S is the default value when EDT is started.
If the default element type has been changed, @PAR ELEMENT-TYPE = *STD returns to the original default type.

INCREMENT

= inc Specifies the increment value for line numbers for @OPEN format 2, @COPY format 2, @INPUT format 2, @XCOPY und @XOPEN . In F mode, this defines the increment value for line numbers of screen lines which do not contain any data. The default increment value is 1.

If @PAR INCREMENT is specified with an increment < 0.01, then, in F mode, the line numbers of records which are read in, copied or inserted are not displayed in full (since the line number display is only 6 positions long). If these incomplete line numbers are used in EDT statements (@COPY, etc.), errors may occur.

LIBRARY

=path Default library name.
The elements of this library are accessed if only an element name is specified in a @COPY, @OPEN, @WRITE or @INPUT statement. If no

library name is specified in a @SHOW statement, the directory of this PLAM library is output.

path may also be specified in the form .str-var.

LIMIT

= cl Specifies the maximum record length in the F mode data window. If a record longer than this is entered, it is truncated and the message % EDT2267 LINE TRUNCATED AFTER nnn CHARACTERS is issued. Indirect changes to records by EDT statements, etc. are not subjected to this test. The permissible value range for cl is 1...256; the default value is 256.

STRUCTURE

= 'char' The structure symbol is the escape character for structure scrolling (see the [section "+/- Position work window by structure depth" on page 114ff](#)). It must be enclosed in single quotes. The single quotes can be redefined by means of @QUOTE. This symbol identifies records which are to be evaluated during structure scrolling.
If a structure symbol other than a blank is defined, only those records containing at least this structure symbol are evaluated.
If a blank is entered for the structure symbol, all records are evaluated.
The default value for the structure symbol is '@' (e.g. for Columbus source programs).

SDF-PROGRAM

Default program name for the @SDFTEST statement and the t statement code.

Data lines beginning with // are considered to be statements of this program.

SDF-A can be used to determine the internal program name if it does not match the name of the program.:

=structured-name
Name of a program.
The name is valid globally, for all work files.

=*NONE Cancels any previous definition.
*NONE is the default when EDT is started.

SDF-NAME-TYPE

Defines the name type of the pre-defined program name in the @PAR SDF-PROGRAM statement and the default name type in the @SDFTEST statement.

=INTERNAL
Program name is internal name, maximum 8 characters.
The internal program name can be determined with SDF-A, if it does not correspond to the name of the program.

=EXTERNAL

Program name is external name, maximum 30 characters
(e.g. LMS, SDF-A, HSMS).

INTERNAL is the default when EDT is started.

The setting of SDF-NAME-TYPE is always effective for all EDT work files (0-22).

Effects of the default values

The following table shows the work files for which the operands specified with @PAR are valid. The determining factor is whether GLOBAL, a work file variable (fwkfv: \$0 to \$9) or neither of these was specified in the @PAR statement.

Operand	@PAR		
	GLOBAL	fwkfv	without GLOBAL / fwkfv
EDIT-FULL	Work files 0-9	Specified work file	in work file 0-9: current work file; in work file 10-22: has no effect
EDIT-LONG			
HEX			
INDEX			
INFORMATION			
PROTECTION			
SCALE			
STRUCTURE			
CODE	Work files 0-22	Specified work file	Current work file
ELEMENT-TYP			
INCREMENT			
LIBRARY			
LIMIT			
LOWER			
RENUMBER			
OPTIMIZE			
SDF-PROGRAM	Work files 0-22		
SEPARATOR			
SPLIT	in work file 0-9: current work file; in work file 10-22: work file before switching to L mode		

@PARAMS Define EDT parameters

@PARAMS defines all symbolic parameters which are used within a procedure.

Parameters in EDT procedures

The parameters can be regarded as character variables which are replaced by the appropriate values before a procedure is executed.

A parameter begins with the character &, which is followed by a letter and by up to six further letters or digits. Lowercase letters may also be used, but it should be noted that EDT distinguishes between uppercase and lowercase letters, which means, for example, that &A and &a are two different parameters. The parameter names used in a procedure are valid only within this work file.

EDT parameters are

- defined by means of @PARAMS in the first line of an EDT procedure and
- set to the desired values by means of @DO when the procedure is called.

A distinction is made between positional parameters and keyword parameters.

Positional parameters are set in the order in which they occur to the values specified in the @DO statement (see @DO).

In *keyword parameters*, the parameter name is followed by an equals sign and the parameter value. Keyword parameters are also set to the values specified in the parameter list of the @DO statement. If no value is specified here, the default value predefined in the @PARAMS statement is used. The value of a parameter is determined by all the characters which are entered, including any blanks. If no value is specified for a positional parameter or for a keyword parameter which has no default value, the parameter is set to an empty string.

If a parameter value contains commas or closing parenthesis marks, the parameter must be enclosed in single quotes. To create a single quote in a parameter value enclosed in single quotes, it is necessary to enter the single quote twice (see also the example). If a @QUOTE statement assigning the function of the single quote to a different character has been issued, it does not apply to the single quotes enclosing the parameter value.

Example

Line in procedure file	Parameter entry	Created line
@ON &F'&SEARCH'	&SEARCH=A"B	@ON &F'A"B'
@SET #S1=&STR	STR=┘'TEXT'	@SET #S1=┘'TEXT'
&DATA	&DATA='A'B	'A'B
@P RANGE	&RANGE='3,7'	@P 3,7

Operation	Operands	@PROC
@PARAMS	formal [...]	

formal A formal (symbolic) parameter.

In @PARAMS, EDT ignores blanks which occur

- in, before or after a formal positional parameter
- anywhere before the equals sign in a formal keyword parameter.

The following, for example, @PARAMS & A B C, &LINE= is equivalent to @PARAMS &ABC,&LINE=

This applies only to @PARAMS. In the @DO statement, no blanks are permitted in the parameter name.

If no default value is to be defined for a keyword parameter, a comma must be entered (or the @PARAMS statement terminated) immediately after the equals sign.

,... This indicates that several formal parameters may be entered, separated from each other by commas. If both positional and keyword parameters are specified, the positional parameters must be entered first. Positional parameters must be specified in the same order in the @PARAMS statement and the @DO statement. Keyword parameters may be specified in any order in either statement.

The maximum number of formal parameters in a @PARAMS statement is limited only by the maximum length of an EDT statement (256 characters).

If parameters are to be used in a procedure, the @PARAMS statement must be entered in the first line of this procedure. @PARAMS statements elsewhere in the procedure are ignored.

Replacing the parameters by their values

The parameter values specified in the @DO statement are assigned to the formal parameters in the procedure by the @PARAMS statement. If @PARAMS is omitted, no current values are assigned to the parameters. The same applies to parameters which are not specified in the @PARAMS statement. If a keyword parameter is not specified in the @DO statement, it receives the default value specified for it in the @PARAMS statement.

The parameters may be used anywhere in the procedure and may be chained with strings and other parameters.

If there is a period between the formal parameter and the following string or following formal

parameter, this period does not appear in the result of the chaining operation. EDT interprets the period as an indicator that chaining is to be executed. If a parameter value is to be chained to a following string which begins with a letter, a digit or a period, the period between the formal parameter and this string must always be entered. In the following examples, it is assumed that the parameter @PARAM has the value A.

Entry in the procedure line	Generated character
&PARAM(BC)	A(BC)
&PARAM.(BC)	A(BC)
&PARAM..(BC)	A.(BC)
&PARAM..BC	A.BC
&PARAM.2BC	A2BC
&PARAM.,2B	A.,2B
BC&PARAM	BCA
BC,&PARAM	BC,A
B2&PARAM	B2A
&PARAM.&PARAM	AA
&PARAM&PARAM	AA
&PARAM..&PARAM	A.A

If, in exceptional cases, a formal parameter specified in @PARAMS is not to be replaced by its current value in the procedure, then the & character must be duplicated. When the procedure is executed, one of these & characters is removed.

Example

@PARAMS &PRINTER=

&&PRINTER=&PRINTER

@DO...(PRINTER=L2) results in the following line in the procedure:

&PRINTER = L2



When a formal parameter is replaced by its current value in a procedure, the resulting line may be longer than 256 characters and an error message will be issued when the procedure is executed.

Example 1

```

7.      @PRINT
1.0000 THE OUTPUT
2.0000 FROM THIS
3.0000 PROCEDURE
4.0000 IS DETERMINED
5.0000 IN THE @DO
6.0000 STATEMENT
7.      @SET #S3 = '*** AS YOU SEE ***'
7.      @PROC 1
1.      @ @PARAMS &LINES ----- (01)
2.      @ @PRINT &LINES ----- (02)
3.      @END
7.      @DO 1(2-4) ----- (03)
2.0000 FROM THIS
3.0000 PROCEDURE
4.0000 IS DETERMINED
7.      @DO 1(#S3),PRINT
7.      @PRINT #S3 ----- (04)
      #S03 *** AS YOU SEE ***
7.      @DO 1(2,4N) ----- (05)
% EDT4963 TOO MANY OPERANDS
7.      @DO 1('2,4N') ----- (06)
2.0000 FROM THIS
IS DETERMINED
7.

```

- (01) In work file 1, the positional parameter &LINES is defined in line 1.
- (02) This parameter appears in the @PRINT statement. The lines which are actually displayed thus depend on the parameter value specified in the @DO statement.
- (03) Work file 1 is executed. Before this is done, however, the value range 2-4 is assigned to the parameter &LINES.
- (04) The replacement of the parameter by its value is easily seen if the procedure statements are displayed on the screen before they are executed, since they already contain the parameter value at this time.
- (05) If the user attempts, for example, to display line 2 with a line number and line 4 without a line number, the comma which is part of the necessary parameter value is regarded as the separator between two parameters, and the @DO statement is rejected.
- (06) It is also possible to enclose the parameter value in single quotes. In this case, everything between the single quotes is passed to the parameter &LINES, which means that commas can also be passed as part of the parameter value.

Example 2

```

1.      @PROC 2
1.      @ @PARAMS &STRVAR1,&STRVAR2,&CONTENT1=**** ----- (01)
2.      @ @SET &STRVAR1 = '&INHALT1'
3.      @ @SET #S2 = '&STRVAR1'
4.      @ @SET #S3 = &STRVAR1
5.      @ @SET &STRVAR2 = &STRVAR1
6.      @ @SET #S4 = 'VON &STRVAR1 BIS &STRVAR2'
7.      @ @PRINT &STRVAR1,&STRVAR2,#S2,#S3,#S4
8.      @END
1.      @DO 2(#S0,#S1) ----- (02)
#S00 ****
#S01 ****
#S02 #S0
#S03 ****
#S04 FROM #S0 TO #S1
1.      @DO 2(#S15,#S13,CONTENT1=RHUBARB) ----- (03)
#S15 RHUBARB
#S13 RHUBARB
#S02 #S15
#S03 RHUBARB
#S04 FROM #S15 TO #S13
1.

```

- (01) Two positional parameters and one keyword parameter are defined in work file 2.
- (02) The values for the positional parameters must be specified in @DO in the order in which these parameters were defined in the @PARAMS line. In this case, &STRVAR1 is set to #S0 and STRVAR2 is set to #S1 when the procedure is executed. Since nothing is specified for the keyword parameter &CONTENT1, the default value (****) is used when the procedure is executed.
- (03) This time, a parameter value is specified in @DO for the keyword parameter &CONTENT1 and replaces the default value.

Example 3

```

1.      @PROC 3
1.      @ @PARAMS &A,&B,&C,&X=111,&Y=222,&Z=333 ----- (01)
2.      @ @CREATE #S10: '&A','&B','&C','&X','&Y','&Z'
3.      @ @PRINT #S10
4.      @END
1.      @DO 3 (AAAAA,BB,CCCCCCC) ----- (02)
#S10 AAAAAABBBBBCCCC111222333
1.      @DO 3 (AA,BBBB,C,Y=****,X=#####) ----- (03)
#S10 ABBBBBC#####****333
1.

```

- (01) Three positional parameters and three keyword parameters are defined in work file 3.
- (02) Work file 3 is executed. Since no values have been specified for the keyword parameters, the default values are used.
- (03) Here, values are specified for two of the keyword parameters. Note that the values for these keyword parameters are not specified in the same order as the definitions of these parameters in the @PARAMS line.

@PREFIX Insert string as prefix

@PREFIX inserts the specified string as a prefix at the beginning of each line in the specified line range (see also the @SUFFIX statement, for a description of appending strings to lines).

Operation	Operands	F mode / L mode
@PREFIX	range WITH string	

- range A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- string The string to be inserted as a prefix at the beginning of each line in the specified line range.
- The string may be specified:
- explicitly, enclosed in single quotes, or
 - implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

Example

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AGAIN.....
5.00 AGAIN.....
6.00 .....

prefix 4-5 with ' ONCE ' .....0001.00:001(0)

```

The string ' ONCE ' is to be inserted as a prefix in lines 4 and 5.

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 ONCE AGAIN.....
5.00 ONCE AGAIN.....
6.00 .....

prefix 4-5 with 1 .....0001.00:001(0)

```

The contents of line 1 are to be inserted as a prefix in lines 4 and 5.

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AND ONCE AGAIN.....
5.00 AND ONCE AGAIN.....
6.00 .....

prefix 4-5 with ' '*5 .....0001.00:001(0)

```

Five blanks are to be inserted as a prefix in lines 4 and 5.

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AND ONCE AGAIN.....
5.00 AND ONCE AGAIN.....
6.00 .....

prefix 4-5 with 4 .....0001.00:001(0)

```

The contents of line 4 are to be inserted as a prefix in lines 4 and 5.

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AND ONCE AGAIN AND ONCE AGAIN.....
5.00 AND ONCE AGAIN AND ONCE AGAIN.....
6.00 .....

```

@PRINT Print or display lines or string variables

In L mode, @PRINT prints or displays the lines in the specified range or the contents of specified string variables. In F mode, only the contents of string variables can be printed or displayed.

In interactive mode, the output is sent to the screen (SYSOUT); in batch mode, it is sent to the printer (SYSLST).

Operation	Operands	F mode / L mode
@PRINT	[rng [:domain] [X] [N] [S] [V E]] [,...]	

- rng** A line range, specified as:
- a single line (e.g. 6)
 - several contiguous lines (e.g. 8-20).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- rng must be specified if X, N, V or S is used.
- If rng is omitted, the entire work file is output one section at a time (as for V).
- domain** A column range, specified as:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25).
- If only one column number is specified, the line is output from this column to the end of the line.
- If the first column number is greater than the line length, this line is ignored.
- The second column number
- must not be less than the first column number,
 - may be greater than the actual line length.
- If no column range is specified, the entire line is output.
- X** The output is to be in hexadecimal format. In this case, the maximum length of a line to be output must not exceed 128 characters. If X is used, rng must be specified.
- N** Suppresses the line numbers or string variable names, as appropriate, in the output. rng must be specified.

- S This is meaningful only in batch mode and suppresses the first empty line which normally precedes the first output line when the output is directed to the printer. If S is used, rng must be specified.
- V This is meaningful only for working at the screen and causes EDT to display the specified range of lines one section at a time. Following output of the first section, EDT always requests a paging input, even if the last line of the specified line range has already been output. In subsequent empty page inputs (only DUE), EDT operates in the same way as when + is entered.
- The display terminal being used determines how many physical lines (screen lines) there are in a given section. The value prescribed by the system (screen size) can be changed with @VDT.
- E Like V except for the following difference:
If the first section contains the last line of the specified line range, the @PRINT function is terminated after outputting that line. In subsequent empty page inputs (only DUE), EDT operates in the same way as when * is entered.

Output of work file contents

If rng is specified, this entire range is output. The output is interrupted only if the overflow monitor of the operating system (MODIFY-TERMINAL-OPTIONS OVERFLOW-CONTROL = USER-ACKNOWLEDGE) is active.

After an output interrupt via % PLEASE ACKNOWLEDGE, output of the line range can be aborted by means of **K2** and RESUME-PROGRAM. Positioning within the line range is not possible.

Like a @PRINT statement without rng, an entry of V or E causes EDT to display the range one section at a time. The number of screen lines set by the system or by means of @VDT are displayed.

If neither V nor E is specified in a @PRINT statement without rng, EDT operates as if V had been specified.

After displaying each section, EDT requests the entry of *, +, - or 0, by means of which the user can abort the output or specify what is to be displayed next.

When only `[DUE]` is entered, EDT's response depends on the operand V or E, or on the most recent paging input *, + or +int.

- * EDT displays the section immediately following the one which has just been displayed. If the new section contains the last line of the specified range, the @PRINT function is terminated after this line has been displayed.
In subsequent empty page inputs (only DUE), EDT operates in the same way as when * is entered.
If the section displayed previously overshoot the upper or lower limit of the requested line range, entering * terminates the @PRINT function immediately, without any further output.
- + EDT displays the section immediately following the one which has just been displayed, even if some or all of the lines lie outside the specified line range. In this way, it is possible to display any lines desired up to the end of the file. In subsequent empty page inputs (only DUE), EDT operates in the same way as when + is entered.
- +int EDT displays the section which starts int lines after the last line which has just been displayed, even if some or all of the lines in this section lie outside the specified line range. In this way, it is possible to display any lines desired up to the end of the file.
In subsequent empty page inputs (only DUE), EDT operates in the same way as when + is entered.
- EDT displays the section immediately preceding the one which has just been displayed, even if some or all of the lines in this section lie outside the specified line range. In this way, it is possible to display any lines desired back to the beginning of the file.
In subsequent empty page inputs (only DUE), EDT operates in the same way as when + is entered.
- int EDT displays the section which starts int lines before the first line which has just been displayed, even if some or all of the lines in this section lie outside the specified line range. In this way, it is possible to display any lines desired back to the beginning of the file.
In subsequent empty page inputs (only DUE), EDT operates in the same way as when + is entered.
- 0 Terminates output of the line range.

If a line in the current work file would extend over several screen lines, EDT will not split this line between two sections of the output. This may mean that one screen contains less than the maximum possible number of lines.

If EDT was called as an independent program by means of START-PROGRAM or LOAD-PROGRAM, the user may enter a statement instead of *, +, +int, -, -int or 0. This causes the @PRINT function to be terminated and this statement to be executed. Note, however, that a sequence of statements must not be entered in this case, even if block mode is active. If EDT was called as a subroutine, it will accept a statement as input when the @PRINT function has not been completed only if bit 2⁰ in function byte 2 of the parameter list is not set (see the “EDT Subroutine Interfaces” manual [1]).

@PROC Switch work files

@PROC has two formats, which provide the following functions in L mode:

- switching to another work file (format 1)
- displaying information about the free and used work files and/or about the current work file (format 2).

@PROC (format 1) Switch work files

This format of @PROC is used in L mode to switch to another work file.

Operation	Operands	L mode / @PROC
@PROC	procno [comment]	

procno The number of a work file in which EDT procedures can be executed (1 to 22) or an integer variable which contains one of these values.

comment A comment of the user's choice.
This makes it possible to insert a comment even when a strict syntax check (@SYNTAX) is to be run.

The work file to which the user switched using this format of @PROC remains the current work file until

- the user returns to the previous work file by means of @END or
- a further @PROC or @SETF (procno) statement is entered to switch to another work file.

@PROC switches to another work file without terminating the old one(s) (nested work files). @SETF (procno), on the other hand, terminates all older work files before switching to the new one.

If the specified work file is empty, the current line number is 1 and the current increment is also 1.

If the specified work file already contains data or statements, the current line number and the current increment have the values which existed when the user last exited from this work file.

Example 1

```

5.      @PRINT
1.0000 AAAA
2.0000 BBBAADAFD
3.0000 AAA
4.0000 CCCCCCCCCCCCCCCC
5.      @PROC 6
1.      @ @SET #I6 = LENGTH !
2.      @ @SET #L6 = !
3.      @ @DO 10
4.      @ @DO 12
5.      @ @CREATE #S6: 'LINE ',#S12,' IS ',#S10,' CHARACTERS LONG'
6.      @ @PRINT #S6 N
7.      @END ----- (01)
5.      @PROC 10 ----- (02)
1.      @ @SET #S10 = CHAR #I6
2.      @ @ON #S10:2-2: DELETE '0'
3.      @ @IF .TRUE. GOTO 2
4.      @END
5.      @PROC 12 ----- (03)
1.      @ @SET #S12 = CHAR #L6
2.      @END
5.      @DO 6 !=%, $
LINE   1.0000 IS  4 CHARACTERS LONG
LINE   2.0000 IS  9 CHARACTERS LONG
LINE   3.0000 IS  3 CHARACTERS LONG
LINE   4.0000 IS 17 CHARACTERS LONG
5.

```

- (01) The user switches to work file 6, which contains 6 EDT statements, including a @DO 10 and a @DO 12. At the moment, these two work files do not exist, and entering @DO 6 would result in an error.
- (02) A procedure is defined in work file 10. Its task is to convert the value in #I6 into printable form and place it in #S10 and then to delete any leading zeros.
- (03) A procedure is defined in work file 12 to convert the contents of line number variable #L6 into printable form and to place them in #S12.

Example 2

```

1.      @SET #I4 = 1
1.      @PROC #I4 ----- (01)
1.      @4.00
4.00    @ @SET #I4 = #I4 + 1 ----- (02)
4.01    @ @IF #I4 > 4 GOTO 8
4.02    @ @PROC #I4
4.03    @ @PROC ----- (03)
4.04    @ @GOTO 4
4.05    @8.00
8.00    @ @SET #I4 = #I4 - 1
8.01    @ @END ----- (04)
8.02    @ @IF #I4 = 2 RETURN
8.03    @ @PROC ----- (05)
8.04    @ @GOTO 8
8.05    @END
1.      @DO #I4 ----- (06)
<02>
<03>
<04>
<03>
<02>
1.

```

- (01) It is possible to switch to another work file with the aid of an integer variable, whose value must lie between 1 and 22.
- (02) When work file 1 is executed, EDT switches to work files 2 to 4, the number of the desired work file being passed in #I4.
- (03) @PROC is used to check which work file is currently being used. Since the work file is selected via #I4, the response to this inquiry will be equal to the value of #I4 each time the work file is executed.
- (04) A single @END is used to switch back to work file 0 in all cases.
- (05) After closing, the procedure again asks which work file is currently being used. This results in the work file numbers 3 and 2.
- (06) Execution of a work file can also be started with the aid of an integer variable.

@PROC (format 2) Output information

With this format of @PROC, the user can request the following information:

- the number of the current work file (@PROC)
- the numbers of all free work files (@PROC FREE)
- the numbers of all used work files (@PROC USED)

Operation	Operands	L mode / @PROC
@PROC	[FREE USED]	

FREE The numbers of the work files (1-22) which have not yet been used are displayed.

USED The numbers of the work files (1-22) which have already been used are displayed, together with the lowest and highest line number for each such file.

If no work files apart from work file 0 have been used, EDT issues a message.

If no operand is specified, the number of the current work file is displayed on the screen.

Example 1

```

1.   @PROC ----- (01)
<00>
1.   @PROC 15 ----- (02)
1.   @PROC ----- (03)
<15>
1.   @END ----- (04)
1.   @PROC ----- (05)
<00>

```

- (01) This statement asks which is the current work file: the response is the number <00>, i.e. the main file.
- (02) This switches to work file 15.
- (03) When the number of the current work file is requested, the answer is now, of course, 15.
- (04) Switches back to work file 0.
- (05) The response is now again 0.

Example 2

```

1.      @DROP ALL ----- (01)
1.      @PROC USED ----- (02)
% EDT0907 NO PROCEDURE FILES DECLARED
1.      @PROC 13
1.      A
2.      @END
1.      @PROC USED ----- (03)
<13>    1.0000 TO    1.0000

```

- (01) All work files are released.
- (02) The user asks which work files have been defined.
- (03) After defining work file 13, the user again asks which work files are defined.

Example 3

```

1.      @DROP ALL ----- (01)
1.      @PROC FREE ----- (02)
% EDT0907 NO PROCEDURE FILES DECLARED
1.      @PROC 13
1.      A
2.      @END
1.      @PROC FREE ----- (03)
01 02 03 04 05 06 07 08 09 10 11 12 14 15 16 17 18 19 20 21 22

```

- (01) All work files are released.
- (02) The user asks which work files have not yet been defined.
- (03) After defining work file 13, the user again asks which work files have not been defined.

@QUOTE Redefine delimiter for strings

Wherever a string has to be specified in a statement (as a search or replacement string, a file name, etc.), this string must be enclosed in single quotes. The @QUOTE statement defines characters which replace these quotes.

Operation	Operands	F mode / L mode
@QUOTE @QE	{ spec, char } spec ,char	

spec Special character which replaces the single quote character (').

char Character which replaces the double quotes character (").

spec and char must be different.

Double quotes are used only in @ON statements, namely together with text delimiters (see @DELIMIT and @ON). Single quotes are likewise used in @ON statements, but also in other statements.

If spec is not a special character, @QUOTE is rejected with the following error message:
% EDT3952 INVALID SYMBOL



A typical application for this statement is where @ON is to be used to find a string which contains a single or double quotes character.

@RANGE Define line range symbol

@RANGE defines a new symbol for a line and column range. The existing range symbol becomes invalid.

Operation	Operands	F mode / L mode
@RANGE	[=r =rng [:domain]]	

r The symbol for the range to be defined.

rng A line range, specified as:
 – a single line (e.g. 6)
 – several contiguous lines (e.g. 8-20).

A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables.

String variables (#S0 to #S20) may also be used.

domain A column range, specified as:
 – a single column (e.g. 10-10)
 – a contiguous column range (e.g. 15-25).

This column range is only valid in conjunction with @ON.

If only one column number is specified, the column range extends from this column to the end of the line.

If the first column number is greater than the line length, this line is ignored.

The second column number

- must not be less than the first column number
- may be greater than the actual line length.

If no domain is specified, the column range is assumed to be 1-256.

Only one range symbol exists at any one time. The default range symbol is the & character and the corresponding default range is 0.0001-9999.9999.

If no operand is specified, the current range symbol is canceled. There is then no current range symbol until a new one is defined again by means of @RANGE. The column range is not changed.

If r is not a special character, @RANGE is rejected with the following error message:
 % EDT3952 INVALID SYMBOL

@READ Read SAM file

@READ is used to read all or part of a SAM file from disk or tape into the current work file.

The file is open only during the execution of the @READ statement. By default, EDT assumes that the SAM file contains variable-length records (see [section “Processing SAM files with nonstandard attributes” on page 52](#)).

Operation	Operands	F mode / L mode
@READ	['file'] [(ver)] [range*] [:col:] [{ RECORDS } { KEY }] [STRIP]	

- file** A file name.
If there is no local @FILE entry for the file name, the name specified here is stored as this entry. If the “file” operand is omitted, the local @FILE entry, if one exists, is used as the file name; otherwise, the global @FILE entry is used (see the description of the @FILE statement). If neither a local nor a global @FILE entry nor the SAM file exists, @READ is rejected with an error message.
If the file link name EDTSAM is assigned to a file, this file can be read by simply specifying / (see [section “File processing” on page 49ff](#)).
- ver** The version number of the file to be read.
This may consist of up to three digits or an asterisk (*). * designates the current version number.
If * is specified, the current version number is displayed. If an incorrect version number is specified, the correct version number is displayed and this file is then read.
- range*** A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15 or 0.0004,0.0006,0.0015)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19 or 0.0005-0.0010,0.0017-0.0019)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8 or 0.0004,0.0007-0.0023,0.0008).
- The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. The values of symbolic line numbers refer to the current work file.
String variables must not be used.

If range* is specified without RECORDS, EDT regards the first 8 characters in each line as the line number, and not as part of the line contents. In this case, for example, 1 would refer to the line which begins with the string 00010000.

If range* is specified together with RECORDS, then range* refers to the logical line numbers. In this case, for example, 0.0001-0.0005 would read the first 5 lines from the file.

If range* is not specified, all lines in the file are read.

col

A column range, specified as:

- one or more columns, separated by commas (e.g. 10,15,8)
- one or more column ranges, separated by commas (e.g. 15-25,18-23)
- a combination of columns and column ranges (e.g. 10,14-29,23-50,17).

Columns and column ranges may be repeated and may overlap each other.

If no column range is specified, the full length of each line is read.

KEY

EDT is to interpret the first 8 characters in each line as a key. Each record moved to the current work file or to a file opened by means of @OPEN receives this key as its line number. The key is not regarded as part of the line contents.

EDT checks that the first 8 characters in each line contain a valid key, i.e. that only the digits 0 to 9 appear here. If the key is not valid, @READ is rejected with an error message.

RECORDS

Permits the user to read a specified portion of a SAM file which has no keys. In this case, a virtual relationship between record numbers and line numbers is used, where 0.0001 stands for the first record in the file, 0.0002 for the second record, etc. The records read from the file are appended to the contents of the work file or of the file opened by means of @OPEN. Their line numbers in this file are calculated from the current line number and the current increment in this file.

If STRIP is not specified, RECORDS may be abbreviated as R.

STRIP

Deletes all trailing blanks from each line which is read in. If a line contains only blanks, all blanks except for the first one are removed.

If KEY or range* is specified without RECORDS, any lines which are shorter than 8 characters are ignored.

Line numbers or column numbers may be repeated in the specification for range* and col, respectively; this causes the specified lines or columns to be read in more than once.

Assignment of line numbers

The line numbers are assigned on the basis of the current line number and the current increment. In an empty work file, the current line number and the current increment are, by default, both 1. Either of these values can be changed by means of @SET In (inc). (See @SET, format 6.)

Example

```
SET 0.01;READ 'file'.
```

Before the file is read, the current line number is set to 0.01 and the increment is set (implicitly) to 0.01.



If the user attempts to read an ISAM file by means of @READ, EDT displays an error message and sets the switch for an EDT error. However, EDT still reads the specified file, since a @GET is issued internally for this file.

If "file" is specified, the abbreviation 'R' can still be used for @READ in F mode; otherwise @REA[D] must be used.

Interaction with XHCS

If the XHCS subsystem is installed, the coded character set name (CCSN) of the file is taken into account in a @READ statement.

The @READ statement is only executed if the CCSN of the file is the same as the CCSN currently selected in EDT or all work files are empty and the coded character set can be displayed on the data display terminal.

Example

```
23.00 .....  
read 'test.sam.f'.....0000.00:001(1)
```

The file TEST.SAM.F is to be read in.

```
22.00 .....  
% EDT4200 'OPEN': DMS ERROR CODE: 'ODC2'  
system.....0000.00:001(1)
```

The attempt to read file TEST.SAM.F is rejected with error message EDT4200.

In order to display the catalog entry for the file, the user switches to system mode:

```

/show-file-attributes test.sam.f,information=all
%00000114 :10SN:$USER.TEST.SAM.F
% ----- HISTORY -----
% CRE-DATE = 1994-08-30 ACC-DATE = 1994-10-07 CHANG-DATE = 1994-08-30
% CRE-TIME = 07:22:23 ACC-TIME = 10:34:09 CHANG-TIME = 07:22:26
% ACC-COUNT = 20 S-ALLO-NUM = 7
% ----- SECURITY -----
% READ-PASS = NONE WRITE-PASS = NONE EXEC-PASS = NONE
% USER-ACC = OWNER-ONLY ACCESS = WRITE ACL = NO
% AUDIT = NONE DESTROY = NO EXPIR-DATE = 1994-08-29
% SP-REL-LOCK= NO EXPIR-TIME = 23:00:00
% ----- BACKUP -----
% BACK-CLASS = A SAVED-PAG = COMPL-FILE VERSION = 1
% MIGRATE = ALLOWED
% ----- ORGANIZATION -----
% FILE-STRUC = SAM BUF-LEN = STD(1) BLK-CONTR = PAMKEY
% IO(USAGE) = READ-WRITE IO(PERF) = STD DISK-WRITE = IMMEDIATE
% REC-FORM = (V,N) REC-SIZE = 20
% ----- ALLOCATION -----
% SUPPORT = PUB S-ALLOC = 18 HIGH-US-PA = 114
% EXTENTS VOLUME DEVICE-TYPE EXTENTS VOLUME DEVICE-TYPE
% 8 10SN.4 D3480
% NUM-OF-EXT = 8
%PLEASE ACKNOWLEDGE
    
```

```

%:10SN: PUBLIC: 1 FILE RES= 3 FREE= 2 REL= 0 PAGES
/set-file-link link-name=edtsam,file-name=test.sam.f, -
/ record-format=fixed(record-size=20)
/resume-program
    
```

The SHOW-FILE-ATTRIBUTES command is used to request and display the catalog entries for file TEST.SAM.F. The file has a fixed record length of 20 bytes.

By means of the SET-FILE-LINK command, the file link name EDTSAM is assigned to file TEST.SAM.F and a fixed record length of 20 bytes is specified for this file.

The RESUME-PROGRAM command switches back to program mode; EDT is still loaded.

```

23.00 .....
read 'test.sam.f' .....0000.00:001(1)
    
```

The file TEST.SAM.F is to be read in.

```

1.00 THIS SAM FILE.....
2.00 HAS A FIXED.....
3.00 RECORD LENGTH;.....
4.00 EACH RECORD IS.....
5.00 20 BYTES LONG.....
6.00 .....

delete ; read '/'.....0001.00:001(1)

```

Assigning the file link name EDTSAM has made it possible to read in the file TEST.SAM.F. The work file is now to be deleted and the file TEST.SAM.F is to be read in again. Instead of the file name TEST.SAM.F, the user may simply specify '/'.

```

1.00 THIS SAM FILE.....
2.00 HAS A FIXED.....
3.00 RECORD LENGTH;.....
4.00 EACH RECORD IS.....
5.00 20 BYTES LONG.....
6.00 .....

write 'test.sam.new'.....0001.00:001(1)

```

The file TEST.SAM.F has been read in again. The current work file is now to be written into the file TEST.SAM.NEW.

```

22.00 .....
% EDT4900 A FILE COMMAND IS IN EFFECT
system '/remove-file-link edtsam' ; write 'test.sam.new'.....0000.00:001(1)

```

As long as the file link name EDTSAM is assigned to a file, it is not possible to write into any other file. The assignment of the file link name EDTSAM to the file TEST.SAM.F is now canceled and the current work file is to be written into the file TEST.SAM.NEW. A new SAM file is now created.

@RENUMBER Renumber lines

@RENUMBER is used to renumber the lines created in a virtual file. The line number at which renumbering is to start and the desired increment may be specified.

The new current line number is equal to the highest line number after renumbering plus the current increment. In the screen window in which @RENUMBER was entered the file section in the data window remains untouched; all that changes is the number display.

Operation	Operands	F mode / L mode
@RENUMBER	[In [(inc)]]	

- In The line number (e.g. 5) at which renumbering is to start. The minimum value is 0.0001, the maximum value 9999.9999. In may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %,\$).
- inc The increment for calculating the new line numbers. The minimum value is 0.0001, the maximum value 9999.9999. If In and inc are omitted, the default increment is 1.

If no operand is specified, the default values for the line number and the increment are both 1.

@RENUMBER must not be used for a file opened by means of @OPEN.

The line present in the copy buffer are not renumbered (see [section "Statement codes in F mode"](#), ["C Mark for copying" on page 90ff.](#) and ["R Mark for copying \(without clearing copy buffer\)" on page 103ff.](#)).

If this statement is issued in interactive mode and would entail the loss of some lines because they would exceed the highest permissible line number, the following message will be output:

```
% EDT0910 '@RENUMBER': LINES WILL BE LOST
% EDT0911 CONTINUE PROCESSING? REPLY (Y=YES; N=NO)
```

N: @RENUMBER is not executed.

Y: @RENUMBER is executed and the following message is output:

```
% EDT2904 MAXIMUM LINE NUMBER WHEN PROCESSING '@RENUMBER'.
SOME LINES ARE LOST.
```

The F-mode window continues to display the same section, unless the lines in the work window are lost in the course of renumbering. In this case, the work window is positioned at the last line.

Calculation of the increment

If `ln` is specified and `inc` is omitted, the increment depends on the number of decimal positions in the value specified for `ln`. `ln = 2`, for example, results in an increment of 1, while `ln = 2.4` implies an increment of 0.1, `ln = 2.40` implies 0.01, etc.



- The entries in the three-level EDT stack (see also @) are not modified by the @RENUMBER statement. After renumbering, however, the use of these stack entries will in most cases not be advisable.
- Information on the number of lines in the current work file can be requested with @LIMITS.

@RESET Reset EDT and DMS error switches

@RESET resets the EDT and DMS error switches (see also @IF, format 1).

Operation	Operands	F mode / L mode / @PROC
@RESET		

@RETURN Terminate screen dialog and abort procedures

@RETURN can be used to

- terminate EDT
- terminate the screen dialog after @DIALOG
- abort procedures (@DO and @INPUT procedures)
- terminate the screen dialog if EDT was called as a subroutine.

Operation	Operands	F mode / L mode
@RETURN	[message]	

message Freely selectable text.

The end of the statement is also the end of the message text. There must be at least one blank between @RETURN and “message”.

“message” may be used only if EDT was called via the subroutine interface.

If there are still work files which have not been saved, the numbers of these files are displayed after the message: % EDT0900 EDITED FILE(S) NOT SAVED!

This is accompanied by one of the following items, if available:

- a local @FILE entry
 - defined explicitly by @FILE, or
 - defined implicitly by @READ, @GET, @OPEN,
- the library and element name of
 - a library element opened by means of @OPEN (format 2) or
- the file name of
 - a SAM or ISAM file opened with @OPEN (format 2) or
 - a POSIX file opened with @XOPEN.

The user then receives the following query:

```
% EDT0904 TERMINATE EDT? REPLY (Y=YES, N=NO)?
```

N: In F mode the work window is displayed again; in L mode the prompting message. The user can close any files with unsaved data and write them back.

Y: Virtual files with unsaved data are lost. EDT is terminated and the specified program started.

The save query can be suppressed by setting task switch 4 before EDT is called.

@RETURN after a screen dialog started using START-PROGRAM \$EDT

If the screen dialog was started using START-PROGRAM \$EDT, @RETURN has the same effect as @HALT. EDT is terminated.

@RETURN after a screen dialog started using @DIALOG

If the screen dialog was started using @DIALOG, @RETURN causes the operation interrupted by @DIALOG (L mode dialog or reading from SYSDTA) to be resumed.

EDT does not issue the save query.

@RETURN after a screen dialog when EDT was called as a subroutine

If @RETURN is entered after EDT has been called as a subroutine, control is returned to the user program which called EDT.

A message can be passed to the calling program by means of the "message" operand and the calling program can evaluate this message when it again receives control. The message text is passed in the message field of control block EDTGLCB (see the manual "EDT Subroutine Interfaces" [1]).

@RETURN in EDT procedures

If @RETURN is used in @DO or @INPUT procedures, execution of the procedure is aborted and control is returned to the point where the procedure call was issued.

Example 1

```

1.   @PROC 6 ----- (01)
1.   @ @SET #S1 = 'THIS IS #S1'
2.   @ @SET #S2 = 'THIS IS #S2'
3.   @ @PRINT #S1
4.   @ @RETURN ----- (02)
5.   @ @PRINT #S2
6.   @END ----- (03)
1.   @DO 6 ----- (04)
    #S01 THIS IS #S1

```

- (01) Work file 6 is opened for processing.
- (02) If this statement is encountered during execution of the procedure in work file 6, all subsequent statements will be ignored.
- (03) Processing of work file 6 is terminated.
- (04) The procedure is executed.

Example 2

```

1.      AAAA
2.      BBBB
3.      CCCC
4.      @PROC 7 ----- (01)
1.      @ @PRINT ! ----- (02)
2.      @ @RETURN ----- (03)
3.      @END
4.      @DO 7,!=%, $ ----- (04)
1.0000 AAAA
4.

```

- (01) The procedure is created in work file 7.
- (02) During execution of the statements in the procedure, the line addressed via the loop symbol ! is to be output.
- (03) Execution of the statements in the procedure is aborted at this point, regardless of whether or not the loop symbol ! has already reached the specified upper limit.
- (04) The procedure is called with the limits 1 and 3 (%=1, \$=3) specified for the loop symbol !. However, due to the @RETURN statement in the procedure, the loop symbol value is never incremented.

@RUN Call user program as subroutine

@RUN loads a user program as a subroutine and starts it. The user program must exist in the form of an object module in a library (see the manual “EDT Subroutine Interfaces” [1]). Unlike program calls using @LOAD or @EXEC, EDT remains loaded and the contents of the virtual files are retained.

If a user program that can only run in 24-bit addressing mode is to be executed, EDT must be loaded via the driver module EDTC (section “Calling EDT” on page 33ff).

The user program called in this manner may process the lines in the current work file, in which @RUN was entered.

@RUN is one of the EDT statements that is relevant to security (see section “Data protection” on page 71). In uninterruptible system procedures in interactive mode and in the case of input from a file, the statement will be rejected (unless it is read from SYSDTA=SYSCMD).

Operation	Operands	F mode / L mode
@RUN	(entry [,modlib]) [:string] [[,]UNLOAD]	

- entry The name
- of a control section (CSECT),
 - of an entry point (ENTRY) in a program.
- The corresponding object module or load unit is loaded. “entry” may also be specified by means of a string variable.
- modlib The name of the library in which the object module or load unit is stored. modlib may also be specified by means of a string variable. If this operand is omitted, the system searches for the module or load unit in the module library with the default name EDTRUNLB.
- string A string which is to be passed to the program being called. The string may be specified:
- explicitly, enclosed in single quotes, or
 - implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).
- If “string” is specified, EDT passes
- the address of the first byte of the string in register 2
 - the string length minus 1 in register 3.
- UNLOAD Specifies that the module is to be unloaded when control is returned to EDT. This may be used if the name of the module is the same as the name of its entry point (ENTRY).

@RUN must not be used if there is a file which has been opened by means of @OPEN.

@SAVE Write as ISAM file

@SAVE writes all or part of the contents of the current work file to disk as an ISAM file.

The ISAM file is open only during execution of the @SAVE statement.

Operation	Operands	F mode / L mode
@SAVE	['file'] [(ver)] [range*] [:col:] [{ UPDATE } [{ RENUMBER [ln [(inc)]]] [OVERWRITE] }]	

- file** A file name.
 If the operand “file” is omitted, the explicit local @FILE entry is used as the file name; if this does not exist, the global @FILE entry is used, and if this does not exist either, then the implicit local @FILE entry is used (see also @FILE). Otherwise, @SAVE is rejected with an error message.
 If the file link name EDTISAM is assigned to a file, the user may simply specify ‘/’ in order to write this file back to disk (see [section “File processing” on page 49ff](#)).
- ver** The version number of the file.
 This may consist of up to three digits or an asterisk (*). * designates the current version number. If an incorrect version number is specified, the correct version number is displayed.
 Version numbers are intended to prevent the inadvertent overwriting of files. A new version number is generated when a file is created for the first time or when an existing file is updated: the version number of a new file is 1 and the version number of an updated file is incremented by 1.
 The highest possible version number is 255; if a further update is executed, the new version receives the version number 0.
- range*** A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.
- If range* is omitted, all lines of the current work file are written into the ISAM file.

- col** A column range, specified as:
- one or more columns, separated by commas (e.g. 10,15,8)
 - one or more column ranges, separated by commas (e.g. 15-25,18-23)
 - a combination of columns and column range (e.g. 10,14-29,23-50,17).
- Columns and column ranges may be repeated and may overlap each other.
- If no column range is specified, the full length of each line is written to the ISAM file.
- UPDATE** This is meaningful only if there is already an ISAM file with the specified name.
- UPDATE causes the lines to be saved to be inserted into the ISAM file. EDT overwrites only those lines in the ISAM file whose numbers also exist within the specified range* in the work file, virtual file or file opened by means of @OPEN. All other lines in the ISAM file remain unchanged.
- RENUMBER** The lines to be saved are to be renumbered.
- The line numbers in the work file, virtual file or file opened by means of @OPEN remain unchanged.
- If RENUMBER is not specified, the ISAM keys are generated from the line numbers of the lines to be saved. When the file is again read into the work file, virtual file or file opened by means of @OPEN, these line numbers are retained if the operand NORESEQ is specified in the @GET statement.
- In** The starting value for the ISAM keys in the file to be written. The minimum value is 0.0001, the maximum value 99.9999. If inc is omitted, the increment is implied by the number of decimal positions in In. 5, for example, implies an increment of 1, while 5.0 implies an increment of 0.1.
- In may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %,\$).
- If In is not specified, the starting value for the ISAM key is 0001.0000.
- inc** The increment for the ISAM key.
- The minimum value is 0.0001, the maximum value 9999.9999. If In and inc are omitted, the increment for the ISAM key is 1.
- OVERWRITE** Suppresses the query OVERWRITE FILE? (Y/N).
- Any existing file with the same name is overwritten. If there is no file with the specified name, OVERWRITE has no effect.

If neither UPDATE nor OVERWRITE is specified, and there is already a file with the same name, EDT asks:

```
% EDT0903 FILE 'file' IS IN THE CATALOG, FCBTYPE = fcctype  
% EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO)
```

If the user responds with

N @SAVE is not executed;

Y @SAVE is executed and the existing file is overwritten as an ISAM file by the contents of the current work file.

In the case of variable-length records (RECORD-FORMAT=VARIABLE; [section "Processing ISAM files with nonstandard attributes" on page 51](#)), the records as of position 257 are lost when the file is written back.

Interaction with XHCS

If the XHCS subsystem is installed, the @SAVE statement also transfers a coded character set name (CCSN) as a code attribute after the file has been written back.

@SAVE assigns the CCSN currently valid in EDT regardless of whether the file already exists and what CCSN it has.

@SDFTEST Start SDF syntax check on data lines

@SDFTEST

- passes the contents of a line or line range to SDF for a syntax check,
- transfers the string returned by SDF to the data area and
- can be used to set a program name for checking statements.

In batch mode, this statement is ignored instead of being executed.

Operation	Operands	F mode / L mode
@SDFTEST	[range*] [PROGRAM [=structured-name [{ INTERNAL } { EXTERNAL }]]]

range* A line range consisting of one or more lines and/or line ranges which can also be specified using symbols.
String variables cannot be specified.
All lines of the line range which begin with a / are passed together with their continuation lines to SDF for a syntax check.
A continuation line is expected when a line ends with – (apart from any blanks or null characters).
A maximum of 255 continuation lines are permitted.
If range* is not specified, all the lines of the work file are checked.

PROGRAM Data lines beginning with // are also passed to SDF as statements.

=structured-name

Name of the program whose statements are to be checked. To check the syntax, SDF uses the current syntax hierarchy in which the statements of the program must be described.
If this operand is not specified, EDT uses the program name set by an earlier @SDFTEST statement with PROGRAM=structured-name or preset using @PAR SDF-PROGRAM=structured-name. If no name has been preset, an error message is issued.

INTERNAL

Program name is internal name, maximum 8 characters (e.g. \$LMSSDF, BINDER, \$SDAEDXT).

EXTERNAL

Program name is external name, maximum 30 characters
(e.g. LMS, SDF-A, HSMS).

If the PROGRAM operand is not specified, the specified line range is checked only in regard to command syntax, i.e. lines which do not begin with a single / are not checked.

After SDF has checked the syntax, the format of the command or statement returned by SDF is written back to the same location in the data area, depending on the setting of the SDF options. Where necessary, continuation lines are inserted or deleted.

If the INTERNAL or EXTERNAL operand is omitted, the name type set with a previous @SDFTEST with INTERNAL or EXTERNAL is used. If there was no such previous statement, the name type used is the one set with @PAR SDF-NAME-TYPE = <type>. The default type is INTERNAL when EDT is started.

Note

The external program name can only be specified with SDF versions as of V04.4. If the external program name is specified while an earlier SDF version is in use (explicitly in the @SDFTEST statement or in the @PAR SDF-NAME-TYPE statement), the following EDT message is output when @SDFTEST is called, and processing is terminated:

```
% EDT5323 SDF: EXTERNAL PROGRAM NAME NOT SUPPORTED
```

Continuation lines in the output

If the output is more than 71 characters long, it extends into one or more continuation lines. The continuation character is set in the 72nd column. Where necessary, subsequent lines are renumbered.

In F-mode, the following message is issued:

```
% EDT0285 SDF: SYNTAX TESTED. '0' ERROR(S) IN RANGE.
```

Response when SDF detects a syntax error

If GUIDANCE=MIN/MED/MAX is set and SDF detects an error during the check, the program branches to SDF's guided correction dialog, and the user can correct the command or statement.

If no correction dialog is possible because GUIDANCE=NO/EXPERT is set or if the user aborts the dialog by pressing **[F1]**, EDT issues the following error message:
% EDT4310 SDF: SYNTAX ERROR AT LINE (&00)

If there are more data lines to be processed, the user is then asked whether processing is to be continued:

```
% EDT0911 CONTINUE PROCESSING? REPLY(Y=YES; N=NO)
```

Y: the program continues with the processing of further lines.

N: processing is aborted.

In F mode, the faulty line is displayed at the top of the screen.



The @STATUS=SDF statement can be used to query the names of the currently set syntax files and the predefined internal program name.

The program name can be preset with the @PAR SDF-PROGRAM=... statement. Specifying a program name explicitly in @SDFTEST overwrites the presetting.

SDF-A can be used to determine the internal program name if it does not match the name of the program.

If the GUIDANCE setting is MIN, MED or MAX, passwords and other operands defined with OUTPUT=SECRET-PROMPT are replaced with P.

Ampersand (&) replacement is accepted only in operand values, not in marks or in command, statement or operand names. Partial replacement of operand values is not possible.

@SEARCH-OPTION Set default value for searching with @ON

The @SEARCH-OPTION statement sets a default value stipulating whether or not a search for strings with @ON is to distinguish between uppercase and lowercase letters.

Operation	Operands	F mode / L mode
@SEA[RCH]- OPTION	CASELESS-SEA[RCH] {=ON =OFF}	

ON The search with @ON does not consider whether the characters in the search string match in terms of uppercase and lowercase notation. In other words, if you are searching for 'string', the strings 'String', 'STRING' and 'STring' are also reported as hits.

OFF When searching for a character, a distinction is made between uppercase and lowercase notation. This is preset when starting EDT.

The allocation of uppercase and lowercase letters corresponds to the conversion table for @LOWER OFF.

If XHCS is installed in the system, the conversion table associated with the coded character set (CCS) is used for the allocation of uppercase and lowercase letters.

If XHCS is not installed or if a 7-bit terminal is being used, EDT performs conversion on the basis of EBCDIC.DF.03. In this case, no allocation is made for the German umlaut characters ä, ö, ü and Ä, Ö, Ü.

Activating the coding function (@CODE) has no influence on the allocation of uppercase and lowercase letters.

@SEPARATE Perform line break

The @SEPARATE statement breaks a line or range of line into several parts.

The point at which the break takes place is specified:

- by a separator character
- by a column position
- by the record separator character (SEPARATOR) preset by means of the @PAR statement

Operation	Operands	F mode / L mode
@SEPARATE	[range*] [AT { 'char' } X'hex' cl]	

- range*** Line range for which a line break is to be performed.
 It is also possible to specify the line range by the current line range symbol (see @RANGE), by symbolic line numbers (e.g. %, \$) or by line number variables.
 Character string variables must not be specified.
 If range* is not specified, the line break covers all lines in the file.
- AT** Definition of the point where the line break takes place.
 If AT is not specified, the record separator character preset by the @PAR SEPARATOR statement determines where the line break is implemented. If no record separator character is preset, an error message is issued.
- 'char'** A freely selectable character used as the separator character to determine where the line break is implemented.
 The character must be specified in single quotes. The quotes can be redefined by means of @QUOTE.
 There can be more than one separator character in a single line.
 Lines are shortened to before char. The part following char is inserted in the file as a new line.
 If char is the last character in a line or if it is followed by further separator characters, line numbers are reserved but no empty records (record length = 0) are created.

X'hex'	<p>A freely selectable character in hexadecimal format which is used as the separator character to determine where the line break is implemented.</p> <p>Lines are shortened to before hex. The part following hex is inserted in the file as a new line.</p> <p>If hex is the last character in a line or if it is followed by further separator characters, line numbers are reserved but no empty records (record length = 0) are created.</p>
cl	<p>Column number.</p> <p>Any characters after this position are separated from the line and inserted in the file as a new line.</p> <p>If the line is shorter than cl, the line is not changed. This also applies in the case of cl=1.</p>

Calculation of line numbers

When lines are inserted, they are numbered according to one of three methods:

1. Default numbering with the default increment 1.0000 (e.g. 21.0000, 22.0000, 23.0000 ... 99.0000) or
2. Numbering with a fixed increment as defined by @PAR INCREMENT or
3. Automatic numbering and renumbering
if the selected increment was too large to accommodate the new lines. In such cases, EDT selects an increment that is smaller by a factor of ten than the default increment (see 1 above) or than the defined increment (see 2. above). When renumbering, this smaller increment is used.
This procedure is repeated until all the lines have been successfully inserted or until EDT has selected the minimum increment of 0.01.

Renumbering with @PAR RENUMBER=ON:

If it is still not possible to insert the lines using the smallest possible increment (0.01), EDT automatically renumbers the lines that already exist after the target point using the same increment.

If it is not possible to find enough space, no lines are inserted and an error message is output.

If existing lines are not to be renumbered, @PAR RENUMBER=OFF must be set.

If a line with a number greater than the highest existing line number is created, the current line number is updated.

Example 1

A printout is to be made narrow:

@SEPARATE 5-100 AT 41

(Lines 5 to 100 are shortened to a length of 40 characters. The remaining portions of each line are inserted in the file after the newly shortened line.)

Example 2

Records contain end-of-record characters which are to be evaluated:

@SEPARATE & AT X'15'

@SEQUENCE Generate or check line numbers

The @SEQUENCE statement:

- writes a number into each line in a specified line range. These numbers are in ascending order (format 1).
- writes the associated line number into each line of a specified line range (format 2).
- examines the contents of the specified column(s) in each line of a specified line range. It interprets the string in the column(s) as a binary number on the basis of the EBCDIC codes for the characters and checks whether these binary numbers are in ascending order (format 3).

@SEQUENCE (format 1) Number lines

@SEQUENCE, format 1, causes EDT to write a number into each line of the specified line range. These numbers are in ascending order.

EDT overwrites any characters which already exist in the columns into which it writes the numbers.

Operation	Operands	F mode / L mode
@SEQUENCE	[rng] [:[cl] [:[[n1] (n2)]]]	

rng

A line range, specified as:

- a single line (e.g. 6)
- several contiguous lines (e.g. 8-20).

A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.

If rng is not specified, EDT writes a number into each line of the virtual file or the file opened by means of @OPEN.

cl

The column into which the first digit of the number is to be written. If cl is omitted, EDT places the first digit of the number in column 73.

n1

The decimal integer which EDT is to write into the first line of the specified line range. n1 may have up to 8 digits and the numbers written into the following lines have the same number of digits as n1. The value of n1 is freely selectable.

If n1 is not specified, EDT writes the number 00000100 into the first line of the line range.

n2 An integer specifying the increment for the following lines. The number written into each line consists of the number in the previous line plus this increment.
 If n2 is not specified, EDT uses an increment of 100.

@SEQUENCE (format 2) Adopt line numbers

@SEQUENCE, format 2, causes EDT to write the associated line number into each line of the specified line range. This line number is written as an 8-digit number without a decimal point. If necessary, the number is padded on the right and left with zeros.

In line 12.345, for example, EDT writes the number 00123450. When writing the number, EDT overwrites any other characters which already exist in the 8 columns it uses for the line number.

Operation	Operands	F mode / L mode
@SEQUENCE	[rng*] : [cl] : LINE	

rng* A line range, specified as:
 – a single line (e.g. 6)
 – several contiguous lines (e.g. 8-20).

The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables must not be used.

If rng* is not specified, EDT writes the associated number into each line of the virtual file or the file opened by means of @OPEN.

cl The column into which the first digit of the number is to be written. If cl is omitted, EDT places the first digit of the number in column 73.

Example

```

0.00 THE LINE.....
1.11 NUMBER.....
88.76 SEQUENCE.....
88.76 IS.....
5555.00 OFTEN.....
9876.54 IMPORTANT.....
9877.54 .....

sequence :20: line .....0000.00:001(0)

```

The associated line number is to be written into each line of the work file, starting at column 20.

```

0.00 THE LINE          00000035.....
1.11 NUMBER           00011110.....
88.76 SEQUENCE        00887610.....
88.76 IS              00887620.....
5555.00 OFTEN         55550000.....
9876.54 IMPORTANT     98765432.....
9877.54 .....

```

The line numbers are written as 8-digit numbers without a decimal point, starting at column 20. Wherever necessary, leading and trailing zeros are added.

@SEQUENCE (format 3) Check line numbers

@SEQUENCE, format 3, causes EDT to examine the contents of one or more columns in each line of the specified line range. EDT interprets the string in these columns as a binary number on the basis of the EBCDIC code for each character in the string. If any of the columns to be examined lies to the right of the end of the line, EDT assumes that this column contains the value X'40'.

EDT checks whether the binary numbers in the lines are in ascending order: it displays each line in which it finds a binary number which is less than or equal to the binary number in the preceding line.

Operation	Operands	F mode / L mode
@SEQUENCE	[rng] : [cl] : CHECK [int]	

- rng** A line range, specified as:

 - a single line (e.g. 6)
 - several contiguous lines (e.g. 8-20).

A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.

If rng is not specified, EDT checks each line in the main file.
- cl** The column containing the first digit of the number to be checked. If cl is omitted, EDT starts checking at column 73.
- int** Specifies how many columns (1-8) are to be checked. If int is omitted, EDT checks eight columns.

Example

```

1.00 15 LINE 1.....
2.00 20 LINE 2.....
3.00 21 LINE 3.....
4.00 16 LINE 4.....
5.00 18 LINE 5.....
6.00 01 LINE 6.....
7.00 99 LINE 7.....
8.00 97 LINE 8.....

sequence & :1: check 2 .....0001.00:001(0)

```

EDT is to check all lines in the file to determine whether the contents of columns 1 and 2 are in ascending order.

```

4.0000 16 LINE 4
6.0000 01 LINE 6
8.0000 97 LINE 8
PLEASE ACKNOWLEDGE

```

EDT displays all lines which are not in ascending order.


```

1.00 15 LINE 1.....
2.00 20 LINE 2.....
3.00 21 LINE 3.....
4.00 16 LINE 4.....
5.00 18 LINE 5.....
6.00 01 LINE 6.....
7.00 99 LINE 7.....
8.00 97 LINE 8.....

```

```

sequence 1-4 :1: check 1 .....0001.00:001(0)

```

Here, only the contents of the first column of line 1 to 4 are to be checked.

```

3.0000 21 LINE 3
4.0000 16 LINE 4
PLEASE ACKNOWLEDGE

```

The sequence is 1 (line 1), 2 (line 2), 2 (line 3) and 1 (line 4). Note, in particular, that two lines with the same number are regarded as not being in ascending order, as in line 3 in this example.

@SET Supply values for EDT variables

@SET consists of 6 formats, offering the following functions:

Assigning values to integer variables (format 1)

- assigns an integer expression to an integer variable
- assigns a printable number as an integer to an integer variable
- assigns the contents of a line number variable as an integer to an integer variable
- assigns the length of a line to an integer variable
- assigns the EBCDI code of a character string to an integer variable

Assigning values to string variables (format 2)

- assigns a string to a string variable
- assigns a line number, the contents of an integer variable or the name of a string variable to a string variable
- converts the contents of an integer variable into a printable number and places the result in a string variable
- converts a line number into printable form and stores the result in a string variable
- places the name of a string variable in a string variable

Assigning values to line number variables (format 3)

- assigns a line number to a line number variable
- converts the contents of an integer variable into a line number and assigns this value to a line number variable
- assigns a printable number as a line number to a line number variable
- assigns the internal representation of a character string to a line number variable

Placing values in lines (format 4)

- places the contents of an integer variable in a line in printable form
- writes the name of a string variable into a line
- converts the contents of a line number variable into printable form and places it in a line

Specifying date and time (format 5)

- places the date or the time of day in a string variable
- places the date or the time of day in a line

Specifying the new current line number and increment (format 6)

- specifies a new current line number and the increment

@SET (Format 1) Assigning values to integer variables

This format of @SET is used to

- assign an integer expression to an integer variable.
- assign a printable number as an integer to an integer variable.
- convert the contents of a line number variable into an integer and assign it to the specified integer variable as a value.
- determine the length of a line and assign it to an integer variable as a value. If the specified line does not exist, the integer variable is assigned the value 0.
- assign the EBCDI code of a specified character string to an integer variable.

Operation	Operands	F mode / L mode / @PROC
@SET	$\text{int-var} = \left\{ \begin{array}{l} [+ -] \text{ int } [+ - \text{ int}] \dots \\ \text{SUBSTR string} \\ \text{ln-var} \\ \text{LENGTH line} \\ \text{STRING string} \end{array} \right\}$	

- int-var** The integer variable (#I0 to #I20) to which a value is to be assigned.
- +/-** Arithmetical operators for the processing of the specified int values.
- int** An unsigned integer (e.g. 5, 0 or 17) or one of the integer variables #I0 to #I20 which is to be assigned (after any specified arithmetical operations with other int values) to the integer variable int-var. If the arithmetic operations result in a value that exceeds either of the permissible limits for an integer variable ($2^{31} - 1, -2^{31}$), EDT issues the error message OVERFLOW ERROR.
- ...** Indicates that several int values, linked by + or -, may be specified. The only restriction is the maximum line length of 256 characters.
- string** A character string. This string may be specified:
- explicitly, enclosed in single quotes, or
 - implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).
- Any blanks in the string are suppressed during conversion.

SUBSTR string

Specifies that EDT is to assign a printable number to an integer variable. The printable number 17, for example, is assigned as the integer 17.

If “string” does not contain a number, @SET is rejected with an error message. Any plus or minus sign in the string is taken into account during conversion. If there is no sign, EDT assumes that the sign is +.

In-var

Specifies one of the 21 line number variables (#L0 to #L20). The maximum possible value for a line number variable is 9999.9999, the minimum 0.0001. For conversion into an integer, the contents of the line number variable are multiplied by 10000 and leading zeros added so that the resulting number has 10 digits. This is necessary so that the digits after the decimal point in a line number are taken into account. The line number 1.23 is converted into the integer 12300. After adding leading zeros, the value 0000012300 is assigned to the integer variable.

LENGTH

Specifies that EDT is to place the length of a line in an integer variable.

line

The line number of the line whose length is to be placed in the integer variable (e.g. 15.34 or #S11 or #S11 + #I1).

STRING string

Specifies that EDT is to place the EBCDI code of a string in the form of an integer in an integer variable.

Example: string = '1', internal representation X'F1', the value assigned to int-var is thus 241. The EBCDI code of “string” is assigned to the integer variable. If “string” has less than four characters, leading zeros are placed in int-var. If “string” has more than four characters, only the first four characters are used.

Example 1

Assign an integer expression to an integer variable.

```

1.      @SET #I0 = -1 ----- (01)
1.      @SET #I1 = #I0 + 1001 ----- (02)
1.      @SET #I2 = #I1 + #I1 -#I0 + 3 - #I0 ----- (03)
1.      @SET #I3 = #I2 + #I2 + #I2 + #I2 + #I2 ----- (04)
1.      @STATUS = I ----- (05)
#I00= 0000000001 #I01= 0000001000 #I02= 0000002005
#I03= 0000010025 #I04= 0000000000 #I05= 0000000000
#I06= 0000000000 #I07= 0000000000 #I08= 0000000000
#I09= 0000000000 #I10= 0000000000 #I11= 0000000000
#I12= 0000000000 #I13= 0000000000 #I14= 0000000000
#I15= 0000000000 #I16= 0000000000 #I17= 0000000000
#I18= 0000000000 #I19= 0000000000 #I20= 0000000000

```

- (01) The value -1 is assigned to the integer variable #I0.
- (02) The expression #I0 + 1001 is assigned to #I1. Since #I0 contains -1, #I1 is set to -1 + 1001, i.e. 1000.
- (03) The same integer variable may be specified several times in one expression.
- (04) Multiplication can be implemented by adding a variable to itself the required number of times.
- (05) The contents of the integer variables are to be displayed.

Example 2

Assign a printable number as an integer to an integer variable.

```

1.      @CREATE 2:'AA1234567890BB'
1.      @CREATE #S2: 'AT 16.05 WE GO HOME' ----- (01)
1.      @SET #I6 = 3
1.      @SET #L1 = 2
1.      @SET #I7 = SUBSTR X'FOF1F2F3' ----- (02)
1.      @SET #I8 = SUBSTR B'11110001'*6 ----- (03)
1.      @SET #I9 = SUBSTR 2:3: ----- (04)
1.      @SET #I10 = SUBSTR 2:5-8,4,3,3,3: ----- (05)
1.      @SET #I11 = SUBSTR #S2: 4-5, 7-8: ----- (06)
1.      @SET #I12 = SUBSTR '123'*3 ----- (07)
1.      @SET #I13 = SUBSTR #S5-#I6:8,8,8,8,8,8,8: ----- (08)
1.      @SET #I14 = SUBSTR #L1:3-6,3-6: ----- (09)
1.      @STATUS = I ----- (10)
#I00= 0000000000 #I01= 0000000000 #I02= 0000000000
#I03= 0000000000 #I04= 0000000000 #I05= 0000000000
#I06= 0000000003 #I07= 0000000123 #I08= 0000111111
#I09= 0000000001 #I10= 0034562111 #I11= 0000001605

```

```
#I12= 0123123123 #I13= 0005555555 #I14= 0012341234  
#I15= 0000000000 #I16= 0000000000 #I17= 0000000000  
#I18= 0000000000 #I19= 0000000000 #I20= 0000000000  
1.
```

- (01) Values are placed in line 2, string variable #S2, integer variable #I6 and line number variable #L1.
- (02) X'F0F1F2F3' is equivalent to the printable number 0123 and this number is assigned to the integer variable #I7.
- (03) B'11110001'*6 is equivalent to X'F1'*6, to '1'*6 and also to the printable number 111111. This value is placed in #I8.
- (04) Column 3 of line 2 contains the printable digit 1 and #I9 is set to this value.
- (05) The printable digits in columns 5 to 8, column 4 and column 3 (repeated 3 times) in line 2, chained together, result in the printable number 34562111 and #I10 is set to this value.
- (06) Columns 4 to 5 and 7 to 8 of string variable #S2 result in the printable number 1605, which is assigned to integer variable #I11.
- (07) '123'*3 is equivalent to '123123123'; this number is assigned to #I12.
- (08) #I6 contains the value 3, which means that #S5-#I6 addresses string variable #S2. If the contents of column 8 of string variable #S2 are repeated 7 times, the result is the number 5555555, which is assigned to #I13.
- (09) Line 2 is addressed via line number variable #L1. Columns 3 to 6 of line 2, repeated twice, result in the printable number 12341234, which is assigned to #I14.
- (10) The contents of integer variables #I0 to #I20 are displayed.

Example 3

Convert the contents of a line number variable into an integer and assign this integer to an integer variable.

```

1.      @SET #L0 = 0.0001 ----- (01)
1.      @SET #I15 = #L0 ----- (02)
1.      @STATUS = #I15 ----- (03)
#I15= 0000000001
1.      @SET #L1 = 55.5555 ----- (04)
1.      @SET #I16 = #L1 ----- (05)
1.      @STATUS = #I16 ----- (06)
#I16= 0000555555
1.      @SET #L2 = 9999.9999 ----- (07)
1.      @SET #I17 = #L2 ----- (08)
1.      @STATUS = #I17 ----- (09)
#I17= 0099999999
1.
```

- (01) Line number variable #L0 is set to 0.0001.
- (02) Integer variable #I15 is set to the value of #L0 multiplied by 10000, i.e. 1.
- (03) The contents of #I15 are displayed.
- (04) #L1 is set to 55.5555.
- (05) #I16 is set to the value of #L1 multiplied by 10000.
- (06) The value of #I16, namely 555555, is displayed.
- (07) #L2 is set to 9999.9999.
- (08) #I17 is set to 99999999.
- (09) The contents of #I17 are displayed.

Example 4

Assign the length of a line to an integer variable.

```

1.      @CREATE 15.34: 'AB'*23
1.      @PRINT 15.34 ----- (01)
15.3400 ABABABABABABABABABABABABABABABABABABABABABABABABAB
1.      @SET #I10 = LENGTH 15.34
1.      @STATUS = #I10 ----- (02)
#I10= 0000000046
1.      @DELETE 15.34
1.      @SET #I10 = LENGTH 15.34 ----- (03)
1.      @STATUS = #I10
#I10= 0000000000

```

- (01) Line 15.34 is created and displayed.
- (02) #I10 is set to the length of line 15.34 and its contents are then displayed.
- (03) Line 15.34 is deleted.
#I10 is again set to the length of line 15.34 (which has now been deleted) and its contents are again displayed.

Example 5

Assign the EBCDI code of a character string to an integer variable.

```

1.      @CREATE 1: '1234567' ----- (01)
1.      @CREATE #S5: 'CBA'
1.      @SET #I0 = STRING X'34'
1.      @SET #I1 = STRING 'A'
1.      @SET #I2 = STRING X'7D5'
1.      @SET #I3 = STRING B'00001111' ----- (02)
1.      @SET #I4 = STRING 1:3,2:
1.      @SET #I5 = STRING #S5:3,3:
1.      @SET #I6 = STRING 1:2-3,1:
1.      @STATUS = I ----- (03)
#I00= 0000000052 #I01= 0000000193 #I02= 0000002005
#I03= 0000000015 #I04= 0000062450 #I05= 0000049601
#I06= 0015922161 #I07= 0000000000 #I08= 0000000000
#I09= 0000000000 #I10= 0000000000 #I11= 0000000000
#I12= 0000000000 #I13= 0000000000 #I14= 0000000000
#I15= 0000000000 #I16= 0000000000 #I17= 0000000000
#I18= 0000000000 #I19= 0000000000 #I20= 0000000000

```

- (01) Line 1 and string variable #S5 are created.
- (02) The integer variables #I0 to #I6 are set in various ways.
- (03) The contents of all integer variables are displayed.

@SET (Format 2) Assigning values to string variables

This format of @SET is used to:

- assign a character string to a string variable. The previous value of the string variable is overwritten, i.e. deleted.
- assign the contents of an integer variable, a line number or the name of a string variable to a string variable. EDT writes the value to the string variable as it is represented internally.
- convert the contents of an integer variable into a printable number and place the result in a string variable starting at a specific column.
- place the printable form of a line number stored in a line number variable in a string variable starting at a specified column.
- place the name of a string variable in a string variable starting at a specified column.

Operation	Operands	F mode / L mode / @PROC
@SET	$\left. \begin{array}{l} = \text{string} \\ \\ = \text{INTERNAL} \left\{ \begin{array}{l} \text{int-var} \\ \text{ln} \\ \text{str-var} \end{array} \right\} \\ \\ [,cl] = \text{CHAR} \left\{ \begin{array}{l} \text{int-var} \\ \text{ln-var} \\ \text{str-var} \end{array} \right\} \end{array} \right\}$	

str-ln Specifies one of the 21 string variables #S0, #S1,..., #S20. The specification may be direct, i.e. the name of the string variable, or indirect, i.e. a string variable name plus or minus an offset. The entry #S0 + 3L, for example, is equivalent to #S3, and #S16-2L is equivalent to #S14. The offset may also be specified as an integer variable. If, for example, #I12 contains 10, then #S3 + #I12 is equivalent to #S3 + 10L, or #S13.

cl Specifies the column in the string variable starting at which the value is to be written. The default value for cl is 1.
 If a string variable is assigned the contents of an integer variable, the value for cl must not exceed 246.
 If a string variable is assigned the contents of a line number variable, the value for cl must not exceed 248.
 If a string variable is assigned the contents of a string variable, the value for cl must not exceed 253.

string The string to be assigned to the string variable.
 The string may be specified:

- explicitly, enclosed in single quotes, or
- implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

int-var

The integer variable (#I0 to #I20) whose contents are to be written into str-ln.

ln A line number (e.g. 5).

The minimum value is 0.0001, the maximum 9999.9999.

ln may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %,\$).

ln-var

The line number variable (#L0 to #L20) whose value is to be placed in str-ln. The maximum possible value of ln-var is 9999.9999, the minimum 0.0001.

str-var

The string variable (#S0 to #S20) whose name is to be placed in str-ln.

INTERNAL

- The value of an integer variable is stored internally as a 4-byte binary number and this binary number is placed unchanged in the first four bytes of the string variable. If, for example, an integer variable has the value 359, it actually contains the corresponding binary number B'0000 0000 0000 0000 0001 0110 0111', i.e. X'00 00 01 67'. This is what is then placed in the first four bytes of the string variable.
- A line number is always kept in a 4-byte field (8 half-bytes). Each half-byte contains one of the eight digits which form the line number. The contents of this 4-byte field are placed unchanged in the first four bytes of the string variable. The line number 47.11, for example, is stored internally as X'00471100' and this is what is actually placed in the first four bytes of the string variable.
- The name of a string variable is stored internally as four printable characters and EDT places these characters unchanged in the first four bytes of str-ln. If, for example, #S1 is specified for the name of the string variable, EDT writes #S01, or X'7BE2F0F1', into the first four bytes of str-ln.

CHAR

Specifies that EDT is to

- convert the contents of an integer variable into a printable number and place this in a string variable. Conversion results in an 11-character printable number, where the first character is a blank (for a positive number) or a minus sign (for a negative number).
- convert the line number stored in a line number variable into printable form and to place it in the form llll.llll in string variable str-ln, starting at column cl. In order to obtain 9 printable characters in all cases, any necessary blanks (nonprintable zeros) are inserted on the left.

- place the name of string variable *str-var* in string variable *str-ln*, starting at column *cl*. The name of *str-var* is converted to the form #*Sdd*, where *dd* lies in the range 00, 01, ..., 20.

Example 1

Assign a string to a string variable.

```

1.      @CREATE 1: '-'*40 ----- (01)
1.      @SET #S1 = 1
1.      @SET #S2 = '2'
1.      @SET #S3 = '3'*15
1.      @SET #S4 = B'11110000'*20 ----- (02)
1.      @SET #S5 = X'F4'*16
1.      @SET #S6 = 'HA'*7
1.      @SET #S7 = #S6:2
1.      @SET #S9-1L = #S1+4L
1.      @SET #I7 = 10 ----- (03)
1.      @SET #S19-#I7 = #S18-#I7 ----- (04)
1.      @PRINT #S1.-#S9 ----- (05)
#S01 -----
#S02 2
#S03 3333333333333333
#S04 00000000000000000000
#S05 44444444444444444444
#S06 HAHAAAAAAAAHAHA
#S07 A
#S08 44444444444444444444
#S09 44444444444444444444

```

- (01) Line 1 is created with 40 '-' characters in it.
- (02) Values are assigned to string variables #S1 through #S7:
 #S1 is set to the contents of line 1.
 #S2 is set to '2'.
 #S3 is set to 15 '3' characters.
 #S4 is set to 20 times B'11110000', i.e. X'F0' or character '0'.
 #S5 is set to 16 times the character X'F4' or character '4'.
 #S6 is set to 7 repetitions of the string 'HA'.
 #S7 is set to the contents of column 2 in #S6.
 #S9 - 1L, i.e. #S8, is set to the contents of #S1 + 4L, i.e. #S5.
- (03) The integer variable #I7 is set to 10.
- (04) #S19 - - #I7, i.e. #S19 - 10L (or #S9) is set to the contents of #S18 - #I7, i.e. #S8.
- (05) The contents of string variables #S1 through #S9 are displayed.

Example 2

Assign the contents of an integer variable, a line number or the name of a string variable (internal format) to a string variable.

```

1.      @SET #I0 = 19 ----- (01)
1.      @SET #S0 = INTERNAL #I0 ----- (02)
1.      @SET #I1 = -1 ----- (03)
1.      @SET #S1 = INTERNAL #I1 ----- (04)
1.      @SET #S3 + #I1 = INTERNAL 25.4356 ----- (05)
1.      @SET #L3 = 2222.2222 ----- (06)
1.      @SET #S10 - 7L = INTERNAL #L3 + 11.11 ----- (07)
1.      @78.7878 ----- (08)
78.7878 @SET #S4 = INTERNAL * ----- (09)
78.7878 @SET #S5 = INTERNAL #S0 ----- (10)
78.7878 @SET #S6 = INTERNAL #S19 ----- (11)
78.7878 @PRINT #S0.-#S6 X,#S5,#S6 ----- (12)
#S00 00000013
#S01 FFFFFFFF
#S02 00254356
#S03 22333322
#S04 00787878
#S05 7BE2F0F0
#S06 7BE2F1F9
#S05 #S00
#S06 #S19
78.7878
    
```

- (01) Integer variable #I0 is set to 19, i.e. X'13'.
- (02) #S0 is set to the internal representation of the contents of #I0, i.e. X'00000013'.
- (03) #I1 is set to - 1. In hexadecimal form, this is expressed as X'FFFFFFF'.
- (04) #S1 is set to the internal representation of the contents of #I1.
- (05) #S3 + #I1 is equivalent to #S3 - 1L, i.e. #S2. #S2 is thus set to the internal representation of the contents of line 25.4356.
- (06) Line number variable #L3 is set to 2222.2222.
- (07) #S10 - 7L is equivalent to #S3, and this string variable is set to the internal representation of the contents of line #L3 + 11.11, i.e. line 2222.2222 + 11.11 = 2233.3322.
- (08) 78.7878 is made the current line number.
- (09) #S4 is set to the internal representation of the contents of the current line.
- (10) #S5 is set to the internal representation of the contents of string variable #S0.

- (11) #S6 is set to the internal representation of the contents of string variable #S19.
- (12) The contents of string variables #S0, ..., #S6 are displayed in hexadecimal form and the contents of #S5 and #S6 are also displayed in normal text mode.

Example 3

Assign the contents of an integer variable (printable number) to a string variable.

```

1.      @SET #I0 = 11223344
1.      @SET #I1 = 55667788 ----- (01)
1.      @SET #I2 = -99999999
1.      @SET #I3 = 5
1.      @CREATE #S16 : '@'*40 ----- (02)
1.      @SET #S16 = CHAR #I0 ----- (03)
1.      @SET #S17-1L,14 = CHAR #I1 ----- (04)
1.      @SET #S11 + #I3,27 = CHAR #I2 ----- (05)
1.      @PRINT #S16
#S16  0011223344@@ 0055667788@@-0099999999@@ ----- (06)

```

- (01) Values are assigned to the integer variables #I0, #I1, #I2 and #I3.
- (02) String variable #S16 is set to 40 times the character '@'.
- (03) The integer in #I0 is converted to a printable number and placed in #S16, starting at column 1.
- (04) The integer in #I1 is converted to a printable number and placed in #S17 - 1L, i.e. in #S16.
- (05) Since #I3 = 5, #S11 + #I3 is the same as #S11 + 5L, i.e. #S16. The integer in #I2 is thus to be converted and placed in #S16, starting at column 27.
- (06) The contents of string variable #S16 are displayed.

Example 4

Assign the contents of a integer variable (printable number) to a string variable by means of a positional operand.

```

1.      @SET #I5 = 18
1.      @SET #I6 = -27 ----- (01)
1.      @SET #I7 = 333333
1.      @PROC 7 ----- (02)
1.      @ @PARAMS &INTVAR ----- (03)
2.      @ @SET #S0 = CHAR &INTVAR ----- (04)
3.      @ @ON #S0:2-2: DELETE '0' ----- (05)
4.      @ @IF .TRUE. GOTO 3 ----- (06)
5.      @ @CREATE #S1: 'THE CONTENTS OF &INTVAR ARE ',#S0 ----- (07)
6.      @ @PRINT #S1 N ----- (08)

```

```

7.      @END ----- (09)
1.      @DO 7(#I5) ----- (10)
THE CONTENTS OF #I5 ARE  18
1.      @DO 7(#I6) ----- (11)
THE CONTENTS OF #I6 ARE -27
1.      @DO 7(#I7) ----- (12)
THE CONTENTS OF #I7 ARE  333333

```

- (01) Integer variables #I5, #I6 and #I7 are set to various values.
- (02) EDT switches to work file 7.
- (03) Positional parameter &INTVAR is defined.
- (04) #S0 is set to the printable value of the integer variable specified in the @DO statement.
- (05) Since column 1 of #S0 contains the sign, the leading (printable) zeros start in column 2. The first of these is deleted.
- (06) This loop deletes all leading zeros, one at a time.
- (07) If there are no further leading zeros, string variable #S1 is created.
- (08) #S1 is displayed without a line number.
- (09) EDT returns to work file 0.
- (10) The procedure in work file 7 is executed, with integer variable #I5 specified as the parameter.
- (11) This time, #I6 is passed as the parameter.
- (12) Finally, #I7 is passed as the parameter.

Example 5

Assign the line number (printable number) stored in a line number variable to a string variable.

```

1.      @SET #I0 = 6
1.      @SET #I1 = 28
1.      @SET #L0 = 0.0045 ----- (01)
1.      @SET #L1 = 9999.9999
1.      @SET #L2 = 55.55
1.      @CREATE #S5 : '*'*40 ----- (02)
1.      @SET #S5,3 = CHAR #L0 ----- (03)
1.      @SET #S3+2L,15 = CHAR #L1 ----- (04)
1.      @SET #S11-#I0,#I1 = CHAR #L2 ----- (05)
1.      @PRINT #S5 ----- (06)
#S05 ** 0.0045***9999.9999**** 55.5500****

```

- (01) Integer variables #I0 and #I1 and the line number variables #L0, #L1 and #L2 are set to various values.
- (02) #S5 is created and consists of 40 '*' characters.
- (03) The printable form of the line number stored in #L0 is placed in #S5, starting at column 3.
- (04) #S3 + 2L again addresses #S5 and the printable form of the line number in #L1 is thus placed in #S5, starting at column 15.
- (05) #S11 - #I0 is equivalent to #S11 - 6L, which in turn is the same as #S5. The printable form of the line number in #L2 is thus placed in #S5, starting at column 28.
- (06) The contents of string variable #S5 are displayed.

Example 6

Assign the name of a string variable to a string variable.

```

1.      @SET #I0 = 19
1.      @SET #S0 = INTERNAL #I0
1.      @PRINT #S0 X
#S00 00000013 ----- (01)
1.      @SET #S3 = '*'*40
1.      @PRINT #S3
#S03 *****
1.      @SET #S3 , 10 = CHAR #S0
1.      @PRINT #S3 ----- (02)
#S03 *****#S00*****

```

- (01) Integer variable #I0 and string variables #S0 and #S3 are set to various values and displayed.
- (02) @SET, format 2, converts the name of #S0 to #S00 and places this new name in #S3, starting at column 10. #S3 is then displayed.

Example 7

Assign the name of a string variable to a string variable by means of positional and keyword parameters.

```

1.      @CREATE #S18 : '*'*40 ----- (01)
1.      @PRINT #S18
#S18 ----- (02)
1.      @PROC 7 ----- (03)
1.      @ @PARAMS &STRVAR1,&STRVAR2,&COLUMN=1 ----- (04)
2.      @ @SET &STRVAR1,&COLUMN = CHAR &STRVAR2 ----- (05)
3.      @ @PRINT &STRVAR1 ----- (06)
4.      @END ----- (07)
1.      @DO 7(#S18,#S16) ----- (08)

```

```

#S18 #S16-----
1.   @D0 7(#S17+1L,#S20,COLUMN=20) ----- (09)
#S18 #S16-----#S20-----
1.   @D0 7(#S18,#S18,COLUMN=14) ----- (10)
#S18 #S16-----#S18--#S20-----
1.   @SET#I12 = 26 ----- (11)
1.   @SET#I13 = 12
1.   @D0 7(#S6+#I13,#S0,COLUMN=#I12) ----- (12)
#S18 #S16-----#S18--#S20--#S00-----

```

- (01) String variable #S18 contains 40 '-' characters.
- (02) The contents of #S18 are displayed.
- (03) EDT switches to work file 7.
- (04) Two positional parameters (&STRVAR1 and &STRVAR2) and one keyword parameter (&COLUMN) are defined.
- (05) When the procedure is executed, @SET format 18 is used. The values for the parameters must be specified in the @D0 7 statement.
- (06) This displays the contents of a string variable.
- (07) EDT returns to work file 0.
- (08) The statements of the procedure in work file 7 are executed, with &STRVAR1 being replaced by #S18 and &STRVAR2 by #S16. The default value (preset to 1) is used for &COLUMN.
- (09) #S17 + 1L addresses string variable #S18 and the value 20 is specified for keyword parameter &COLUMN.
- (10) Note, in this case, that the two positional parameters have the same value. This means that the name of the string variable itself is placed in the string variable.
- (11) #I12 and #I13 are set to the specified values.
- (12) #S6 + #I13 is equivalent to #S6 + 12L, i.e. #S18. The integer variable #I12 is used for keyword parameter &COLUMN.

@SET (Format 3) Assigning values to line number variables

This format of @SET is used to

- assign a line number to a line number variable.
- convert the integer value in an integer variable into a line number and assign it to a line number variable.
- assign a printable number as a line number to a line number variable.
- assign the internal representation of a string to a line number variable. This internal representation consists of 8 hexadecimal digits. It is assumed that the decimal point of the line number lies between the 4th and 5th digit.

Operation	Operands	F mode / L mode / @PROC
@SET	$\text{In-var} = \left. \begin{array}{l} \text{In} \\ \text{int-var} \\ \text{SUBSTR string} \\ \text{STRING string} \end{array} \right\}$	

- In-var** The line number variable (#L0 to #L20) to which a value is to be assigned. The maximum possible value of In-var is 9999.9999, the minimum 0.0001.
- In** A line number (e.g. 5). The minimum possible value is 0.0001, the maximum 9999.9999.
In may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %, \$).
- int-var** The integer variable (#I0 to #I20) whose contents are to be converted and assigned to In-var. The permissible value range for int-var is 1 to 99999999, resulting in the line numbers 0.0001 to 9999.9999. For conversion, the integer is divided by 10000 and the result is assigned to the line number variable. This is necessary in order to permit any possible line number to be generated.
- string** A character string.
The string may be specified:
- explicitly, enclosed in single quotes, or
 - implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).
- SUBSTR string**
Specifies that EDT is to assign a printable number to the specified line number variable. "string" must be a printable number greater than '0'.

STRING string

Specifies that EDT is to interpret the internal representation of a string as a line number. The EBCDIC code of "string" is assigned to the line number variable. "string" may contain only characters whose EBCDIC codes consist exclusively of digits. If "string" has less than four characters, leading zeros are inserted. If "string" has more than four characters, only the first four characters are used.



If integer variables are used within operand Ln, it should be noted that, for example, the expression #L5 = #L6 + #I7 does not form the sum of the values of #L6 and #I7. Instead, the result in #L5 will point to the line which is #I7 lines after the line specified in #L6. If, for example, #I7 contains the value 7, #L5 = #L6 + #I7 is equivalent to #L5 = #L6 + 7L.

Example 1

Assign a line number to a line number variable.

```

1.      @DELETE
1.      @1:AAAAA
2.      @C1-10 TO 2.5 (0.5)
11.5   @PRINT
1.0000 AAAAA
2.5000 AAAAA
3.0000 AAAAA
3.5000 AAAAA
4.0000 AAAAA
4.5000 AAAAA
5.0000 AAAAA
5.5000 AAAAA
6.0000 AAAAA ----- (01)
6.5000 AAAAA
7.0000 AAAAA
7.5000 AAAAA
8.0000 AAAAA
8.5000 AAAAA
9.0000 AAAAA
9.5000 AAAAA
10.0000 AAAAA
10.5000 AAAAA
11.5   @SET #I10 = 4
11.5   @SET #I11 = 7
11.5   @SET #L0 = 1 ----- (02)
11.5   @SET #L1 = #L0 + #L0 ----- (03)
11.5   @SET #L2 = % ----- (04)
11.5   @SET #L3 = #L2 + 1 ----- (05)

```

```

11.5   @SET #L4 = #L2 + 1L ----- (06)
11.5   @SET #L5 = #L3 + #L4 ----- (07)
11.5   @SET #L6 = #L4 + #I11 ----- (08)
11.5   @SET #L7 = #L6 - #I10 ----- (09)
11.5   @SET #L8 = $-2L ----- (10)
11.5   @SET #L9 = $+$ ----- (11)
11.5   @SET #L10 = $-% ----- (12)
11.5   @STATUS = L ----- (13)
#L00=  1.0000   #L01=  2.0000   #L02=  1.0000
#L03=  2.0000   #L04=  2.5000   #L05=  4.5000
#L06=  6.0000   #L07=  4.0000   #L08=  9.5000
#L09= 21.0000   #L10=  9.5000   #L11=  0.0000
#L12=  0.0000   #L13=  0.0000   #L14=  0.0000
#L15=  0.0000   #L16=  0.0000   #L17=  0.0000
#L18=  0.0000   #L19=  0.0000   #L20=  0.0000

```

- (01) Several lines are created in the virtual file and values are assigned to integer variables #I10 and #I11.
- (02) #L0 is set to 1.
- (03) #L1 is set to #L0 + #L0, i.e. $1 + 1 = 2$.
- (04) #L2 is set equal to the first existing line number, i.e. 1.
- (05) #L3 is set to #L2 + 1, i.e. $1 + 1 = 2$.
- (06) #L4 is set to #L2 + 1L, thus addressing the line number after the one specified in #L2. Since #L2 has the value 1, and the line following line 1 has the line number 2.5, #L4 is set to 2.5.
- (07) #L5 is set to the sum of #L3 and #L4 and is thus $2 + 2.5 = 4.5$.
- (08) #L6 is set to #L4 + #I11.
This is not the sum of 2.5 and 7, but is equivalent to the expression #L4 + 7L, or the 7th line number after line #L4. Since #L4 = 2.5, #L6 is set to 6.
- (09) As in (08), #L7 is set to #L6 - #I10, which is equivalent to #L6 - 4L, i.e. 4.
- (10) #L8 is set to the number of the third line from the end of the file, since \$ refers to the last line, \$-1L the one before the last and \$-2L the third line from the end. #L8 is thus set to 9.5.
- (11) #L9 is set to \$ + \$ - i.e. $10.5 + 10.5 = 21$.
- (12) #L10 is set to \$ - % = $10.5 - 1 = 9.5$.
- (13) The values of the line number variables are displayed.

Example 2

Assign the contents of an integer value as a line number to a line number variable.

```
1.    @SET #I0 = 1
1.    @SET #L0 = #I0 ----- (01)
1.    @STATUS = #L0
#L0=  0.0001
1.    @SET #I1 = 20000
1.    @SET #L1 = #I1 ----- (02)
1.    @STATUS = #L1
#L01=  2.0000
1.    @SET #I2 = 12345678
1.    @SET #L2 = #I2 ----- (03)
1.    @STATUS = #L2
#L02=1234.5678
1.
```

- (01) #I0 is set to 1.
#L0 is set to #I0/10000, i.e. 0.0001 and its contents are displayed.
- (02) #I1 is set to 20000.
#L1 is set to #I1/10000, i.e. 2.0000 and its contents are displayed.
- (03) #I2 is set to 12345678.
#L2 is set to #I2/10000, i.e. 1234.5678 and its contents are displayed.

Example 3

Assign a printable number as a line number to a line number variable.

```

1.      @CREATE 1: 'ABC1.23.4.5.67' ----- (01)
1.      @CREATE #S0: '      .5'
1.      @SET #L0 = SUBSTR X'F1F2F3' ----- (02)
1.      @SET #L1 = SUBSTR '123.456' ----- (03)
1.      @SET #L2 = SUBSTR B'11110111'*4 ----- (04)
1.      @SET #L3 = SUBSTR 1:11-13,6-7: ----- (05)
1.      @SET #L4 = SUBSTR #S0 ----- (06)
1.      @SET #L5 = SUBSTR '9'*3 ----- (07)
1.      @STATUS = L ----- (08)
#L00= 123.0000      #L01= 123.4560      #L02=7777.0000
#L03=  5.6230      #L04=  0.5000      #L05= 999.0000
#L06=  0.0000      #L07=  0.0000      #L08=  0.0000
#L09=  0.0000      #L10=  0.0000      #L11=  0.0000
#L12=  0.0000      #L13=  0.0000      #L14=  0.0000
#L15=  0.0000      #L16=  0.0000      #L17=  0.0000
#L18=  0.0000      #L19=  0.0000      #L20=  0.0000
1.

```

- (01) Line 1 and string variable #S0 are created.
- (02) X'F1F2F3' is identical to the printable string '123'. This number is assigned to #L0.
- (03) #L1 is set to 123.4560.
- (04) B'11110111'*4 is equivalent to X'F7'*4, i.e. '7777'; #L2 is thus set to 7777.0000.
- (05) Columns 11, 12, 13, 6, 7 (in this order) of line 1 result in the value '5.623'; #L3 is thus set to 5.6230.
- (06) #S0 contains '.5'. Since blanks are ignored when converting, #L4 is set to 0.5000.
- (07) '9'*3 is equivalent to '999' and #L5 is set to 999.0000.
- (08) The values of all line number variables are displayed.

Example 4

Assign the internal representation of a character string as a line number to a line number variable.

```

1.      @CREATE 1: X'01020304050607' ----- (01)
1.      @CREATE #S0: B'01111001'*20 ----- (02)
1.      @SET #L0 = STRING X'34' ----- (03)
1.      @SET #L1 = STRING B'00000110' ----- (04)
1.      @SET #L2 = STRING ' '*2 ----- (05)
1.      @SET #L3 = STRING 1:5,4: ----- (06)
1.      @SET #L4 = STRING 1:1-2,2-4: ----- (07)
1.      @SET #L5 = STRING #S0:6: ----- (08)
1.      @STATUS = L ----- (09)
#L00=   0.0034      #L01=   0.0006      #L02=   0.4040
#L03=   0.0504      #L04=  102.0203      #L05=   0.0079
#L06=   0.0000      #L07=   0.0000      #L08=   0.0000
#L09=   0.0000      #L10=   0.0000      #L11=   0.0000
#L12=   0.0000      #L13=   0.0000      #L14=   0.0000
#L15=   0.0000      #L16=   0.0000      #L17=   0.0000
#L18=   0.0000      #L19=   0.0000      #L20=   0.0000
1.

```

- (01) Line 1 is created with a 7-character (nonprintable) string.
- (02) String variable #S0 consists of 20 repetitions of the nonprintable character X'79'.
- (03) #L0 is set to line number 0.0034.
- (04) #L1 is set to line number 0.0006 since B'00000110' = X'06'.
- (05) #L2 is set to line number 0.4040 since ' '*2 = X'40'*2 = X'4040'.
- (06) Since columns 5 and 4 of line 1 contain X'05' and X'04', #L3 is set to 0.0504.
- (07) Columns 1, 2, 2, 3, 4 (in this order) of line 1 form the value X'0102020304'. Since not more than four columns are used to form the contents of #L4, the last column (4) is ignored and #L4 is set to 102.0203.
- (08) Column 6 of #S0 contains B'01111001', i.e. X'79' and #L5 is thus set to 0.0079.
- (09) The values of all line number variables are displayed.

@SET (Format 4) Placing values in lines

This format of @SET is used to

- convert the contents of an integer variable into printable form and place the result in the line specified by the line number variable.
- write the name of a string variable into a line, starting at a specified column. The number is specified by means of a line number variable.
- convert the contents of a line number variable into printable form and place the result in a line addressed by a second line number variable.

Operation	Operands	F mode / L mode / @PROC
@SET	In-var [,cl] = CHAR $\left\{ \begin{array}{l} \text{int-var} \\ \text{str-var} \\ \text{In-var1} \end{array} \right\}$	

- In-var** A line number variable (#L0 to #L20) which specifies the line where EDT is to write the contents of int-var, str-var or In-var1 in printable form. The maximum possible value of In-var is 9999.9999, the minimum 0.0001.
- cl** Specifies the column in the line starting at which the value is to be written. The default value for cl is 1.
- If** cl > 1 is specified, the columns from the beginning of the line up to cl-1 are filled with blanks.
- If an integer is written to a line, the value for cl must not exceed 246.
 - If the name of a string variable is written to a line, the value for cl must not exceed 253.
 - If the value of a line number variable is written to a line, the value for cl must not exceed 248.
- CHAR int-var** Specifies that EDT is to convert the value in int-var into printable form and place it, starting at column cl, in the line specified by In-var. int-var is an integer variable (#I0 to #I20). Conversion of the integer variable results in a printable number with 11 characters: the first character is the sign of the number and the remaining characters are the digits of the number. The sign for a non-negative number is a blank, that for a negative number a minus sign (-). Since 11 characters are generated in all cases, the positions after the sign are filled, if necessary, with printable leading zeros.

CHAR str-var Specifies that EDT is to write the name of a string variable into a specified line.

str-var is one of the 21 string variables (#S0 to #S20). EDT writes this name into the line specified by ln-var. The name of the string variable is converted to the form #Sdd, where dd lies in the range 00, 01, ..., 20.

CHAR ln-var1 Specifies that EDT is to convert the contents of the line number variable ln-var1 into printable form and to write the result into a specified line.

ln-var1 is a line number variable (#L0 to #L20) whose contents are to be placed in the specified line. Conversion of the value of a line number variable always results in 9 printable characters in the form IIII.IIII where each I represents a printable digit. Leading zeros are replaced by blanks.

Example 1

Place the contents of an integer variable in the line specified by means of a line number variable.

```

1.      @SET #I18 = 1234 ----- (01)
1.      @SET #I19 = 5678
1.      @SET #I20 = #I18 + #I19
1.      @CREATE 5: 'XXXXXXXXXX PLUS YYYYYYYYYY IS ZZZZZZZZZZ'
1.      @SET #L18 = 5
1.      @SET #L18 = CHAR #I18 ----- (02)
1.      @PRINT 5
5.0000 0000001234 PLUS YYYYYYYYYY IS ZZZZZZZZZZ
1.      @SET #L18,17 = CHAR #I18 ----- (03)
1.      @PRINT 5
5.0000 0000001234 PLUS 0000001234Y IS ZZZZZZZZZZ
1.      @SET #L18,18 = CHAR #I19
1.      @PRINT 5
5.0000 0000001234 PLUS 0000005678 IS ZZZZZZZZZZ
1.      @SET #L18,33 = CHAR #I20 ----- (04)
1.      @PRINT 5
5.0000 0000001234 PLUS 0000005678 IS 0000006912
1.

```

- (01) Integer variables #I18, #I19 and #I20 and line number variable #L18 are provided with values and line 5 is created.
- (02) The number in #I18 is converted to printable form and placed in the line addressed by #L18, starting at column 1.
- (03) The number in #I18 is converted to printable form and placed in the line addressed by #L18, starting at column 17.
- (04) Finally, the number in #I20 is converted to printable form and placed in the line addressed by #L18, starting at column 33.

Example 2

Place the name of a string variable in the line specified by means of a line number variable.

```

1.      @SET #L1 = 666.66 ----- (01)
1.      @SET #L1 = CHAR #S20 ----- (02)
1.      @PRINT #L1
666.6600 #S20
1.      @SET #L1,30 = CHAR #S13 ----- (03)
1.      @SET #L1,15 = CHAR #S7 ----- (04)
1.      @PRINT #L1
666.6600 #S20      #S07      #S13
1.

```

- (01) #L1 is set to 666.6600.
- (02) The new line 666.6600 is created and the text '#S20' is written into it, starting at column 1.
- (03) The characters '#S13' are to be stored in the line addressed by #L1, starting at column 30.
- (04) The internal representation of the line number of #S7 is '#S07', and this string is placed in line 666.66, starting at column 15.

Example 3

Place the contents of a line number variable (In-var2) in the line specified by means of a line number variable (In-var1).

```

1.      @DELETE
1.      @SET #L3 = 57.45 , ----- (01)
1.      @SET #L4 =99.99 0
1.      @SET #L3 = CHAR #L4 ----- (02)
1.      @PRINT
57.4500  99.9900
1.      @SET #L3,20 = CHAR #L3 ----- (03)
1.      @PRINT
57.4500  99.9900      57.4500
1.

```

- (01) Values are assigned to the line number variables #L3 and #L4.
- (02) EDT writes the contents of #L4 in printable form in a line which did not previously exist, starting at column 1.
- (03) EDT writes the contents of #L3 in printable form in an existing line, starting at column 20. Note that, as in this case, In-var1 and In-var2 may be the same.

@SET (Format 5) Specifying date and time

This format of @SET is used to

- place either the date or the time of day in a string variable, starting at a specified column.
- place either the date or the time of day in the line whose number is specified in the line number variable.

Operation	Operands	F mode / L mode / @PROC
@SET	$\left. \begin{matrix} \text{str-ln} \\ \text{ln-var} \end{matrix} \right\} [,c] = \left. \begin{matrix} \text{DATE [ISO[4]]} \\ \text{TIME} \end{matrix} \right\}$	

- str-ln** Specifies one of the 21 string variables #S0, #S1, ..., #S20. The specification may be direct, i.e. the name of the string variable, or indirect, i.e. a string variable name plus or minus an offset. The entry #S0 + 3L, for example, is equivalent to #S3, and #S16-2L is equivalent to #S14. The offset may also be specified as an integer variable. If, for example, #I12 contains 10, then #S3 + #I12 is equivalent to #S3 + 10L, or #S13.
- ln-var** A line number variable (#L0 to #L20) which specifies the line into which EDT is to write the date or time. The maximum possible value for ln-var is 9999.9999, the minimum 0.0001.
- cl** The column starting at which the date or time is to be placed. The default column is 1. The value for cl must not exceed 246 or 251, respectively.
- DATE** Specifies that EDT is to write the date in the specified string variable or line, as appropriate. If ISO is not specified, EDT writes the date in the format mm/dd/yyjjj, where mm is the month, dd is the day of the month, yy is the year and jjj is the day of the year.
- ISO** Specifies that EDT is to write the date in the format yy-mm-ddjjj.
- ISO4** Specifies that EDT is to write the date in the format yyyy-mm-ddjjj.
- TIME** Specifies that EDT is to write the time in the specified string variable or line, as appropriate. EDT writes the time in the format hhmmss, where hh is the hours, mm is the minutes, and ss is the seconds.

Example 1

Place the date in the form mm/dd/yyjjj and the time in the form hhmmss in a string variable.

```

1.      @PROC 3 ----- (01)
1.      @ @DELETE #S0.-.#S4 ----- (02)
2.      @ @SET #S0 = DATE ----- (03)
3.      @ @SET #S1 = TIME ----- (04)
4.      @ @CREATE #S2: '*'*10, ' THE DATE IS ',#S0:4-5: ----- (05)
5.      @ @CREATE #S2: #S2, '.',#S0:1-2:,'.',#S0:7-8:,' ', '*'*11
6.      @ @CREATE #S3: '*'*15, ' THE TIME IS ', '*'*16
7.      @ @CREATE #S4: '***** ',#S1:1-2:,' HOURS ',#S1:3-4:,' MINUTES AND '
8.      @ @CREATE #S4: #S4,#S1:5-6:,' SECONDS *****'
9.      @ @PRINT #S2.-.#S4N ----- (06)
10.     @END ----- (07)
1.      @DO 3
***** THE DATE IS 29.09.90 ***** ----- (08)
***** THE TIME IS *****
***** 14 HOURS 25 MINUTES AND 14 SECONDS *****

```

- (01) EDT switches to work file 3.
- (02) The contents of string variables #S0, #S1, #S2, #S3 and #S4 will be deleted at this point in the procedure, i.e. each will contain precisely one blank.
- (03) The date is placed in #S0, starting at column 1.
- (04) The time is placed in #S1, starting at column 1.
- (05) String variables #S2, #S3 and #S4 are used to set up the date and the time with the appropriate texts.
- (06) When procedure file 3 is executed, the contents of string variables #S2, #S3 and #S4 will be displayed at this point.
- (07) EDT returns to work file 0.
- (08) When the procedure in work file 3 is executed, the date and time will be displayed, accurate to the nearest second.

Example 2

Place the date in the form yyyy-mm-ddjjj in a line specified by means of a line number variable.

```
1.      @CREATE 2.5: '-'*40 ----- (01)
1.      @SET #L13 = 2.5 ----- (02)
1.      @SET #L13,15 = DATE IS04 ----- (03)
1.      @PRINT 2.5
2.5000 -----1992-05-05126----- (04)
1.
```

- (01) Line 2.5 is created.
- (02) Line number variable #L13 is set to 2.5.
- (03) The date is to be placed in the line addressed by #L13, starting at column 15.
- (04) Line 2.5 is displayed.

@SET Specify new current line number and increment

@SET (Format 6)

This format of @SET is used to specify a new current line number and a new current increment. It is also possible to specify an input text for EDT.

Operation	Operands	F mode / L mode
@SET	In [(inc)] [:text]	

The keyword "S[ET]" must always be specified in F mode. @In entered without "SET" will be rejected with an error message.

- | | |
|------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| In | <p>The new current line number (e.g. 5).
The minimum value is 0.0001, the maximum 9999.9999. If no value is specified for inc, In also implicitly defines the new current increment, e.g. 5 defines an increment of 1 and 5.0 defines an increment of 0.1.
In may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %, \$).</p> |
| inc | <p>The new current increment value.
The minimum value is 0.0001, the maximum 9999.9999. If In and inc are omitted, the increment is 1.</p> |
| text | <p>Any character string.
If the first non-blank character in this string is</p> <ol style="list-style-type: none"> not the EDT statement symbol or user escape symbol, then any blanks following the : are regarded as part of the text. The following processing guidelines apply: <ul style="list-style-type: none"> "text" is placed at the beginning of line In any tab characters are interpreted the current line number is incremented by the current increment value. the EDT statement symbol, then any blanks following the : are ignored. If the next character is <ul style="list-style-type: none"> not the EDT statement symbol, then "text" is interpreted as an EDT statement and executed immediately the EDT statement symbol, then "text" is treated as a text line as described in 1), above. the user escape symbol, then the external statement routine is executed (see @USE). |

Unlike @ (q.v.), @SET does not create a new stack entry.

Example

```

23.00 .....
set 105(0.3):read 'xmpl.text' .....0000.00:001(1)

```

Line 105 is to be created. Since the text after the colon does not begin with @, it is stored as the contents of line 105.

```

105.00 READ 'XMPL.TEXT' .....
106.00 .....

delete ; lower on ; set 105(0.3):@read 'xmpl.text' .....0105.00:001(1)

```

The work file is to be deleted and the file XMPL.TXT is to be read in, starting at line 105 and with an increment of 0.3. @LOWER ON causes the lowercase letters in the file to be displayed correctly.

```

105.00 @SET defines a new current line number and a new current increment.....
105.30 A text input for EDT may also be specified. This input may be an.....
105.60 EDT statement, which is then executed immediately. The input may.....
105.90 also be any other string, which then becomes the contents of the.....
106.20 specified new line. After this, the current line number is.....
106.50 incremented by the current increment value.....
107.80 .....

```

@SETF Position window

@SETF is used

- to switch to another work file or
- to position the window vertically or horizontally:
 - in the current work file
 - in any work file (0 to 9) without leaving the current work file
 - in any work file and switch to this work file
 - in all work files (0 to 9) at the same time.

Operation	Operands	F mode / L mode
@SETF #	$\left[\begin{array}{c} \text{fwkfv} \\ \mathbf{GLOBAL} \\ \text{(fwkfnr)} \end{array} \right] \left[\begin{array}{c} \text{ln} \\ \text{vpos} \end{array} \right] \left[\begin{array}{c} \text{:cl:} \\ \text{hpos} \end{array} \right]$	

is permissible only in F mode. At least one operand must be specified.

fwkfv	A work file variable (\$0..\$9) which specifies the work file to be positioned. The current work file remains unchanged.
fwkfnr	The work file (0..9) which is to be positioned. Before this is done, EDT switches to this work file. If only (wklno) is specified, EDT simply switches to this work file, which becomes the current work file.
GLOBAL	All 10 work files (0 to 9) are positioned at the same time. In @DO and @INPUT procedures, @SETF GLOBAL is rejected with an error message.
vpos	The new (relative) vertical position. This may be specified as +[n], ++, - -[n], - - or as +([m,..]), ++([m,..]), - -([m,..]) and - -([m,..]). m is one of the 9 possible record marks to which the cursor is to be positioned. More than one record mark can be specified. Marks with special functions (e.g. mark 15 for write protection) are not evaluated here.
cl	The new (absolute) horizontal position, i.e. the number of the column which is to be displayed as the first column of the work file in the window. Values between 1 and 256 are permitted.
hpos	The new (relative) horizontal position. This may be specified as >[n], <[n] or <<.

If only the first operand is specified, then

- @SETF (fwkfnr) simply switches to the new work file; previous work files are first terminated;
- @SETF fwkfv positions the specified work file to line 1, column 1;
- @SETF positions the current work file to line 1, column 1;
- @SETF GLOBAL positions all work files to line 1, column 1.

In procedures (@DO and @INPUT procedures), the window can be positioned with @SETF ? after an @ON statement.

The previous operands FIRST and LAST must be replaced by the symbolic line numbers % and \$, respectively.

Example

```

23.00 .....
setf (1) 7 :3: .....0000.00:001(2)

```

EDT is to position the work window to line 7, column 3 in work file 1.

```

7.00 @SETF is used.....
8.00 .....
9.00 1. to switch to another work file or.....
10.00 .....
11.00 2. to position the window vertically or horizontally:.....
12.00 .....
13.00 - in the current work file.....
14.00 - in any work file (0 to 9) without leaving the current work file,..
15.00 - in any work file and switch to this work file.....
16.00 - in all work files (0 to 9) at the same time.....
17.00 .....

```


@SETJV Catalog job variable and assign value

@SETJV can be used to:

- enter a job variable in the catalog
- assign a value to a job variable.

If the subsystem "job variable support" is not installed, this statement is rejected with an error message.

Operation	Operands	F mode / L mode
@SETJV	$\left. \begin{array}{l} \{ [string1] = string2 [,...] \} \\ \{ string1 \} \end{array} \right\}$	

string1 Character string specifying a fully qualified job variable name. If the job variable has not yet been entered in the catalog, it is cataloged via the standard functions of the DCLJV macro.
string1 can be specified:

- explicitly as a character string in single quotes
- implicitly via a line number, a line number variable (#L0-#L20) or a string variable (#S0-#S20), with the appropriate column range in each case.

The job variable is assigned the link name *EDTLINK and can be addressed via this name.
 If **string1** is not specified, the job variable is addressed via the link name *EDTLINK. In this case, **string2** must be specified.

string2[,...] Character string to be assigned as a value to the job variable.
 The length of the job variable value is determined by the length of the edited character string. If the edited string contains more than 256 characters, only the first 256 characters are assigned as the value. EDT then outputs an error message.
 If **string2** is specified more than once, chaining takes place in the specified order. **string2** can be specified:

- explicitly as a string in single quotes
- implicitly via a line number, a line number variable (#L0-#L20) or a string variable (#S0-#S20), with the appropriate column range in each case.

Tab characters are not processed by @SETJV.

Example: see the example for @STAJV

@SETLIST Extend list variable

@SETLIST can be used to

- delete a list (/FREE-VAR)
- add an individual element to a list
- add all the data lines in a line range to a list
- add the marked data lines in a line range to a list.

Operation	Operands	F mode / L mode
@SETLIST	$\text{string} \left\{ \begin{array}{l} [\text{range}^*] [\text{MARK } [m]] \\ \text{str-var} \end{array} \right\} \text{[:col:],[,]}$ <p style="text-align: center;">[MODE=APPEND PREFIX OVERWRITE]</p>	

string A character string specifying the name of the list variable.

string can be entered

- explicitly, i.e. as a character string enclosed in single quotes, or
- implicitly, i.e. by means of a line number, a line-number variable or a string variable (each can be specified with a column range).

range*

A range of lines consisting of:

- one or more line numbers separated in each case by a comma (e.g. 4,6,15)
- one or more line ranges separated in each case by a comma (e.g. 5-10,17-19)
- a combination of individual lines and line ranges (e.g. 4,7-23,8,15-30))

A range of lines can also be specified by means of the current line-range symbol (see @RANGE), symbolic line numbers (e.g. %,\$) or line-number variables.

It is not permissible to specify string variables.

If no line range is specified, the statement acts on all of the lines in the current work file.

MARK

The statement is to act only on the marked lines of the specified line range.

m m = 1,...,9

Number of the record mark which is to be used as a criterion for selecting lines. If m is not specified, the statement acts only on lines marked with 1.

str-var Name of a string variable containing a string which is to be added as an element.

col One or more columns or column ranges

Column and column-range entries may overlap and/or occur multiple times. If no entry is specified for this operand, the statement acts on the entire line.

MODE

Specifies how the list is to be extended.

=PREFIX

Adds the strings to the beginning of the list, i.e. the strings are inserted one after another as elements in front of the first element.

If the file is empty or there are no lines in the specified line range, EDT issues the message % EDT2903 FILE IS EMPTY.

=APPEND

Adds the strings to the end of the list, i.e. the strings are inserted as elements after the last element.

If the file is empty or there are no lines in the specified line range, EDT issues the message % EDT2903 FILE IS EMPTY.

=OVERWRITE

A /FREE-VARIABLE is executed before the write operation is performed. The elements are then appended one after another in the same way as with APPEND.

If the file is empty or there are no lines in the specified line range, only /FREE-VARIABLE is executed, and message % EDT0211 (FREE-VARIABLE COMMAND PROCESSED FOR S-VARIABLE) is issued in F mode.

If the MODE operand is not specified, MODE=APPEND is assumed.

If, apart from a list name or some range specification, no other operands are specified, it is necessary to enter a comma in front of the MODE operand to distinguish it from MARK as in, for example, @SETLIST 'LISTE',M=O.

The list variable must be declared beforehand with /DECLARE-VARIABLE chars, MULTIPLE-ELEMENTS=LIST and TYPE=STRING or TYPE=ANY.

The length of a newly added list element is determined by the number of line or string-variable characters specified in the column range.

If the line or string variable is shorter than the column entry, it is padded with blanks.

If no column range is specified, the element length is determined by the length of the line or the string variable.

Line and column numbers may occur multiple times in range* or col entries, which results in those lines or columns being read in multiple times.

Assignment of line numbers

Line numbers are assigned without regard to the current line number and current increment. In an empty work file, the current line number and current increment are 1 by default. Both values can be changed with @SET In (inc) (see @SET, format 6).

@SETSW Set switches

@SETSW is used to set or reset user and task switches.

Operation	Operands	F mode / L mode
@SETSW	$\left. \begin{array}{l} \{ \text{ON=} \\ \text{OFF=} \} \end{array} \right\} [\text{U}] \text{int1} [-\text{int2}] [, \dots]$	

ON = The specified switches are to be set.

OFF = The specified switches are to be reset.

int1 The switch which is to be set or reset. This must be an integer in the range 0 through 31 or a integer variable (#I0-#I20).

If the parameter U is specified before int1, the entry refers to a user switch of the user's own user ID, otherwise it refers to a task switch.

int2 All switches between int1 and int2 are to be set or reset. int2 must be an integer in the range 0 through 31 or an integer variable (#I0-#I20). The type of switch depends on what is specified for int1.

@SETSW is used mainly in EDT procedures. @IF, format 4, can be used to check whether or not a task switch or a user switch of the user's own ID has been set.

It is possible to set or reset both user switches and task switches within the same @SETSW statement.

Example 1

@SETSW ON = U1-6,12-20,U31 (user switches 1 to 6 and 31 and task switches 12 to 20 are set).

Example 2

```

1.      @SET #S2 = 'SWITCH 15 IS OFF'
1.      @SET #S3 = 'SWITCH 15 IS ON'
1.      @PROC 9
1.      @ @IF ON = 15 GOTO 4 ----- (01)
2.      @ @PRINT #S2 N
3.      @ @RETURN
4.      @ @PRINT #S3 N
5.      @END
1.      @SETSW OFF = 15 ----- (02)
1.      @DO 9 ----- (03)
SWITCH 15 IS OFF
1.      @SETSW ON = 15 ----- (04)
1.      @DO 9 ----- (05)
SWITCH 15 IS ON
1.

```

- (01) A procedure is created in work file 9. If task switch 15 is on, this procedure displays string variable #S3; otherwise, it displays string variable #S2.
- (02) Task switch 15 is reset.
- (03) The procedure in work file 9 is executed.
- (04) Task switch 15 is set.
- (05) The procedure in work file 9 is executed.

@SETVAR Declare S variable and assign value

@SETVAR can be used to:

- declare an S variable (TYPE=ANY)
- assign a value to a declared S variable (TYPE=STRING, TYPE=INTEGER).

@SETVAR is rejected with an error message in systems in which the SDF-P subsystem has not been installed,

Operation	Operands	F mode / L mode
@SETVAR	$\left\{ \begin{array}{l} \text{string [=string1 =int-var]} \\ \text{SYSEDT} \end{array} \right\}$	[,MODE=ANY NEW UPDATE]

string Character string specifying the name of a simple S variable.

string1 Character string which is assigned as a value to the STRING-type S variable specified by "string".
 "string" can be specified:

- explicitly as a character string in single quotes
- implicitly via a line number, a line number variable or a string variable (in each case with a column range, if required).

int-var Integer variable (#I0-#I20) containing a value which is to be assigned to the INTEGER-type S variable specified by "string".

If neither string1 nor int-var is specified, an S variable of the ANY type is declared.

SYSEDT The S variables SYSEDT-S00 to SYSEDT-S20 are assigned the contents of the string variables #S00 to #S20. If SYSEDT is specified, no messages as to whether this assignment was successful or unsuccessful are issued. No error switches are set.

MODE Specifies whether the S variable must have been previously declared.

= ANY A value is assigned to an existing S variable or to a new S variable.

= NEW The S variable must be new, i.e. not yet declared. If int-var is specified, the S variable is defined with TYPE=INTEGER; otherwise, the S variable is defined with TYPE=STRING.

= UPDATE The S variable must already have been declared.

If an S variable with SCOPE=TASK is to be generated, this must be done with the SDF-P command DECLARE-VARIABLE.

@SHOW Display directory

Format 1 of the @SHOW statement is used to display the directory of a program library or user catalog.

Format 2 of @SHOW is used to output a list of the coded character set names (CCSN) available in the system.

@SHOW (Format 1) Displaying a directory

The list of all elements in the library or user catalog (the directory) is either

- placed in work file 9,
- placed in the current work file,
- displayed on the screen in L mode, or
- output to SYSOUT in batch mode.

Operation	Operands	F mode / L mode
@SHOW	$\left\{ \begin{array}{l} [\text{LIBRARY}=\text{path1 } [, [\text{TYPE}=\text{elemtyp}]] \\ \text{TYPE}=\text{elemtyp} \\ \text{FILES}[\text{=ppath}] \end{array} \right\}$ $[[\text{TO}] \text{In } [(inc)]] \left\{ \begin{array}{l} [\text{SHORT}] \\ \text{LONG } [\text{ISO4}] \end{array} \right\}$	

path1 The name of the program library.

path1 may also be specified by means of a string variable.

elemtype

An element type. elemtype may also be specified by means of a string variable.

EDT displays a directory listing the elements of the specified type. If no element type is specified, the entire library directory is displayed.

Permissible type entries are: S, M, P, J, D, X, R, C, H, L, U, F, *STD or user-defined type names with appropriate base type.

Users who specify a user-defined type name are responsible for ensuring that its associated base type corresponds to one of the permissible types S, M, P, J, D, X, R, C, H, L, U or F.

Type	Contents
S	Source programs
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format
R	Object modules
C	Load modules
H	Created by ASSEMBH
L	created by BINDER
U	created by IFG
F	created by IFG

***STD**

Type S is the default value when EDT is started. The element type set by means of @PAR has no effect in this statement.

- in** The number of the line at which the directory is to start in the current work file. In may also be specified as a symbolic line number or as a line number variable.
- EDT calculates the numbers of the following lines in the target range by incrementing this line number with the increment value specified for this range. The minimum value is 0.0001, the maximum 9999.9999.
- If inc is not specified, EDT uses the increment value implied by the number of decimal places in the line number: for example, 5 implies an increment of 1 and 5.0 implies an increment value of 0.1.
- If In is not specified, the directory is
- displayed on the screen in L mode,
 - output to SYSOUT in batch mode,
 - written into work file 9 in F mode; this work file is cleared before the directory is written.
- If the display of an information line is activated (@PAR INFORMATION = ON), the line with the names of the entries is not written to the work file as the first line but as a write-protected header line.
- inc** The increment for calculating the line numbers. If inc is not specified, the increment implied by the line number specification is used.
- ppath** Specifies the names of the files to be output.
ppath may also be specified by means of a string variable.

SHORT Default value.
 Library elements:
 The line length is 72. If type names have more than 4 characters, element names more than 32 characters or version designations more than 12 characters, an entry will consist of two lines.
 Files:
 Corresponds to the output for @FSTAT when the SHORT operand is specified.

LONG Library elements:
 Each entry is a single line. The length of a line is 120 characters. LONG is particularly well-suited for processing EDT procedures.

Column	Header	Meaning
1-8	TYP	Element type
9-72	ELEMENT	Element name
73-96	VERSION	Version number
97-106	VAR	Variant number
107-116	DATE	Date of last update

The name of the library is included in the header.

Files:
 Corresponds to the output for @FSTAT when the LONG operand is specified.

LONG ISO4 Library elements:
 The ISO4 operand is ignored, since output is automatically in the form YYYY-MM-DD, even if the operand is not specified.

Files:
 Corresponds to the output for @FSTAT when the LONG ISO4 operand is specified.

Only the highest variant of each element is shown. The variant has the function of a write-protection counter, i.e. the variant number of a library element is incremented by 1 whenever the element is written to.

If an element has versions with different numbers, all versions are shown.

@SHOW is rejected if

- there are no elements of the specified type
- the specified library does not exist or is not a program library
- there is still a library element which has been opened by means of @OPEN in work file 9.



- The current line number is changed if a line with a number higher than the existing highest line number is created.
- @ is displayed as the version number for the element with the highest version number.

Example

```

1.00 @SHOW DISPLAYS THE DIRECTORY OF A LIBRARY. THE NAME OF THE.....
2.00 LIBRARY MUST BE SPECIFIED IN THE OPERAND OF THE STATEMENT.....
3.00 .....

show library=edt.lib.xmpl 100(10) .....0001.00:001(0)

```

The entire directory of library 'EDT.LIB.XMPL' is to be written into the current work file, starting at column 100 and with an increment of 10.

```

1.00 @SHOW DISPLAYS THE DIRECTORY OF LIBRARY. THE NAME OF THE LIBRARY.....
2.00 MUST BE SPECIFIED IN THE OPERAND OF THE STATEMENT. ....
100.00 TYP  E L E M E N T                VERSION      VAR      DATE
110.00 C    PROG1                        @            0004    1989-01-11
120.00 D    E.TEXT1                      @            0006    1988-12-05
130.00 D    E.TEXT2                      @            0013    1988-12-05
140.00 D    E.TEXT3                      100         0011    1988-12-13
150.00 D    E.TEXT3                      101         0121    1988-12-20
160.00 D    E.TEXT3                      102         0007    1989-01-11
170.00 J    ASSEMB                       @            0099    1989-01-11
180.00 J    FILE-TRANSFER                @            0045    1989-01-11
190.00 J    PROC1                        1            0001    1988-12-05
200.00 D    THIS-IS-AN-ELEMENT-WITH-A-VERY-L @          0000    1994-08-17
210.00      ONG-NAME
220.00 D    THIS-IS-AN-ELEMENT-WITH-A-VERY-L VERY-OLD.VER 0000    1994-08-17
230.00      ONG-VERSION                   SION
240.00 FREE THIS-IS-AN-ELEMENT-WITH-A-FREE-T @          0000    1994-08-17
250.00 TYPX YPE-NAME
251.00 .....
252.00 .....
253.00 .....
254.00 .....
255.00 .....

show library=edt.lib.xmpl,d 1000(10) ; #1000 .....0100.00:001(0)

```

All elements of type D in the directory of library EDT.LIB.XMPL are to be written into the current work file, starting at line 1000 and with an increment of 10.

1000.00	TYP	E L E M E N T	VERSION	VAR	DATE
1010.00	D	E.TEXT1	@	0006	1988-12-05
1020.00	D	E.TEXT2	@	0013	1988-12-05
1030.00	D	E.TEXT3	100	0011	1988-12-13
1040.00	D	E.TEXT3	101	0121	1988-12-20
1050.00	D	E.TEXT3	102	0007	1989-01-11
1060.00	D	THIS-IS-AN-ELEMENT-WITH-A-VERY-L	@	0000	1994-08-17
1070.00		ONG-NAME			
1080.00	D	THIS-IS-AN-ELEMENT-WITH-A-VERY-L	VERY-OLD.VERSION	0000	1994-08-17
1090.00		ONG-VERSION	SION		
1091.00				

SHOW LIBRARY = EDT.LIB, R

This writes the names of all elements of type R in library EDT.LIB into work file 9.

@SHOW (Format 2) Displaying the coded character set names

Format 2 of @SHOW is used to display a list of the coded character set names available in the system. In addition, a partial code is indicated by a 'P', an EBCDI code by an 'E' and an ISO code by an 'I'.

This statement is rejected with an error message in systems in which the XHCS subsystem is not installed.

Operation	Operands	F mode / L mode
@SHOW	CCS [[TO] ln [(inc)]]	

ln The number of the line at which the list is to start in the current work file. In may also be specified as a line number variable (#L0-#L20) or as a symbolic line number (e.g. %,\$).

If ln is not specified, the list is

- displayed on the screen in L mode,
- output to SYSOUT in batch mode,
- written into work file 9 in F mode; this work file is cleared before the list is written.

If ln is not specified and @PAR INFORMATION=ON is set, EDT will display a header line for describing the displayed information in F mode.

inc The increment for calculating the line numbers. If inc is not specified, the increment implied by the line number specification is used.

Column	Meaning
1-8	CCSN
10	P, if partial code
12	E (EBCDIC) or I (ISO)
14	*, if the coded character set can be displayed on the data display terminal.

@SORT Sort lines in line range

@SORT is used to sort contiguous line ranges in the current work file, in ascending or descending order, byte-by-byte.

@SORT uses a combination of "quicksort" and "bubblesort". The data is sorted by changing the concatenation of the records.

Operation	Operands	F mode / L mode
@SORT	$[\text{rng}^*] \left\{ \begin{array}{l} [:\text{domain}] \\ :R (\text{clrng}) \end{array} \right\} \left\{ \begin{array}{l} [A] \\ D \end{array} \right\}$	

- rng*** The line range within which the data is to be sorted. It can comprise the following:
- a single line (e.g. 6)
 - several contiguous lines (e.g. 8-20)
- The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables.
- String variables must not be used.
- If no line range is specified, all the records or the line range specified by @RANGE, as appropriate, are sorted.
- domain** The column range containing the characters according to which sorting is to be performed. domain consists of:
- a single column (e.g. 10-10)
 - a contiguous column range (e.g. 15-25)
- If only one column number is specified, sorting is performed according to the characters between this column and the end of the line. If the first column number is greater than the length of the line, this line is ignored and is regarded as lexicographically "less than" others.
- Sorting is performed byte-by-byte from left to right. If, during the sort operation, the end of a line is reached, this line is regarded as lexicographically "less than" others which are identical but longer.
- The second column number
- must not be less than the first column number,
 - may be greater than the actual line length.
- If no range is specified, sorting is performed according to the whole line.

R	The column specification in clrng is interpreted starting at the end of the record in the direction of the beginning of the record.
clrng	Column range, is counted from end of record. The same conventions as for domain are valid.
A	Default value. Sorting is performed in ascending order.
D	Sorting is performed in descending order.

Example

@SORT &:1-15

sorts all the records in the current work file in ascending order according to the columns 1 to 15.

@SORT %-.%+19L :R(1-8) D

sorts the first 20 records of the current work file in descending order according to the contents of the last eight columns.

@SORT 20-.\$:#I1-#I2

sorts the records from line number 20 to the end of the current work file in ascending order. The column range which determines the sorting is specified by means of the integer variables #I1 and #I2.

In Number of the line starting at which the job variable information is written to the current work file. The minimum value is 0.0001, the maximum value 9999.9999. In can also be specified via line number variables (#L0-#L20) or via symbolic line numbers (e.g. %, \$).
 If In is not specified, the information is

- displayed on the screen in L mode
- output to SYSOUT in batch mode
- written into work file 9 in F mode; this work file is cleared before the JV information is written.

inc Increment separating the line numbers following In.
 If inc is not specified, the implicitly defined increment is used.

SHORT Only the job variable names, including catalog and user IDs, are output (default value).

LONG In addition to the job variable names (including catalog and user IDs), further catalog information is output.
 If In was not specified and @PAR INFORMATION=ON applies, a header line describing the catalog information is output in F mode.

Header	Meaning
SIZE	Length of the current value in characters
M	Monitoring job variable (, **)
JOBVARIABLE NAME	Job variable name
CR-DATE	Creation date (YY-MM-DD)
S	SHARE attribute (Y/N)
A	ACCESS attribute (W/R)
R	READ-PASS attribute (Y/N)
W	WRITE-PASS attribute (Y/N)

When LONG is specified in F mode, the output length for each job variable name is 80 characters.

ISO4 The creation date (CR-DATE) is specified in the form YYYY-MM-DD.



- The current line number is changed if a line is created with a number greater than the previously highest number.
- Any attempt to query the status of system job variables (\$SYSJV.) will be rejected. The following query is, however possible:
 @SYSTEM 'SHOW-JV-ATTR JV-NAME(\$SYSJV.)' TO 1

Example

```

1.00 90-06-27178.....
2.00 GOOD MORING, TODAY IS THE 27.06.1990.....
3.00 .....

setjv 'today'=2.....0001.00:001(0)

```

The job variable TODAY is assigned the character string in line 2.

```

1.00 90-06-27178.....
2.00 GOOD MORING, TODAY IS THE 27.06.1990.....
3.00 .....

par global,information=on,index=off;stajv 'today' long.....0001.00:001(0)

```

The information line is activated and the line number display is switched off. @STAJV is used to write information on the job variable TODAY into work file 9.

```

SIZE      M      JOBVARIABLE NAME      CR-DATE  S  A  R  W
0000038 $EW.TODAY      90-06-27 Y  W  N  N
.....
.....

.....0001.00:001(9)

```

Information on the job variable is displayed.

@STATUS Show current EDT settings and variable contents

@STATUS is used to display the defined EDT modes and the contents of various EDT constants and variables.

Operation	Operands	F mode / L mode
@STATUS	[=ALL] [= TIME BUFFER SIZE SYMBOLS DELIM VDT MODES FILE PAR[(procnr)] LINEV INTV ln-var int-var SDF CCS LOG SEARCH-OPTION] [,...] [TO ln [(inc)]]	

ALL All information on the following parameters is output: TIME, BUFFER, SIZE, SYMBOLS, DELIM, VDT, MODES and FILE. The user ID and the task sequence number (TSN) are also output.

If ALL is specified, any other operands in the statement have no effect.

TIME This displays

- the current time
- the duration of the current EDT session
- the total CPU time used
- the CPU time used since the last @STATUS statement was entered.

BUFFER	<p>This displays</p> <ul style="list-style-type: none">– the current buffer size– the column number at which the check for valid line length is to begin (@CHECK)– on data display terminals, the right-hand margin. <p>In batch mode, only the column number is shown.</p>
SIZE	<p>Displays the total number of virtual pages needed to store the work files at the moment.</p>
SYMBOLS	<p>This displays</p> <ul style="list-style-type: none">– the current statement symbol; this can be changed using @: (q.v.)– the current text delimiter; this can be changed using @QUOTE (q.v.)– the current wildcard symbols; these can be changed by means of @SYMBOLS (q.v.)– the current range symbol and the currently defined range (see @RANGE)– the current filler character in hexadecimal form (see @SYMBOLS)– the current record separator character (see @PAR SEPARATOR).
DELIM	<p>This displays the current set of text delimiter characters (see @DELIMIT).</p>
VDT	<p>This displays the number of lines and columns on the screen in L mode (see @VDT). With the 9763 Data Display Terminal, the current screen format is also output (see @VDT).</p>
MODES	<p>This displays the settings defined using @BLOCK, @CHECK, @LOWER, @INPUT, @TABS FORWARD, @EDIT and @VTCSET.</p> <p>EDT also displays the setting of the syntax check in L mode, the execution mode (see @SYNTAX) and the values that can be set with @AUTOSAVE.</p>
FILE	<p>This displays the global file name defined by the last @FILE statement. If a version number was also specified in @FILE, this is also displayed.</p> <p>if a POSIX file was opened with @XOPEN, the name of the file is displayed.</p> <p>If a local @FILE entry has been implicitly or explicitly defined, or if a library element or file has been opened by means of @OPEN (format 2), the name of the file or the library and element names are output.</p> <p>FILE is ignored if neither a global nor a local @FILE entry has been specified.</p>

PAR(procno)	<p>Displays the following:</p> <ul style="list-style-type: none">– the local @FILE entry, the library element which was opened (with library name and type) or the name of the POSIX file which was opened, analogous to @STATUS = FILE (see @FILE LOCAL)– the name of any default library or type (see @PAR LIBRARY or @PAR ELEMENT-TYPE)– the values set with @PAR LIMIT and INC. <p>For work files 0 to 9, the following are also output:</p> <ul style="list-style-type: none">– the type of representation in the F mode screen: @PAR LOWER, @PAR HEX, @PAR EDIT LONG, @PAR CODE (see @PAR)– the current structure symbol (see @PAR STRUCTURE) and the values set by means of @PAR LIMIT and @PAR INCREMENT– information on screen services (see @PAR SCALE, @PAR INFORMATION, @PAR PROTECTION)– window-specific default values for position and representation (see @PAR INDEX, @PAR EDIT FULL, @SETF). If not defined, the position is 0.
procno	Number of the work file (0-22).
LINEV	Displays the contents of all line number variables #L0 to #L20.
INTV	Displays the contents of all integer variables #I0 to #I20.
In-var	The contents of the named line number variable are output.
int-var	The contents of the named integer variable are output.
In	<p>Number of the line as of which the information is written to the current work file.</p> <p>If a line is created with a number greater than the previous highest line number, the current line number is changed.</p> <p>If In is not specified, the result is</p> <ul style="list-style-type: none">– output to the screen in L mode– output to SYSOUT in batch mode– written to work file 9 in F mode. The contents of work file 9 are cleared beforehand.
inc	Increment between the line numbers following In. If inc is not specified, the implicitly defined increment is used.
SDF	Either the internal or external program name is output with @STATUS=SDF, depending on the setting. In addition, the setting of the current name type is also output.

- CCS The following is output:
- the name of the predefined CCS
 - the name of the currently selected coded character set (CCSN).
- If the XHCS subsystem is not installed in the system, CCS is ignored.
- LOG Displays the values set for logging (see @LOG).
- SEARCH-OPTION Outputs the default values for the search function (@ON) defined by means of @SEARCH-OPTION.

If no operand is specified, the default value is ALL.

If any operand, except ALL, is specified, the equals sign must also be specified. There must be no comma between the equals sign and the first operand.

If @SYNTAX TEST=ON has been issued to activate the test mode for L-mode input and the @STATUS statement is entered in L mode, any TO In(inc) entry is ignored, i.e. output is directed to SYSOUT instead.



It is possible to use In-var and int-var simultaneously or repeatedly in the same @STATUS statement.

@SUFFIX Append string to lines

@SUFFIX appends a string to the end of one or more existing lines (see also the @PREFIX statement, inserting a string as a prefix).

Operation	Operands	F mode / L mode
@SUFFIX	range WITH string	

- range A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %,\$) or via line number variables. String variables (#S0 to #S20) may also be used.
- string The string to be appended.
The string may be specified:
- explicitly, enclosed in single quotes, or
 - implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

Example

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AND.....
5.00 AND.....
6.00 .....

suffix 4-5 with ' ONCE ' .....0001.00:001(0)

```

The string ONCE is to be appended to lines 4 and 5.

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AND ONCE.....
5.00 AND ONCE.....
6.00 .....

```

```

suffix 4-5 with 3 .....0001.00:001(0)

```

The contents of line 3 are to be appended to lines 4 and 5.

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AND ONCE AGAIN.....
5.00 AND ONCE AGAIN.....
6.00 .....

```

```

suffix 4-5 with ' '*5 ; suffix 4-5 with 4 .....0001.00:001(0)

```

First, 5 blanks are appended to lines 4 and 5. The contents of line 4 are then appended to these two lines.

```

1.00 AND.....
2.00 ONCE.....
3.00 AGAIN.....
4.00 AND ONCE AGAIN      AND ONCE AGAIN.....
5.00 AND ONCE AGAIN      AND ONCE AGAIN.....
6.00 .....

```

@SYMBOLS Define symbols

@SYMBOLS can be used to:

- redefine the wildcard symbols asterisk ('*') and slash ('/') for specifying the search string for the @ON statement (e.g. in order to search for the * and / characters)
- redefine the current filler character for an area in the data window between the end of the record and the end of the screen line.

Operation	Operands	F mode / L mode
@SYMBOLS	[,] [ASTERISK [= '*' ='spec1'] [,] [SLASH [= '/' ='spec2'] [,] [FILLER] [=X'00' =X'hex' ='char']	

ASTERISK Defines the wildcard symbol for any length of character string, even an empty one.

= '*' Default value.

= 'spec1' Special character which defines the wildcard symbol for any length of character string, even an empty one.

SLASH Defines the wildcard character.

= '/' Default value.

= 'spec2' Special character which defines the wildcard symbol.

FILLER Defines the filler character to be inserted in F mode between the end of the record and the end of the screen line.

= X'00' Default value.

= X'hex' Any character in hexadecimal representation. Non-printable characters are represented as smudge characters.

= 'char' Any character which defines the filler character.

spec1 and spec2 must be different from each other and different to the characters defined by means of @QUOTE.

If spec1 or spec2 are not special characters, @SYMBOLS is rejected with the error message: % EDT3952 INVALID SYMBOL

Filler characters at the end of a screen line are not included in the file. Filler characters within a record are converted into blanks when a screen line is entered for the first time or is modified.

In the F mode screen, the filler character between the end of the record and the end of the screen line is by default X'00' (null character). This ensures that the end of the record is recognizable and prevents blanks at the end of the record from being cut off inadvertently.

The use of `LZE` and `LZF` to delete a complete record is subject to certain limitations:

- `LZE` deletes all characters in a record as of the specified position.
- `LZF` deletes only the rest of the line; any characters in the next record are moved up.

A complete record in a column position other than 1 must be deleted explicitly via `@DELETE` or via statement code D.

The corresponding form of representation up to EDT V16.2 is set by means of `@SYMBOLS FILLER = ' '`.

Screen lines which consist only of filler characters other than ' ' are not entered in the file. Screen lines which consist only of filler characters = ' ' are created as records consisting of two blanks.

@SYNTAX Set syntax check and execution mode

@SYNTAX can be used to

- set the type of syntax check to be performed and
- activate or deactivate a test mode

for line-mode input (read with @EDIT, @EDIT ONLY or from SYSDTA).

Operation	Operands	F mode / L mode
@SYNTAX	[SECURITY { [=HIGH] }] [[,] TESTMODE { [=ON] }]	{ [=OFF] }

SECURITY

- = HIGH The syntax check used in L mode is the same as that used in F mode.
Main differences to LOW:
 - blanks in keywords are no longer skipped
 - comments at the end of a statement are permitted only if specified in the syntax description (e.g. @NOTE, @PROC, @END)
- = LOW Compatible syntax check of L mode.
This setting tolerates scattered comments in statements from the EDT V15 statement set. This type of comment does not preclude ambiguity, i.e. error situations are not always detected.

If the SECURITY operand is not specified, the current setting remains unchanged.

The @STATUS statement can be used to output the current syntax setting.

TESTMODE

- = ON Subjects statements to a syntax check without executing them (and without executing filter routines). Data lines entered in a line-mode dialog are not transferred to the work file. Inputs beginning with more than one statement symbol (e.g. @@...) are subjected to a syntax check.
The following statements are executed:
 - @LOG, @SYNTAX and redefinition of the EDT escape character (@:)
 - @STATUS, albeit with output to SYSOUT
 - @HALT and @RETURN, with implicit resetting of test mode
 The following statements and operands are not checked:
 - external statements (statements with the user escape character)
 - The <text> operand in the statements @SET, format 6, @IF, format 1 and @+, @-

In these cases the following message is output in interactive mode:
% EDT0110 TESTMODE: SYNTAX CANNOT BE TESTED

If there are no errors, an acknowledgment is output in interactive mode:
% EDT0100 TESTMODE: NO SYNTAX ERROR

For the purposes of the syntax check, the statement @3:@... should be split into the two statements @SET3 and @... .

= OFF Deactivates test mode. If @HALT or @RETURN is issued to terminate DIALOG mode, test mode is also deactivated.

If the TESTMODE operand is not specified, the execution mode still applies.

Example:

```
@SYNTAX SEC,TEST
```

This statement gives the user the possibility of checking old EDT procedures to determine whether or not they comply with the syntax described in the manual. Statements that fail to comply with this syntax should be corrected.

Data records and statements input via the LU15 interface are not affected by the setting of the test mode. This statement is not supported if EDT is called up only via the old line-mode-subroutine interface.

Logging in test mode

In L mode, not only the faulty statement, but also the the location at which the error was detected is marked with : .

If SECURITY=LOW is set in L mode and characters are skipped in the syntax check, EDT issues message % EDT0120 TESTMODE: CHARACTER(S) SKIPPED as a warning and marks the skipped characters with : .

Default value

SECURITY=LOW is the default

- in batch mode,
- in an L-mode dialog with system switch 5 set and
- at the LU15 interface.

In all other cases, the default value is SECURITY=HIGH. At the beginning of an EDT session, the test mode is deactivated.

@SYSTEM Enter system commands

@SYSTEM can be used

- to interrupt the EDT session and branch to the operating system (as with [K2](#))
- to execute an operating system command without interrupting the EDT session.

@SYSTEM is one of the EDT statements that is relevant to security (see [section "Data protection" on page 71](#)). In uninterruptible system procedures in interactive mode and in the case of input from a file, the statement will be rejected (unless it is read from SYSDTA=SYSCMD).

Operation	Operands	F mode / L mode
@SYSTEM	[string [TO ln [(inc)]]]	

string

A string specifying the system command to be executed.

The string may be specified:

- explicitly, enclosed in single quotes, or
- implicitly in the form of a line number, a line number variable or a string variable (in each case with a column range, if required).

The command is executed immediately and control is then returned to EDT. The system commands EXIT-JOB, LOGOFF, HELP-SDF, CALL-PROCEDURE, START-PROGRAM and LOAD-PROGRAM and all user commands defined via SDF-A and implemented by means of command procedures abort or terminate the EDT session and unload EDT.

The command may be specified with or without a slash at the beginning. The default value LOWER OFF means that lowercase letters are converted to uppercase; in LOWER mode this does not happen.

Only commands which may be entered using the CMD macro may be used. If a command is not permitted at the CMD interface or in the current SDF syntax file (as of SDF V4.1), it is rejected with an error message. See the "Executive Macros" manual [\[8\]](#) for details on the CMD macro and the permissible commands.

If "string" is not specified, control is passed to the operating system. The RESUME-PROGRAM command causes the EDT session to be resumed where it was interrupted by means of the @SYSTEM statement. In uninterruptible system procedures, this is not possible in interactive mode.

- In** When In is entered, any system command which is not output with MODE=PHYS or MODE=FORM is transferred to the current file, beginning at line number In.
In this case, EDT requires an additional buffer, which it requests via the REQM macro.
In can also be specified by line number variables (#L0-#L20) or symbolically (e.g. %, \$).
- inc** Increment between the line numbers following In. If inc is not specified, the implicitly defined increment is used.

The current line number is changed if a line is created whose number is higher than the previous highest line number.

If a file has been opened by means of @OPEN for real processing, the user should never terminate the EDT session by means of @SYSTEM, since the file will then not be closed.

Instead of using @SYSTEM 'START-PROGRAM...' or @SYSTEM 'LOAD-PROGRAM...', users should use @EXEC or @LOAD because EDT closes any files that are still open and deletes any autosave files that exist before executing these latter statements (see @AUTOSAVE).

In the case of the error message %EDT4300 ERROR AT SYSTEM COMMAND, the output of error messages includes the message code derived from the command return code.

@TABS Set tabs

@TABS is used to

- define a tab character and up to 8 positions (columns)
- display the current tab character and the appropriate columns
- evaluate the tab characters in work files and in string variables
- define up to 8 positions for the hardware tabulator
- activate and deactivate the hardware tabulator function.

The tab character is not evaluated in batch mode and in EDT procedures (@DO and @INPUT procedures) in the case of input from RDATA or when processing an EDT procedure.

Operation	Operands	F mode / L mode
@TABS	$\left\{ \begin{array}{l} :: [\text{tab} [[:] \text{cl1} [, \text{cl2}, \dots]] \left\{ \begin{array}{l} \text{CHECK} \\ \text{FORWARD} \end{array} \right\} [\text{cl}]]] \\ \text{RANGE} [= \text{range}] \\ [\text{cl1} [, \text{cl2}, \dots]] \{ \text{ON} \mid \text{OFF} \} \\ [:] \text{VALUES} \end{array} \right\}$	

tab The character for the software tabulator, which EDT is to interpret as the tab character from now on.
 The semicolon (;) must not be used as a tab character in F mode since it is interpreted as a statement delimiter. "tab" must be followed by ":" whenever it stands for one of the characters C, F or V, or for a digit.

cl1,cl2,... The numbers, separated by commas, of up to 8 columns to which the cursor can be positioned using this tab character. EDT positions to these columns in the precise order in which they are specified here; the column numbers must therefore be entered in ascending order. Any value between 1 and 255 may be specified.

If the positional operand :: is specified, the values refer to the software tabulator.

If :: is not specified, the values refer to the hardware tabulator. In this case, EDT checks whether the values have been specified in ascending order, and rejects @TABS with an error message if this is not the case.

- CHECK** If EDT positions the cursor to a column, it does not normally check whether this column already contains text which was entered before the tab character; any such text is overwritten.
- CHECK causes EDT to issue a warning message on the screen in such cases. In addition, it causes the evaluation of tab characters (@TABS RANGE statement) to be aborted when positioning in the reverse direction. The current line and all following lines are not affected by this.
- FORWARD** Prevents backward positioning. EDT still moves to the tab positions in the specified order; if, however, the new tab position lies on or before the last column into which text was written, EDT skips this tab position and moves to the next one.
- CHECK and FORWARD are both off when EDT is started.
- cl** (Only in L mode).
- cl changes the value for checking the line length ($1 \leq cl \leq 256$). EDT checks each text input to determine whether it is longer than cl columns. The default value for cl is 256. If a text input consists of more than cl columns, the line is still created (with a maximum length of 256 characters), but EDT displays a warning message on the screen.
- The value for checking the line length can also be changed by means of @CHECK (L mode).
- If the user switches to F mode, the value for cl is changed back to 256. There is a corresponding function in F mode (see @PAR LIMIT).
- RANGE** The software tab characters are evaluated in the specified line range in accordance with the current definition.
- The operands tab, cl1, cl2,..., CHECK and FORWARD of the previous @TABS statement are taken into account.
- The maximum line length is 256 characters.
- range** Line range consisting of:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30)
- A line range may also be specified using the current line range symbol (see @RANGE), by means of symbolic line numbers (e.g. %, \$) or via line number variables. String variables (#S0 bis #S20) may also be used.
- If "range" is not specified, all lines in the file are processed.

ON	If positions for the hardware tabulator have already been defined in the same statement or in a previous one, the function is activated, i.e. the cursor positions itself to the next defined column by means of <code>→</code> . The default value is ON.
OFF	The hardware tabulator function is deactivated. The defined positions are retained and can be reactivated by means of <code>@TABS ON</code> .
VALUES	The current software tabulator character and the associated tab positions are displayed. If the hardware tabulator has been defined, only the tabulator positions are displayed. If no tab character is defined, the statement is ignored. In batch mode, output is directed to SYSLST.

If `@TABS::` is specified without further operands, the tabulator is set as not defined (regardless of whether it is a hardware or software tabulator).

If `@PAR EDIT LONG = ON` is set, the hardware tabulator can only be used to position to positions encompassed by the screen width; positions lying outside this range are ignored.

The hardware tabulator is not supported on the 3270 Data Display Terminal.

If the hardware tabulator is set and activated, `[EFG]` and `[AFG]` can only be used to insert and delete text within the specified tab positions. When working with strings that overshoot the field boundaries, it is advisable to switch temporarily to `@TABS OFF`.

Within statements, tab characters are recognized and processed as such only if they are specified in the "text" operand (see `@SET`, format 6).

If a text which has been entered contains more tab characters than the number of defined tab positions, EDT treats the surplus tab characters as normal text characters.

Column justification with the aid of the tabulator function is implemented only when new records are entered. If, for example, a record is inserted via copying, at least one character must be overwritten, modified or inserted.



The main application of `@TABS` is to permit files, such as source code files, to be created with specific information in specific columns. In a file for Assembler programs, for example, a suitable setting is `@TABS:::10,16,40 CHECK 71`. `CHECK` ensures that a message is issued if excessively long names are entered (`FORWARD` instead of `CHECK` would not achieve the desired effect). Specifying a maximum line length of 71 is a good idea, since it prevents inadvertent overwriting of column 72, which is reserved for the continuation character.

Example: software tabulator

```

23.00 .....
tabs ::[: 10, 16, 40 .....0000.00:001(0)

```

[is defined as the tab character and the tab positions are 10, 16 and 40.

```

1.00 [ba]r[14,15[sub]rutine [ then return ] .....
2.00 sprung[dc[c' all correct' .....
3.00 .....

```

The text is entered with tab characters. Note that only three tab positions were defined, but the first text line contains four tab characters.


```

1.00          BALR  14,15          SUBROUTINE [ THEN RETURN ]..
2.00 LABEL    DC C  ' ALL CORRECT'.....
3.00 .....

```

As can be seen, the text is aligned on columns 10, 16 and 40. The fourth tab character is regarded by EDT as a normal text character, since only three tab positions were defined.

Example: hardware tabulator

The same result (except for the CHECK function) is achieved when @TABS 10,16,40 is entered as a statement and positioning is implemented by .

Example

The characters /t in a UFS file are edited so that EDT aligns this file on a column boundary.

```

@XCOPY F=/edt/table
@ON & C A X'05' TO '!'
@TABS ::!:8,16,24,32,40 FORWARD
@TABS RANGE=&

```

@TMODE Display task information

@TMODE provides the user with information about the task under which EDT is running. The information is returned in the form of a message.

Operation	Operands	F mode / L mode
@TMODE		

The following information is returned about the task under which EDT is running:

- TSN the task sequence number
- USERID the user ID specified in the LOGON command
- ACCOUNT the account number of the task
- CPU-TIME the CPU time used
- DATE the date (YYYY-MM-DD)
- TIME the time of day
- STATEMENT SYMBOL the current statement symbol (e.g. @)
- TERMINAL the terminal type

Example

```

23.00 .....
tmode.....0000.00:001(0)

```

Information about the task is requested.

```

22.00 .....
% EDT0300 OLQA USER1 12345       1.2647 2007-07-30 15:12:19 @ 9763
.....0000.00:001(0)

```

@UNLOAD Unload module

@UNLOAD is used to unload modules which have been loaded using @RUN or @USE.

Operation	Operands	F mode / L mode
@UNLOAD	(name)	

name The name of the object module or load unit which is to be unloaded.

If the module cannot be unloaded, @UNLOAD is rejected with an error message and the EDT error switch is set.

Possible reasons:

- incorrect module name specified
- module is already unloaded
- module is loaded in shareable mode
- "name" is not a module name, but the name of a CSECT or ENTRY.

@UNSAVE Delete file

@UNSAVE deletes a specified file.

x

Operation	Operands	F mode / L mode
@UNSAVE	'file' [(ver)]	

file The name of the file to be deleted.
"/" must not be specified, even if a file link name has been defined by means of a SET-FILE-LINK command.

ver The version number of the file.
This may consist of up to three digits or an asterisk (*). * designates the current version number. If * is specified, the current version number is displayed on the screen before the file is deleted. If an incorrect version number is specified, the correct version number is displayed, but the file is not deleted.



Unlike @ELIM, @UNSAVE also deletes the catalog entry for the file.

@UPDATE Update records

@UPDATE updates or corrects records, adds records to the file, or displays records, edited for correction, on the screen. The statement has three formats.

@UPDATE (format 1) Update records

Existing records are updated or deleted, either completely or only within a specified column range.

New records are created.

Operation	Operands	L mode
@UPDATE	In [:domain] ; text	

In The number of the line to be updated or created.
The minimum value is 0.0001, the maximum 9999.9999.
In may also be specified as a line number variable (#L0 to #L20) or symbolically (e.g. %, \$).

domain Column range consisting of:

- one individual column (e.g. 10-10)
- one contiguous column range (e.g. 15-25)

whose contents are replaced by the string specified after the semicolon (;).
If the line does not yet exist, or if it is shorter than specified by the first column number, it is filled with blanks up to this point.

If only one column number is specified, the second column number is assumed to be 256.
If no column range is specified, the default domain defined by means of @UPDATE, format 3, is used. The default column range when EDT is started is 1-256.

text The string specified after ";" replaces the text in "domain". "text" may also be an empty string.

Tab characters in "text" are not evaluated.

"text" may also begin with the statement symbol (@ or any user-defined symbol) without being regarded as a statement.

This statement does not change the current line number unless the last line of the file is deleted by specifying an empty string for "text".

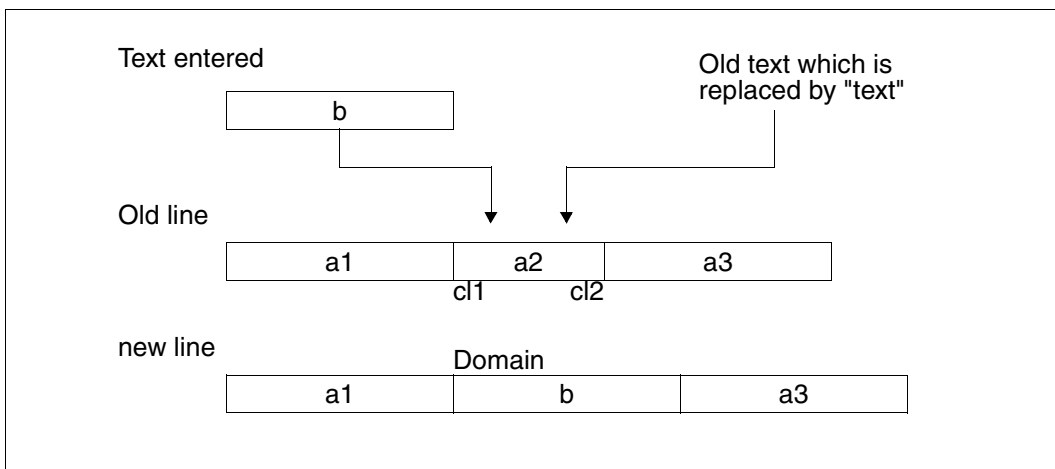


Figure 10: Updating a line using @UPDATE, format 1

The text entered using @UPDATE (b) replaces precisely the text a2 in the specified column range clrng. Since the new text b is longer than the old text a2, the following text a3 in the record is moved to the right. If the new text were shorter than the old text, the remaining text in the record would be moved to the left to fill the resulting gap.

@UPDATE (format 2) Display edited records

A section of a file is displayed on the screen in edited form for input by means of @UPDATE, format 1.

This format is ignored in batch mode and rejected in EDT procedures.

Operation	Operands	L mode
@UPDATE	[[ln] [:domain]	

ln This line and the following lines are displayed in edited form. All lines up to the end of the file or - if the file is too large - one "screenful" of lines are displayed. The number of lines displayed in one screen is the default value set by the system for the display terminal being used or the number set by the user by means of the @VDT statement.

domain Column range consisting of:

- one individual column (e.g. 10-10)
- one contiguous column range (e.g. 15-25)

whose contents are output.

If only one column number is specified, the second column number is assumed to be 256.

If no column range is specified, the default domain defined by means of @UPDATE, format 3, is used. Lines which are shorter than the first column number are skipped in the output. The default domain when EDT is started is 1-256.

A prerequisite for working with @UPDATE, format 2, is that block mode is active. If the user has switched block mode off (BLOCK ON is the default value), this statement switches block mode on internally. The message BLOCK ON is displayed to indicate that this has been done.

@UPDATE, format 2, must always be the last statement in a statement block, since all remaining statements in the statement block which have not yet been executed are lost. The end-of-line indicator (logical end of line) must be set if it has been overwritten.

```
@Uxxxx.xxxx;text<
```

or, if a column range other than the default set by means of @UPDATE, format 3, is specified:

```
@Uxxxx.xxxx:yyy-yyy;text<
```

@	The current statement symbol.
xxxx.xxxx	The EDT line number.
yyy-yyy	The column range.
text	The file contents, edited in accordance with the input mode.
<	The logical end of line.

If one screen is insufficient to display all lines from ln to the end of the file, the current line number is displayed in the last screen line in the format:

@Nxxxx.xxxx

This indicates that a new line with the line number xxxx.xxxx will be created if data is entered. Further lines will then be created with numbers determined by the current increment value.

The input block following the output is processed without restrictions. It may consist of modified @UPDATE statements, new statements, or new data. Scrolling with * + - 0 is not possible, since these are not true statements and are therefore regarded, within an input block, as data records.

[K1] or an empty input may be used to scroll towards the end of the file.

This statement changes the value of the line number symbol ? (the line number of the first line in which a hit was found after @ON). This symbol always points to the first line of the next file section to be displayed. (If the end of the file is reached, it contains the highest existing line number.)

Consequently, if ? is entered for ln in @UPDATE, the next file section will be displayed in edited form.

i If the file section which is displayed contains characters which cannot be represented in character mode @INPUT CHAR, corrections must be implemented in hexadecimal mode.

In character mode, such lines are displayed as:

@Nxxxx.xxxxztext<
 or
 @Nxxxx.xxxx:yyyztext<

z The device-specific smudge character.

Nonprintable characters in "text" are also displayed as z (the device-specific smudge character). @N for @NOTE causes the statement to be ignored in an input block.

@UPDATE (format 3) Define default column range

@UPDATE, format 3 can be used to define a default column range for formats 1 and 2 of @UPDATE.

Operation	Operands	L mode
@UPDATE	COLUMN [domain]	

domain

Column range consisting of:

- one individual column (e.g. 10-10)
- one contiguous column range (e.g. 15-25)

whose contents are output.

If only one column number is specified, the second column number is assumed to be 256.

If no column range is specified, the default column range 1-256, set when EDT is started, is used.

@USE Define external statement routines

With the aid of @USE, the user can

- define external statement routines (see the "EDT Subroutine Interfaces" manual [1]) and
- define escape symbols with which the statement routines can be called.

@USE is one of the EDT statements that is relevant to security (see [section "Data protection" on page 71](#)). In uninterruptible system procedures in interactive mode and in the case of input from a file, the statement will be rejected (unless it is read from SYSDTA=SYSCMD).

Operation	Operands	F mode / L mode
@USE	COMMAND = 'usersymb' [({entry}) [,modlib]]	

usersymb The user escape symbol for the external statement routine. This may be any character except the statement symbol, a semicolon (;) or a blank. If EDT is called as a subroutine (CMD function, version 2), an empty string may also be defined as the escape symbol (special "statement filter" application, see the description of the external statement routines in the "EDT Subroutine Interfaces" manual [1]). The enclosing single quotes can be redefined by means of @QUOTE.

entry The entry point of the external statement routine. entry may also be specified by means of a string variable in the form <str-var>. The module or load unit is loaded immediately.

***** The entry point name of the statement routine is displayed when the external statement is entered (see the "EDT Subroutine Interfaces" manual [1]). The module is not loaded until the external statement is entered.

modlib The name of the library in which the module or load unit is stored. modlib may also be specified by means of a string variable in the form <str-var>. If the module or load unit is not found in the specified library, the system first searches for it in the alternate libraries BLSLIBxy and then in the private tasklib or in the system tasklib \$TASKLIB, as appropriate. If no library is specified, the system first searches the private tasklib and then the system tasklib \$TASKLIB. If the module is not found, an error message is issued.

Up to five different escape symbols can be defined.

If (entry[,modlib]) is not specified, the statement routine defined previously by the specified symbol is deactivated.

The module or load unit can be unloaded using @UNLOAD. The associated escape symbol is then invalidated.

See the "EDT Subroutine Interfaces" manual [1] for details of the interface to the external statement routine.

EDT-specific entries and module names are rejected with the following error message:
% EDT4933 MODULE LOADING NOT POSSIBLE

Example 1

Entry point defined by means of @USE

```

Entry name      : JOBVAR
Syntax         : CATJV <name>
                ERAJV <name>
                GETJV <name>,<ln>
                SETJV <name>,<ln>
Declaration    : @USE COMMAND = '*' (JOBVAR,PRIVLIB)
Application    : *CATJV JV.TEST   ==> Call module JOBVAR with parameter
                'CATJV JV.TEST'
                *SETJV JV.TEST,3 ==> Call module JOBVAR with parameter
                'SETJV JV.TEST,3'

```

Example 2

Entry point defined by means of an external statement

```

Declaration    : @USE COMMAND = '*' (*,PRIVLIB)
Application    : *SORT 20-100    ==> Call module SORT with parameter
                '20-100'
                *HELP EDT5100    ==> Call module HELP with parameter
                'EDT5100'

```

@VDT Control screen output

@VDT changes the number of lines displayed at one time during output, one screen at a time, of a virtual file or of a file opened by means of @OPEN (see @PRINT V).

Operation	Operands	F mode / L mode
@VDT	[int] [,] [F1 F2]	

int Specifies how many lines are to be displayed in one screen in F mode. The specified value must be less than or equal to the default value defined by the system for the data display terminal being used. If int is omitted, EDT uses the default value defined by the system.

F1 Specifies the default value preset by the system (24 lines and 80 columns).

F2 Specifies the screen format with 27 lines and 132 columns. The F2 operand cannot be specified unless the 9763 Data Display Terminal supports this screen format. Otherwise it is either rejected with an error message (F mode) or ignored (L mode).

If F1 or F2 is specified without int, the appropriate line number for L mode is set at the same time.

@VDT without parameters applies globally to all work files.

When EDT is called up in F mode, the standard format is preset. If the user switches over to L mode, the format setting is retained until the first @VDT statement is issued.

At the start of an EDT session, the user can issue @STATUS=VDT to find out how many lines are displayed in the standard format for the display terminal being used.

The operand F1 or F2 is only accepted with the 9763 Data Display terminal. In F mode it is rejected with the message % EDT4945 NOT POSSIBLE ON THIS TERMINAL and in L mode it is ignored.

In F mode, @VDT terminates the processing of a statement line, i.e. anything left in the statement line is not processed.

If EDT is interrupted using [K2], the preset screen format is reconstructed via [K3] after RESUME-PROGRAM has been entered to return to F mode. Upon an interruption in L mode, the preset screen format is reactivated after a return by means of SEND-MESSAGE TO=PROGRAM.



- @VDT sets up the current window with the desired format, which remains valid until the next time the statement is called.
- @VDT implicitly leads to @PAR SPLIT = OFF.
- In batch tasks, the @VDT statement is ignored.

@VTCSET Control screen output

@VTCSET specifies whether line mode control characters (see the "Executive Macros" manual [8], macro WRTRD, operand MODE=LINE) in file contents are to be evaluated during output or converted to device-specific smudge characters.

Operation	Operands	F mode / L mode
@VTCSET	[ON] OFF	

- ON Specifies that output is not to be checked and control characters are thus not to be converted into smudge characters.
- OFF Specifies that line mode control characters in the file are to be converted into device-specific smudge characters during output. All characters which cannot be displayed are also converted to smudge characters.

This statement has no effect for outputs sent to SYSLST (@LIST) in batch tasks.

@WRITE Write file or library element

@WRITE has two formats with the following functions:

- writing the contents of the current work file into a SAM file (format 1)
- writing the contents of the current work file into a library element (format 2).

@WRITE (format 1) Writing the contents of the current work file into a SAM file

@WRITE writes all or part of the virtual file or the file opened by means of @OPEN into a SAM file on disk or tape.

The SAM file is physically open only during execution of the @WRITE statement.

Operation	Operands	F mode / L mode
@WRITE	['file'] [(ver)] [range*] [:col:] [KEY] [{ UPDATE OVERWRITE }]

- file** The file name.
 If no file with this name already exists, a file with this name is cataloged. If "file" is omitted, the explicit local @FILE entry is used as the file name; if this does not exist, the global @FILE entry is used; if this does not exist either, the implicit local @FILE entry is used (see also @FILE); otherwise, @WRITE is rejected with an error message.
 If the file link name EDTSAM is assigned to a file, it is sufficient to enter '/' in order to write into this file (see [section "File processing" on page 49ff](#)).
- ver** The version number of the file.
 This may consist of up to three digits or an asterisk (*). * designates the current version number.
- range*** A line range, specified as:
- one or more line numbers, separated by commas (e.g. 4,6,15)
 - one or more line ranges, separated by commas (e.g. 5-10,17-19)
 - a combination of line numbers and line ranges (e.g. 4,7-23,8,15-30).
- The line range may also be specified using the current range symbol (see @RANGE), by means of symbolic line numbers (e.g. %, \$) or via line number variables. String variables must not be used.
 If range* is not specified, all lines in the file are written to the SAM file.

- col** A column range, specified as:
- one or more columns, separated by commas (e.g. 10,15,8)
 - one or more column ranges, separated by commas (e.g. 15-25,18-23)
 - a combination of columns and column ranges (e.g. 10,14-29,23-50,17).
- Columns and column ranges may be repeated and may overlap each other. If no column range is specified, the full length of each line is stored.
- KEY** When the SAM file is written, each line is preceded by an 8-character key which is formed from the line number. This permits the file to be read again later with precisely the same line numbers (see @READ with operand KEY.)
- UPDATE** This is meaningful only if there is already a SAM file with the specified name.
UPDATE causes the lines which are to be stored to be appended to the end of the existing SAM file.

If UPDATE is not specified, the entire SAM file is overwritten, i.e. its old contents are deleted.
- OVERWRITE** Suppresses the query OVERWRITE FILE? (Y/N). An existing file with the same name is overwritten without further checks. If the specified file does not exist, OVERWRITE has no effect.

If neither UPDATE nor OVERWRITE is specified and a file with the same name already exists, EDT issues the messages:

```
% EDT0903 FILE 'file' IS IN THE CATALOG, FCBTYPE = fcbtype
% EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO)
```

If the user responds with

N @WRITE is not executed;

Y @WRITE is executed and the existing file is overwritten with the contents of the current work file as a SAM file.

In the case of variable-length records (RECORD-FORMAT = VARIABLE) records are lost as of position 257 during the write operation.



If * is specified as the version number, the current version number is displayed on the screen (after the file has been written to disk, not during the save query). A new version number is created when a file is created for the first time or when an existing file is updated. In the second case, the existing version number is incremented by 1, while a new file receives the version number 1 when it has been written to disk. The version number is incremented each time the file is updated, up to a maximum of 255; the next new version number is then 0. Version numbers are provided in

order to preclude the inadvertent overwriting of files. If, namely, an incorrect version number is specified, the correct version number is displayed on the screen, but the current work file is not written to the disk.

Interaction with XHCS

If the XHCS subsystem is installed, the @WRITE statement transfers a coded character set name (CCSN) as a code attribute after the file has been written back.

@WRITE assigns the CCSN currently valid in EDT, regardless of whether the file already exists and what CCSN it has.

Example

```

1.      A VERY SHORT FILE ----- (01)
2.      @WRITE 'TEST.@WRITE.1' ----- (02)
2.      @FILE 'TEST.@WRITE.1' ----- (03)
2.      @WRITE UPDATE ----- (04)
2.      @DELETE ----- (05)
1.      @READ ----- (06)
3.      @PRINT
1.0000 A VERY SHORT FILE
2.0000 A VERY SHORT FILE ----- (07)
3.

```

- (01) One line is written into the virtual file.
- (02) This line is written to disk as the file TEST.@WRITE.1.
- (03) The file name TEST.@WRITE.1 is declared by means of @FILE.
- (04) @WRITE now refers to the file name declared in step (03). UPDATE causes the contents of the virtual file - still the line created in step (01) - to be appended to the file TEST.@WRITE.1.
- (05) The contents of the virtual file are deleted.
- (06) The file TEST.@WRITE.1 is read into the virtual file (again, no file name needs to be entered).
- (07) As we can see, the line was appended to the file in step (04).

@WRITE (format 2)**Writing the contents of the current work file into a library element or file**

@WRITE writes the contents of the current work file into a library element or file. The contents of the work file are retained (cf. @CLOSE). If the library or library element does not yet exist, the library or library element or the file is created by means of @WRITE.

Operation	Operands	F mode / L mode
@WRITE	$\left[\left. \begin{array}{l} \text{LIBRARY=path1 ([ELEMENT=]elemname [(vers)][,elemtyp])} \\ \text{ELEMENT=elemname [(vers)][,elemtyp]} \\ \text{FILE=path2 [,TYPE=ISAM SAM]} \end{array} \right\} \right]$ [,MODE= <u>ANY</u> UPDATE NEW REPLACE]	

At least one operand must be specified.

If more than one operand is specified, the operands must be separated from each other by commas or blanks.

LIBRARY = path1 (ELEMENT=elemname [(vers)][,elemtype])

The name of the library and the element.

ELEMENT = elemname [(vers)][,elemtype]

The name of the element, without a library name.

In this case, the library name must have been set previously by means of @PAR ELEMENT-TYPE.

path1 The library name.

path1 may also be specified by means of a string variable.

If path1 is omitted, the default library name specified by means of @PAR LIBRARY is used.

elemname The element name.

elemname may also be specified by means of a string variable.

vers The version number of the desired element (see the "LMS" manual [14]). If vers is not specified or if *STD is specified, the element with the highest possible version (X'FF', represented as @) is created or replaced.

elemtyp The element type.

elemtyp may also be specified by means of a string variable.

Permissible type entries are: S, M, P, J, D, X, *STD or user-defined type names with appropriate base type. If no type is specified, the type set in @PAR ELEMENT-TYPE is used.

Users who specify a user-defined type name are responsible for ensuring that its associated base type corresponds to one of the permissible types S, M, P, J, D or X.

Type	Contents
S	Source programs
M	Macros
P	Data edited for printing
J	Procedures
D	Text data
X	Data in any format

***STD**

Type S is the default value when EDT is started. Any other permissible type designation may be specified as the default value by means of @PAR.

- FILE = path2** Stores the work file in a BS2000 file.
- path2** Fully qualified file name.
path2 may also be specified by means of a string variable.
- TYPE** Defines the access method of the file.
- = SAM Default value. The file to be saved as a SAM file.
- = ISAM The file to be saved as an ISAM file.
- MODE** Defines the open mode for the library element or file.
The work file is written to
- = ANY a new or existing library element or file. If the library element or file does not yet exist, it is created by means of @WRITE.
- = NEW a new library element or file, i.e. there is at present no library element or file with this name. The new library element or file is created by means of @WRITE. The contents of the current work file are written to the library element or file, as appropriate. The library element is then closed (implicit @OPEN and @CLOSE).
- = REPLACE an existing library element or file, the contents of this element or file being cleared before the work file is stored, or a new library element or file created by means of @WRITE.
- = UPDATE an existing library element or file, the contents of this element or file being cleared before the work file is stored. If an existing library element or file has

been opened using @OPEN (format 2), @WRITE only stores intermediate states. The library element or file remains open until it is closed using @CLOSE.

If an existing library element or file has been opened using @OPEN (format 2), the path, elemname and elemtype or path2 operands may be omitted if the MODE operand is specified. The contents of the library element or file are replaced by the contents of the work file. The library element or file remains open until it is closed using @CLOSE.



Since MODE = ANY is the default value, any existing library elements or files will be overwritten without a warning message.

Example

```
@WRITE LIBRARY = PROGLIB (ELEMENT = SYNT)
```

The current work file will be written into element SYNT of program library PROGLIB.

```
@WRITE ELEMENT = PROC.TSCHO, J
```

The current work file will be written into element PROC.TSCHO with element type J (contains a procedure). The library in which the element PROC.TSCHO is to be stored must have been defined previously by means of @PAR LIBRARY.

Interaction with XHCS

If the XHCS subsystem is installed, the @WRITE statement transfers a coded character set name (CCSN) as a code attribute after the file or library element has been written back.

@WRITE assigns the CCSN currently valid in EDT, regardless of whether the file or library element already exists and what CCSN it has.

@XCOPY Read POSIX file

@XCOPY is used to copy a POSIX file stored in the POSIX file system into the current work file.

This function is supported as of BS2000/OSD V2.0. POSIX must be activated as subsystem.

Operation	Operands	F mode / L mode
@XCOPY	FILE=xpath [,CODE= <u>EBCDIC</u> ISO]	

FILE	Copies the data of a POSIX file into the work file.
xpath	Path name of the POSIX file relative to the home directory. Subdirectories may be specified, provided the entry does not exceed the maximum length of 256 characters. xpath may also be specified as a string variable.
CODE	Specifies the code in which the file data exists. If no entry is made for the CODE operand, the default set with @PAR CODE is used.
=EBCDIC	EDT is to interpret the data as EBCDI code. The data is not converted when read or written; instead, it is transferred in binary form. The X'15' symbol is evaluated as the record separator.
=ISO	EDT is to interpret the data as ISO code. When read, the data is converted to EBCDIC. The X'0A' symbol is evaluated as the record separator.



If the POSIX file name contains lowercase letters, @PAR LOWER=ON must be activated before @XCOPY is entered.

Calculation of line numbers when reading the file

1. Default numbering with the default increment 1.0000 or
2. Numbering with a fixed increment as defined by @PAR INCREMENT or
3. Automatic numbering with @PAR RENUMBER=ON
(see "Calculation of line numbers when reading the file" in the section on the @OPEN statement).

The current line number is changed if a line is created whose number is higher than the previous highest line number.

Interaction with XHCS

If the current work file is to be written into a BS2000 file with a certain CCS name (by means of @WRITE, @SAVE), the CCS name must be set using the @CODENAME statement before the @XCOPY statement is issued.

If this was not done, the user must enter the CCS name with the SET-FILE-ATTRIBUTE command after the BS2000 file has been written.

@XOPEN Open and read POSIX file

@XOPEN can be used to

- open a POSIX file stored in the POSIX file system,
- read the POSIX file into the current work file or
- create a new POSIX file in the POSIX file system.

This function is not supported until BS2000/OSD-BC V2.0. POSIX must be activated as subsystem.

Operation	Operands	F mode / L mode
@XOPEN	FILE=xpath [,CODE= <u>EBCDIC</u> ISO] [,MODE= <u>ANY</u> UPDATE NEW REPLACE]	

FILE Opens and reads in a POSIX file.

xpath Path name of a POSIX file in relation to the home directory. Subdirectories may be specified, provided the entry does not exceed the maximum length of 256 characters. xpath may also be specified as a string variable.

CODE Specifies the code in which the data exists and how the data is to be saved when it is written into the file. If no entry is made for the CODE operand, the default set with @PAR CODE is used. In the work file, the data is always in EBCDI code.

=EBCDIC EDT is to interpret the data as EBCDI code. The data is not converted when read or written; instead, it is transferred in binary form. The X'15' symbol is evaluated as the record separator.

=ISO EDT is to interpret the data as ISO code. When read, the data is converted to EBCDIC. If written back to the same POSIX file using @XWRITE or @CLOSE, the data of the work file is converted into the appropriate ISO variant. The X'0A' symbol is evaluated as the record separator.

MODE	Specifies the open mode for the file
= ANY	Default value Opens an existing or a new file for processing.
= UPDATE	Opens an existing file for processing.
= NEW	Creates a file in the current directory; the file must not exist already.
=REPLACE	Replaces the contents of the existing file. The contents are not read into the work file.

If the file has already been opened in another work file or the current work file is not empty, an error message is issued.



If the POSIX file name contains lowercase letters, @PAR LOWER=ON must be activated before @XOPEN is entered.

Calculation of line numbers when reading the file

1. Default numbering with the default increment 1.0000 or
2. Numbering with a fixed increment as defined by @PAR INCREMENT or
3. Automatic numbering with @PAR RENUMBER=ON
(see "Calculation of line numbers when reading the file" in the section on the @OPEN statement).

After the file has been read, the current line number is set to the value of the last line read, plus the current increment.

Interaction with XHCS

If the current work file is to be written into a BS2000 file with a certain CCS name (by means of @WRITE, @SAVE), the CCS name must be set using the @CODENAME statement before the @XCOPY statement is issued.

If this was not done, the user must enter the CCS name with the SET-FILE-ATTRIBUTE command after the BS2000 file has been written.

Terminating EDT

If, when EDT is being terminated (@HALT, @END, @RETURN), a file is opened with @XOPEN and save confirmation % EDT0900 is displayed, the POSIX file name will be displayed in the form 'X=xpath'.

@XWRITE Save contents of current work file to POSIX file

@XWRITE is used to write the contents of the current work file to a POSIX file in the POSIX file system. The work file continues to exist. If the destination file does not yet exist, it is created.

This function is not supported until BS2000/OSD-BC V2.0. POSIX must be activated as subsystem.

Operation	Operands	F mode / L mode
@XWRITE	FILE=xpath [,CODE= <u>EBCDIC</u> ISO] [,MODE= <u>ANY</u> UPDATE NEW REPLACE]	

- FILE** Writes the data into a POSIX file.
- xpath** Path name of the POSIX file relative to the home directory. Subdirectories may be specified, provided the entry does not exceed the maximum length of 256 characters. xpath may also be specified as a string variable.
- CODE** Specifies the code in which the data is to be saved. If no entry is made for the CODE operand, the default set with @PAR CODE is used.
- =EBCDIC** The data is not converted when written; instead, it is transferred from the work file in binary form. The X'15' symbol is evaluated as the record separator.
 - =ISO** =ISOThe data in the work file is converted into the appropriate ISO code and written into the POSIX file. The X'0A' symbol is evaluated as the record separator.
- MODE** Specifies the open mode for the file.
- =ANY** Default value. Writes the work file into a new or existing file. If the file does not yet exist, it is created.
 - =UPDATE** Writes the work file into an existing file, overwriting any previous contents. If the specified file was opened with @XOPEN, the file's code remains unchanged, i.e. any entry for the CODE operand is ignored.
 - =NEW** Writes the work file into a newly created file, i.e. a file which must not have existed already.

=REPLACE

Writes the work file into an existing file, overwriting any previous contents. The code in which the data is stored may be changed.



If the POSIX file name contains lowercase letters, @PAR LOWER=ON must be activated before @XWRITE is entered.

If a file was opened with @XOPEN, the specification of the file name in @XWRITE may be omitted if the MODE operand is specified. The contents of the file are replaced with the contents of the work file. The file remains open until the @CLOSE statement is issued.



Since MODE=ANY is the default, existing files are overwritten without warning if no MODE entry is specified.

@ZERO-RECORDS Setting empty line mode

This statement enables empty lines to be handled when reading data from a file (POSIX, SAM, ISAM or library member) to an EDT work file or when data is written from an EDT work file to a file (POSIX, SAM, ISAM or library member).

Operation	Operands	F mode / L mode
@ZERO-RECORDS	[ON] OFF	

ON Has the effect that lines of length 0 or lines of length 8 are written into the EDT work file with the end-of-line character X'0D' as contents in the following cases.

Lines of length 0

- When reading from a POSIX file with @XOPEN or @XCOPY
- When reading from a SAM file with @READ, @OPEN or @COPY
- When reading from a library member with @OPEN or @COPY

Lines of length 8

- When reading from an ISAM file with standard properties (see [section "File processing" on page 49](#)) with @GET, @OPEN or @COPY
- When reading from a SAM file with @READ and the KEY operand

Has the effect that lines in the EDT work file that consist of just the end-of-line character X'0D' are written as lines of length 0 or length 8 in the following cases.

Lines of length 0

- When writing to a POSIX file with @WRITE or @CLOSE
- When writing to a SAM file with @WRITE (format 1 or 2) or @CLOSE
- When writing to a library member with @WRITE (format 2) or @CLOSE

Lines of length 8

- When writing to an ISAM file with standard properties with @SAVE, @WRITE (format 2) or @CLOSE
- When writing to a SAM file with @WRITE and the KEY operand

- OFF
- Has the effect that lines of length 0 are not written into the EDT work file when reading from a POSIX file, SAM file or a library member, and lines of length 8 are not written into the EDT work file when reading from an ISAM file with standard properties.
 - Has the effect that lines in the EDT work file that only consist of the end-of-line character X'0D' are also written as lines consisting of the character X'0D' when writing to POSIX files, SAM files and library members.
 - Has the effect that lines in the EDT work file that only consist of the end-of-line character X'0D' are also written as lines consisting of the record key and the end-of-line character X'0D' when writing to ISAM files with standard properties.

If the @ZERO-RECORDS statement is specified without operands, empty line mode is enabled.

Notes

- Empty line mode is disabled by default when EDT is started.
- When EDT is started from the POSIX shell (edt command), AUTOFORM mode (@BLOCK ON, AUTOFORM) is enabled. For POSIX files, this mode has the same effect as empty line mode.
- The current setting of empty line mode can be displayed using the @STATUS=MODES statement.
- In real processing of ISAM files (@OPEN format 1), switching empty character mode on or off has a direct effect on the subsequent changes to the file. When a SAM file with @OPEN format 1 is to be processed (AS operand), the required empty character mode must be set before the @OPEN statement.

Tips for handling empty lines in EDT

- | | |
|--------------------|---------------------------------------------------------------------------------------------------------------------|
| Fill empty lines | overwrite the X'0D' character with new contents |
| Create empty lines | enable hexadecimal mode (HEX ON)
insert line (e.g. with statement code 1)
insert X'0D' as the first character |
| Delete empty lines | as other EDT lines (statement code D, @DELETE). |

7 EDT messages

EDT0001 (&00) STARTED

EDT0100 TESTMODE: NO SYNTAX ERROR

Meaning

Test mode is set. There was no syntactical error. The statements have not been processed.
Error switch: not set.

EDT0110 TESTMODE: SYNTAX CANNOT BE TESTED

Meaning

Test mode is set. The syntax of the statement can only be tested at run time.
Possible reasons: indirect operands, operands in variables or user statements.
The statement has not been processed.
Error switch: not set.

EDT0120 TESTMODE: CHARACTER(S) SKIPPED
(B) Routing code: * Weight: 99

Meaning

Test mode is set. The syntax check in line mode skipped one or more characters. A strict syntax check with the option SECURITY=HIGH would possibly find an error.
Error switch: not set.

Response

See your EDT manual for the correct syntax of the statement. Correction to the therein described form will ensure processing in following EDT versions. The support of this statement is not guaranteed.

EDT0160 FILE '(&00)' WRITTEN
EDT0170 MEMBER '(&00)' IN LIBRARY '(&01)' REPLACED AND WRITTEN
EDT0171 FILE '(&00)' REPLACED AND WRITTEN
EDT0172 MEMBER '(&00)' IN LIBRARY '(&01)' CREATED AND WRITTEN
EDT0173 FILE '(&00)' CREATED AND WRITTEN
EDT0178 FILE '(&00)' CLOSED
EDT0190 WORK FILE (&00) EMPTY
EDT0192 FILE '(&00)' OPENED REAL IN WORK FILE (&01)
(B) Routing code: * Weight: 99
EDT0193 WORK FILE (&00) CLEARED
EDT0194 FILE '(&00)' CREATED AND OPENED REAL IN WORK FILE (&01)
EDT0195 FILE '(&00)' REPLACED AND OPENED REAL IN WORK FILE (&01)
EDT0200 CCS CHANGED TO '(&00)'
(B) Routing code: * Weight: 99

Meaning

By reading or opening a file or library element with the attribute (&00), EDT uses this Coded Character Set.

Error switch: not set.

EDT0210 ELEMENT(S) ADDED TO S-VARIABLE '(&00)'
(B) Routing code: * Weight: 99

Meaning

The SDF-P list variable (&00) has been extended by appending or prefixing one or more elements to the list.

Error switch: not set.

EDT0211 /FREE-VARIABLE COMMAND PROCESSED FOR S-VARIABLE '(&00)'
(B) Routing code: * Weight: 99

Meaning

The contents of the SDF-P variable (&00) have been destroyed. In the given statement @SETLIST with operand MODE=NEW the specified range did not contain any line or column.

Error switch: not set.

EDT0227 ISAM FILE '(&00)' CREATED AND OPENED IN WORK FILE (&01)
 EDT0228 ISAM FILE '(&00)' REPLACED AND OPENED IN WORK FILE (&01)
 EDT0229 ISAM FILE '(&00)' OPENED IN WORK FILE (&01)
 EDT0230 FILE '(&00)' OPENED IN CURRENT WORK FILE (&01)
 EDT0231 FILE '(&00)' CREATED AND OPENED IN CURRENT WORK FILE (&01)
 EDT0232 FILE '(&00)' REPLACED AND OPENED IN WORK FILE (&01)
 EDT0235 FILE '(&00)' WRITTEN AND CLOSED
 EDT0236 FILE '(&00)' CLOSED UNCHANGED
 EDT0242 FILE '(&00)' COPIED
 EDT0243 UFS FILE '(&00)' COPIED
 EDT0244 ALLOW WRITE ACCESS FOR READ ONLY FILE? REPLY (Y=YES; N=NO)

Meaning

This query is issued following a @XOPEN or a @XWRITE statement if the file is read only and the current user id is TSOS.

Error switch: not set.

Response

Y: the file will be overwritten / opened for writing

N: the file will not be overwritten / opened for writing.

EDT0258 MEMBER '(&00)' IN LIBRARY '(&01)' OPENED
 EDT0259 MEMBER '(&00)' IN LIBRARY '(&01)' CREATED AND OPENED
 EDT0264 MEMBER '(&00)' IN LIBRARY '(&01)' WRITTEN AND CLOSED
 EDT0265 MEMBER '(&00)' IN LIBRARY '(&01)' CLOSED UNCHANGED
 EDT0266 WORK FILE EMPTY: MEMBER '(&00)' CLOSED UNCHANGED

Meaning

The work file specified in the CLOSE or WRITE statement is empty.
 The member (&00) has been closed but not written back.

EDT0268 MEMBER '(&00)' IN LIBRARY '(&01)' OPENED FOR REPLACEMENT
 EDT0274 MEMBER '(&00)' IN LIBRARY '(&01)' COPIED
 EDT0281 /DELETE-FILE COMMAND PROCESSED FOR FILE '(&00)'

Meaning

The file has been erased from catalog.

EDT0282 DELETE PROCESSED FOR MEMBER '(&00)'

Meaning

The member has been deleted from library.

EDT0285 SDF: SYNTAX TESTED. (&00) ERROR(S) IN RANGE
(B) Routing code: * Weight: 99

Meaning

At the processing of statement @SDFTEST (&00) errors have been detected.
Error switch: not set.

EDT0290 ALL LINES ARE DIFFERENT

Meaning

All lines to be compared are different.
Error switch: EDT.

EDT0291 ALL LINES ARE EQUAL

Meaning

All lines to be compared are equal.
Error switch: not set.

EDT0292 COPY BUFFER CLEARED

Meaning

Acknowledgement following an '**' in the mark column.
Error switch: not set.

EDT0293 FILE NOT WRITTEN

Meaning

Either N has been specified in response to an OVERWRITE inquiry, or an error occurred when the file was written back.
Error switch: not set.

EDT0294 MAXIMUM LINE NUMBER

Meaning

When generating the screen, line number 9999 is exceeded.
No additional blank lines are provided at the end of the file.
For more detailed information on when the maximum line number (9999.9999) is generated see the "EDT" manual.
Error switch: not set.

EDT0295 OLD COPY BUFFER CLEARED, NEW COPY BUFFER FILLED

Meaning

An R mark is followed by a C or M mark.
The copy buffer created by means of R mark(s) has been cleared.
Error switch: not set.

EDT0296 OVERWRITE FILE? REPLY (Y=YES; N=NO)

Meaning

This query is issued following a @WRITE or a @SAVE statement if the file already exists.
Error switch: not set.

Response

Y: the file will be overwritten

N: the file will not be overwritten.

EDT0297 COMPARE RESULT IN WORK FILE (&00)

Meaning

The result of a successfully processed @COMPARE statement (format 2) is output to work file (&00).

Error switch: EDT.

EDT0298 ERASE ALL JOB VARIABLES '(&00)'? REPLY (Y=YES; N=NO)

Meaning

This query is issued following a @ERAJV statement, if the name was specified partially qualified or in wildcard syntax and this refers to more than one job variable.

Error switch: not set.

Response

Y: all job variables concerned will be erased from the catalog.

N: the statement will be aborted and no job variable will be erased.

EDT0299 JOB VARIABLES NOT ERASED

Meaning

Message EDT0298 (ERASE ALL JOB VARIABLES?) was answered with N.

Error switch: not set.

EDT0300 (&00)

Meaning

Following a @TMODE statement, the task attributes are displayed from left to right in this order:

TSN	- task sequence number
USER ID	- user ID in the /LOGON command
ACCOUNT	- account number of the task
CPU TIME	- CPU time used
DATE	- date (YYYY-MM-DD)
TIME	- time
'@'SYMBOL	- actual statement symbol
TERMINAL	- type of terminal.

EDT0600 LOGICAL LINELENGTH > LENGTH OF SCREENLINE
(B) Routing code: * Weight: 99

Meaning

At initiation time the logical linelength could not be fitted to the length of the screenline.
Error switch: not set.

EDT0610 BUFFER SIZE UNCHANGED
(B) Routing code: * Weight: 99

Meaning

The buffer for output to the screen could not be changed by EDT.
Error switch: not set.

EDT0650 UNABLE TO SUPPORT NATIONAL TERMINAL. STANDARD WILL BE USED
(B) Routing code: * Weight: 99

Meaning

The connected DSS is a national 7-bit terminal, but EDT can only support it with standard functions. Possible reasons:

- The DSS has been generated with wrong parameters or a variant has been used EDT cannot support yet.
- There is problem at XHCS or VTSU.

Response

Try to generate the DSS in a different way.

EDT0651 CCS '(&00)' INCOMPATIBLE WITH TERMINAL. STANDARD WILL BE USED

Meaning

A file or library element which was to be read or opened had the catalog attribute (&00), or the CCS (&00) was asked for at a @CODENAME statement. But on this terminal only files with a CCS attribute EDF03IRV or without a CCS attribute can be edited.

Response

Use the /MODIFY-FILE-ATTRIBUTE command to change or erase the CCS attribute of the file.

The statement @CODENAME should not be used on this terminal.

EDT0800 STATEMENT '(&00)' ONLY SUPPORTED UP TO THIS VERSION

Meaning

Statement '(&00)' will not be supported in this form in the next version of EDT.

Response

Please consult the "EDT" manual and replace statement (&00) by the correct one.

EDT0900 EDITED FILE(S) NOT SAVED!

Meaning

A @HALT statement has been entered in order to terminate EDT, but some dates have not yet been saved.

EDT will output a list of work files whose dates have not yet been saved.

Error switch: not set.

EDT0901 NO MATCH IN RANGE

Meaning

No match exists for the first string in the @ON statement.

If this error occurs while an EDT procedure (@DO) or an INPUT file is being processed, this message is not displayed and the EDT error switch is not set unless logging has been activated by means of the PRINT operand.

Error switch: EDT (see 'Meaning').

EDT0902 FILE (&00) VERSION (&01)

Meaning

In the statement @WRITE, @SAVE,... a version number or * was explicitly specified by the user. The version was correct and EDT outputs the current version number.

EDT0903 FILE '(&00)' IS IN THE CATALOG, FCBTYP = (&01)

EDT0904 TERMINATE EDT? REPLY (Y=YES; N=NO)

Meaning

EDT inquires whether EDT is to be terminated.

Error switch: not set.

Response

Y: EDT will be terminated

N: EDT will not be terminated.

EDT0905 EDITED MEMBER TO BE ADDED? REPLY (Y=YES; N=NO)

Meaning

Before returning control to LMS, EDT inquires whether the edited work file is to be saved by LMS.

EDT0906 REPEAT ATTEMPT? REPLY (Y=YES; N=NO)

Meaning

If there is not enough virtual memory space to process the statement, the statement can be repeated after appropriate measures have been taken.

Response

Y: The attempt will be repeated.

N: The statement will be aborted.

EDT0907 NO PROCEDURE FILES DECLARED

Meaning

A @DROP ALL statement has been issued by the user but no procedure file has been declared.

EDT0909 AUTOSAVE ABORTED. ERASE SAVING FILES? REPLY(Y=YES; N=NO)
(B) Routing code: * Weight: 99

Meaning

The writing of the backup files could not be performed. The autosave function had to be aborted.

Possible reasons: no virtual address space available, a not foreseen DMS error.

Error switch: not set.

Response

Y: existing saving files will be erased.

N: existing saving files will not be erased.

EDT0910 '@RENUMBER': LINES WILL BE LOST
(B) Routing code: * Weight: 99

Meaning

A @RENUMBER statement has been entered in order to renumber the lines.

If EDT rennumbers in the asked way, the maximum line number (9999.9999) would be reached and the rest of the file would be deleted.

Response

Get information about the number of lines in the work file by entering the statement @LIMIT before asking to renumber.

EDT0911 CONTINUE PROCESSING? REPLY (Y=YES; N=NO)
(B) Routing code: * Weight: 99

Meaning

At the processing of a statement an error has been detected.

EDT inquires whether it should continue processing.

Response

Y: Processing of the statement will be continued.

N: The statement will be aborted.

EDT0914 RECORD SIZE > 256. ONLY 256 CHARACTERS WILL BE WRITTEN.

Meaning

Only 256 characters are written for each record, the rest of the records becomes undefined.

EDT0999 (&00)

Meaning

Message from external routine.

EDT1115 'COPY': NO RECORD EXISTS IN SPECIFIED RANGE

Meaning

The record range specified in the COPY statement cannot be copied, as the records do not exist.

EDT1137 SPECIFIED WORK FILE IGNORED IN CONJUNCTION WITH 'SPLIT'

Meaning

In the @PAR statement the work file variable (wkflvar) has been specified as the first operand. The actions initiated by the @PAR statement are to be performed only with regard to the specified work file, but the effect of the operand SPLIT is global.

EDT1150 NAME OF PLAM LIBRARY MEMBER TRUNCATED AFTER 64 CHARACTERS

EDT1151 VERSION OF PLAM LIBRARY MEMBER TRUNCATED AFTER 24 CHARACTERS

EDT1174 FILE ATTRIBUTES IGNORED

Meaning

By means of the @WRITE statement (format 2) an internal work file is written back to the associated external BS2000 file. File attributes cannot be defined by means of the @WRITE statement, as they have already been defined for the external file. The specified file attributes are ignored.

EDT1180 CODE ATTRIBUTE IGNORED
(B) Routing code: * Weight: 99

Meaning

By means of the XWRITE statement the actual work file is written back to the UFS file opened before by means of XOPEN. The CODE attribute was ignored, as there is already one defined for that file.

By asking to write the file with MODE=UPDATE that attribute cannot be changed.
Error switch: not set.

Response

A change of the code can be performed in the following way:
Write back the work file with MODE=REPLACE und requested CODE-Operand
and then close the file by issuing @CLOSE NOWRITE.

EDT1190 WORK FILE (&00) IS EMPTY. COPY OPERATION NOT PERFORMED

EDT1226 SPECIFIED FCBTYPED IGNORED: '(&00)' IS ASSUMED

Meaning

The FCB type specified in the @OPEN or @WRITE statement (format 2) does not match the catalog entry. The specified type is ignored and the FCBTYPED (&00) is taken over from the catalog.

EDT1227 CCS ATTRIBUTE CANNOT BE SET.
(B) Routing code: * Weight: 99

Meaning

The file has been created or updated, but the CCS attribute cannot be set. Possible reasons are i.g. that the file belongs to a foreign userid or is situated in a remote system (by use of RFA).

EDT1243 FILE '(&00)' TO BE COPIED IS EMPTY

EDT1244 FILE '(&00)' EMPTY

EDT1245 JV IS EMPTY

(B) Routing code: * Weight: 99

Meaning

An attempt has been made to get the value of a JV by a @GETJV statement, but the entire JV is empty, i.e. the length of the value string is zero.

Error switch: EDT.

EDT1253 (SOME) RECORD(S) TRUNCATED

Meaning

Records longer than 256 bytes are truncated when being read into the internal work file.

EDT1254 NO MARKS SET FOR FILE TO BE PROCESSED IN REAL MODE

Meaning

No marks can be set for files processed in real mode (i.e. that have been read in by means of @OPEN format 1).

Error switch: EDT.

EDT1901 ISAM FILE. '@GET' STATEMENT PROCESSED

Meaning

A @READ statement has been entered for an ISAM file. EDT automatically processes a @GET statement.

Error switch: EDT.

EDT1902 SAM FILE. '@READ' STATEMENT PROCESSED

Meaning

A @GET statement has been entered for a SAM file. EDT automatically processes a @READ statement.

Error switch: EDT.

EDT1903 INPUT TRUNCATED

Meaning

More than 256 characters have been read for one line.

Error switch: EDT.

EDT1904 SOME LINES > 256

Meaning

Some lines read by means of a @GET or @READ statement are longer than 256 bytes. The lines concerned are truncated after 256 characters.

Error switch: EDT.

EDT1905 INPUT TOO LONG. CORRECT INPUT

Meaning

The following conditions cause a termination error when reading:

- input for a @CREATE...READ statement >256 bytes, or
- input >284 subsequent to a @PRINT statement and input request *+-0, or
- input of a statement with indirect operands and the sum of characters of operation string and the length of the string variable exceeds 256.

Error switch: not set.

EDT1906 TOO MANY NAMES. LIST INCOMPLETE

Meaning

The 15 pages provided for FSTAT are not sufficient to accommodate all the file names, or the 8 pages provided for STAJV are not sufficient to accommodate all the names of job variables, or the 8 pages provided for CMD are not sufficient to accommodate all the lines to be output to the buffer.

The list (of names) is not complete.

Error switch: EDT.

EDT1907 MODULE CANNOT BE UNLOADED

Meaning

The module specified in the @RUN statement or in the @UNLOAD statement could not be unloaded. Either an incorrect module name has been specified or the module has already been unloaded.

Error switch: EDT.

EDT1936 MODIFIED LINE >256 CHARACTERS

Meaning

An edited line became too long as a result of modification. This error can be caused by an @ON, @PREFIX, @SUFFIX, @COL or @CREATE statement.

Moreover, an extended line containing formal operands may have become too long in a procedure. The line is truncated after 256 characters.

In a @SETJV statement, when the extended string for the value of a job variable became too long, only the first 256 characters are taken over as the value.

Error switch: EDT.

EDT2169 WORK FILE (&00) IS EMPTY. WRITE OPERATION NOT PERFORMED

Meaning

The statement @WRITE (format 2) or @XWRITE could not be performed, for the work file (&00) is empty.

Error switch: not set.

EDT2266 WORK FILE IS EMPTY: MEMBER '(&00)' CLOSED UNCHANGED

Meaning

As the work file specified in the @CLOSE or @WRITE statement (format 2) is empty, the member (&00) has been closed but not saved.

EDT2267 LINE TRUNCATED AFTER (&00) CHARACTERS

Meaning

As the modified record is longer than the LIMIT specified in the @PAR statement, it is truncated.

(&00): maximum permissible record length.

Error switch: not set.

EDT2301 COPY BUFFER OVERFLOW

Meaning

The copy buffer cannot hold more than 256 line numbers.

Error switch: not set.

EDT2900 A KEY WAS ZERO AND HAS BEEN SET TO 0.0001

Meaning

A key with the value 0 has been detected during processing of a @GET or @READ statement (with the KEY function). The key is set to 0.0001 by EDT.

Error switch: EDT.

EDT2901 DATA LOSS DUE TO TABULATOR FUNCTION. CHECK LINE LENGTH

Meaning

Some text is lost due to tabulator definition.

Error switch: EDT.

EDT2902 CHECK TAB COLUMNS

Meaning

The CHECK function was specified in the @TABS statement. CHECK detected that the line which has just been entered with tabs causes reverse positioning, i.e. text was overwritten.

Error switch: EDT.

Response

Check the line, as it probably contains an error.

EDT2903 FILE IS EMPTY

Meaning

The file specified in the statement is empty. This message is displayed under one of the following conditions:

- an empty file on disk is accessed by means of a @READ, @GET, @INPUT, or @ELIM statement
- the work file is empty and a @SAVE, @WRITE, @XWRITE or @SETLIST statement has been specified or a procedure file specified in a @COMPARE statement is empty.

Error switch: EDT.

EDT2904 MAXIMUM LINE NUMBER WHEN PROCESSING '@RENUMBER'. SOME LINES ARE LOST

Meaning

The maximum permissible line number (9999.9999) has been reached during processing of a @RENUMBER statement. EDT does not permit duplicate line numbers in a work file. The rest of the file has been erased.

Error switch: EDT.

EDT3002 OPERAND ERROR

Meaning

EDT reports that the statement contains either an invalid operand or a syntax error.

Response

Correct and re-enter the statement.

EDT3003 '(' MISSING

Response

Insert the missing bracket and re-enter the statement.

EDT3004 ') ' MISSING

Response

Insert the missing bracket and re-enter the statement.

EDT3040 INVALID NAME OR NAME MISSING
(B) Routing code: * Weight: 99

Meaning

The string contains more than 8 characters or does not fulfill the syntax of the operand or is missing.

Error switch: EDT.

EDT3050 INVALID SYSLST=NUMMER
(B) Routing code: * Weight: 99

Meaning

The SYSLST number issued in the statement @LOG is invalid.
Only values between 1 and 99 are valid.
Error switch: EDT.

EDT3065 NUMBER OF LINES OR 'OFF' OR 'O' EXPECTED

Meaning

The operand SPLIT of the @PAR statement must contain

- either the number of lines of the second window and the name of the work file to be displayed in the second window, or
- OFF or O in order to set the screen back to one window.

Response

Correct and re-enter the statement.

EDT3066 WRONG COLUMN NUMBER

Meaning

A wrong column number has been specified in the @SETF statement.
The statement has not been processed.

Response

Correct and re-enter the statement.

EDT3067 UPPER RANGE LIMIT INVALID OR MISSING

Meaning

The upper range limit required in a COPY or DELETE statement has either not been specified at all, or it is invalid. The statement has not been processed.

Response

Correct and re-enter the statement.

EDT3068 POSITION INVALID OR MISSING

Meaning

The specification of POSITION in the @SETF statement is mandatory.
The statement could not be processed.

Response

Correct and re-enter the statement.

EDT3069 STRING TO BE INSERTED IS MISSING

Meaning

The statement has not been processed because the string to be inserted is missing.

Response

Correct and re-enter the statement.

EDT3070 'EDIT-LONG' EXPECTED

Meaning

The operand specified in the @PAR statement is wrong.

Correct form of the operand: @PAR EDIT-LONG =

Response

Correct and re-enter the statement.

EDT3071 'ON', 'OFF' OR 'O' EXPECTED

Meaning

EDT expects the ON, OFF, or O operand in a given statement.

Possible error cause:

- one of the @PAR statement operands contains an error; ON, OFF or O is missing after an equals sign
- an attempt has been made to define more than 8 positions in the @TABS statement, but only ON, OFF or O is permitted here.

Response

Correct and re-enter the statement.

EDT3072 NUMBER INVALID OR MISSING

Meaning

The format of the specified numeric value is incorrect, or the value has not been specified at all. The statement has not been processed.

Response

Correct and re-enter the statement.

EDT3073 TARGET POSITION IS INVALID OR MISSING

Meaning

The statement has not been processed because the target position in the COPY or INSERT statement is missing or invalid.

Response

Correct and re-enter the statement.

EDT3074 'KEEP' OPERAND EXPECTED IN 'COPY' STATEMENT

Meaning

The COPY statement has not been processed because the operand KEEP is missing.

Response

Correct and re-enter the statement.

EDT3075 RECORD RANGE CANNOT BE SPECIFIED

Meaning

The statement has not been processed because no record range can be specified in the statement.

Response

Correct and re-enter the statement.

EDT3076 'COPY KEEP' PERMISSIBLE ONLY FOR ISAM FILES

Meaning

The KEEP operand in the COPY statement can only be specified for ISAM files. The statement has not been processed.

EDT3077 OPERAND 'STRUCTURE=' INCORRECT

Meaning

In the @PAR statement the symbol for STRUCTURE is either missing or not given in single quotes.

Response

Correct and re-enter the statement.

EDT3078 SPECIFIED NUMBER INVALID (VALID RANGE: 1..256)

Meaning

The specified value for LIMIT in the @PAR statement or the factor n in the repetition statement # is not within the permissible range.

Response

Correct and re-enter the statement.

EDT3079 COLUMN '0' NOT PERMISSIBLE

Meaning

The statement has not been processed because '0' cannot be specified as a column number.

Response

Correct and re-enter the statement.

EDT3080 SPECIFIED COLUMN INVALID, OR ':' MISSING IN COLUMN RANGE

Meaning

The statement has not been processed because the character ':' is missing in the specified column range, or the specified range is invalid.

Response

Correct and re-enter the statement.

EDT3081 LINE NUMBER > 9999.9999

Meaning

The specified line number is too high. The maximum permissible line number is 9999.9999.

Response

Correct and re-enter the statement.

EDT3082 LINE NUMBER 0 INVALID

Meaning

The statement has not been processed, as 0 cannot be specified as a line number.

Response

Correct and re-enter the statement.

EDT3085 '(&00)' NOT POSSIBLE FOR PLAM ELEMENT TYPE '(&01)'

Meaning

PLAM library elements of the type (&01) cannot be used with statement (&00).
e.g. (&01): R, C, H, L, U, F or equal free typename
with (&00): @COPY, @OPEN, @WRITE or @INPUT statement (all format 2).

EDT3086 INVALID PLAM TYPE

Meaning

The PLAM type specified in the statement is invalid.
Valid PLAM types: S, M, J, P, D, X, R, C, H, L, U, F and equal free typenames. A free typename must not start with \$ or SYS and consists of 2 to 8 characters.

EDT3087 INVALID JOB VARIABLE NAME

Meaning

The string used to specify a job variable name is incompatible with the syntax of a job variable name, or the job variable name in a @SETJV or @GETJV statement was not fully qualified, or it was an invalid request by an @ERAJV statement.

Response

Correct and re-enter statement.

EDT3088 INVALID NAME OF S-VARIABLE
(B) Routing code: * Weight: 99

Meaning

The string used to specify a SDF-P variable in a @GETVAR or @SETVAR statement is incompatible with the syntax of a SDF-P variable.
Error switch: EDT.

Response

Correct and re-enter the statement.

EDT3089 INVALID NAME OF UFS FILE
(B) Routing code: * Weight: 99

Meaning

A string used for specifying the name of a UFS file did not follow the syntax of a file name in the POSIX file system or one of the directories issued, does not exist.
Error switch: EDT.

EDT3093 INVALID STRUCTURED NAME OR STRUCTURED NAME MISSING

Meaning

The string contains more than 30 characters or does not fulfil the syntax of the operand or is missing.
Error switch: EDT.

EDT3101 INVALID STATEMENT

Meaning

The first character in the statement is invalid. The statement has not been processed.

EDT3106 SPECIFIED WORK FILE INVALID. (VALID RANGE: 0..9)

Response

Correct and re-enter the statement.

EDT3110 EQUATION MARK EXPECTED. STATEMENT NOT PROCESSED

Response

Correct and re-enter the statement.

EDT3111 'TYPE' OPERAND IS MISSING

Meaning

In the OPEN or WRITE statement, the TYPE operand has been specified without a valid operand value.

Response

Correct and re-enter the statement.

EDT3112 'TYPE' OPERAND ALREADY DEFINED

Meaning

In the OPEN or WRITE statement, an attempt was made to specify the TYPE operand a second time.

Response

Correct and re-enter the statement.

EDT3116 'CODE' OPERAND MISSING OR INVALID
(B) Routing code: * Weight: 99

Meaning

In the @XOPEN, @XWRITE or @XCOPY statement, the CODE operand has been specified without a valid operand value.

Response

Correct and re-enter the statement.

EDT3117 'MODE' OPERAND MISSING OR INVALID

Meaning

In one of the statements @OPEN (format 2), @WRITE (format 2), @XOPEN, @XWRITE or @SETVAR the MODE operand has been specified without a valid operand value.

Response

Correct and re-enter the statement.

EDT3118 'MODE' OPERAND ALREADY DEFINED

Meaning

In the OPEN or WRITE statement, an attempt was made to specify the MODE operand a second time.

Response

Correct and re-enter the statement.

EDT3119 WORK FILE ALREADY DEFINED

Meaning

In the OPEN or WRITE statement, an attempt was made to define the work file a second time.

Response

Correct and re-enter the statement.

EDT3120 FILE NAME ALREADY DEFINED

Meaning

In the OPEN or WRITE statement, an attempt was made to define the file name a second time.

Response

Correct and re-enter the statement.

EDT3121 LIBRARY NAME MISSING OR FORMAT OF SPECIFIED LIBRARY NAME INVALID

Response

Correct and re-enter the statement.

EDT3122 FILE NAME MISSING OR FORMAT OF SPECIFIED FILE NAME INVALID

Response

Correct and re-enter the statement.

EDT3123 NO VALID NAME OF PLAM MEMBER

Meaning

When processing a PLAM library, no valid member name has been specified.

Response

Correct and re-enter the statement.

EDT3124 VERSION NUMBER MISSING OR INVALID

Meaning

The version number of a PLAM library member has not been specified or the specified version number contains invalid characters.

Response

Correct and re-enter the statement.

EDT3125 'OPEN REAL' PERMISSIBLE ONLY FOR ISAM FILES

Meaning

It is not possible to process the specified file in OPEN REAL mode because that mode is permitted only for ISAM files.

Response

Process the specified file in virtual memory.

EDT3126 FILE ATTRIBUTES CANNOT BE SPECIFIED

Meaning

It is not possible to specify file attributes in the relevant statement, therefore the statement has not been processed.

EDT3127 NAME OF WORK FILE IS INVALID OR MISSING

Meaning

The statement has not been processed because no name has been specified for the work file, or because the specified name is invalid.

Response

Correct and re-enter the statement.

EDT3128 PLAM LIBRARY NAME INVALID OR MISSING

Response

Correct and re-enter the statement.

EDT3129 NO FILE ATTRIBUTES CAN BE DEFINED FOR PLAM LIBRARIES

Meaning

When processing a PLAM library an attempt was made to define the file attribute FCBTYP=ISAM or SAM. The statement has not been processed.

Response

Correct and re-enter the statement.

EDT3132 PLAM TYPE IS MISSING OR INVALID

Meaning

The TYPE attribute of the PLAM member has not been specified in the statement, or the specified attribute is invalid.

Response

Correct and re-enter the statement.

EDT3133 NUMBER OF PLAM VERSION INVALID

Response

Correct the version number and re-enter the statement.

EDT3134 '*STD' EXPECTED

Meaning

When processing a PLAM library, '*' has been specified for the type or the version.

Response

Replace '*' by '*STD' and re-enter the statement.

EDT3135 MODUL NAME MISSING

Meaning

The @UNLOAD statement has not been processed for the name of the modul has not been specified.

Response

Correct and re-enter the statement.

EDT3136 'INCREMENT=0' NOT PERMISSIBLE

Meaning

The specification INCREMENT=0 in the @PAR statement is not permitted. The statement has not been processed.

Response

Correct and re-enter the statement.

EDT3138 ONLY ONE CHARACTER POSSIBLE AS SYMBOL

Meaning

In a statement more than one character has been specified as a symbol:

- for the separator or structure symbol in @PAR, or
- for the ASTERISK, SLASH or FILLER symbol in @SYMBOLS.

Response

Correct and re-enter the statement.

EDT3170 SYNTAX ERROR IN LINE NUMBER
(B) Routing code: * Weight: 99

Meaning

The operand which is supposed to be a line number is syntactically incorrect.

Response

Correct and re-enter the statement.

EDT3171 NO EDT V15 OR EDT V16.0 STATEMENTS IN 'CONTROL' MODE

Meaning

It is not possible to process EDT statements for V15 or V16.0 in control mode.

EDT3172 MODULE NAME TOO LONG

Meaning

The modul name specified in an @UNLOAD statement was longer than 8 characters. Error switch: EDT.

EDT3173 NUMBER OF WORK FILE FOR COMPARE OPERATION MISSING OR INVALID

Meaning

Error switch: EDT.

EDT3174 NAME TOO LONG

Meaning

A string used for specifying a file or jobvariable name consists of more than 54 characters. Error switch: EDT.

EDT3175 SYNTAX ERROR IN SPECIFIED RANGE

EDT3176 STATEMENT SYMBOL INVALID OR TOO LONG

Meaning

The statement symbol in the @USE statement must be specified within single quotes and must consist of exactly one character.

Response

Correct and re-enter the statement.

EDT3177 ENTRY NAME TOO LONG

Meaning

The entry name is longer than the permissible maximum of 8 characters.
Error switch: EDT.

Response

Correct and re-enter the statement.

EDT3178 LIBRARY NAME TOO LONG

Meaning

The library name is longer than the permissible maximum of 54 characters.

Response

Correct and re-enter the statement.

EDT3179 ENTRY NAME MISSING

Meaning

If external statement routines are used as statement filters in the @USE statement, a constant entry name must be specified.

Response

Correct and re-enter the statement.

EDT3180 JOKER SYMBOL EQUALS QUOTE

Meaning

Possible reasons:

- The value specified in the @SYMBOLS statement for the ASTERISK or SLASH character is invalid as it is the same as one of the QUOTE characters.
- An @ON statement with keyword PATTERN could not be processed, as QUOTE1 or QUOTE2 is the same as the ASTERISK or SLASH character.

Error switch: EDT.

Response

Choose different symbols for ASTERISK, SLASH, QUOTE1 and QUOTE2.

EDT3181 BOTH JOKER SYMBOLS ARE THE SAME

Meaning

An attempt was made to redefine one of the joker symbols by means of a @SYMBOLS statement. The statement was not processed because different symbols must be defined for ASTERISK and SLASH.

Error switch: EDT.

Response

Choose different symbols for ASTERISK and SLASH, and re-enter the @SYMBOLS statement.

EDT3182 CCSN TOO LONG

Meaning

A string used for specifying a coded character set name consists of more than 8 characters.

Error switch: EDT.

Response

Correct and re-enter statement.

EDT3183 LINE NUMBER EXPECTED

Meaning

A valid line number has to be specified after the keyword TO in statement @FSTAT, @STAJV,...

EDT3901 ILLEGAL BINARY CONSTANT

Meaning

A string with a 'B' in front of the first single quote is errored.

Only the digits '0' and '1' are valid characters, and the string must not be empty.

Error switch: EDT.

EDT3902 ILLEGAL HEX CONSTANT

Meaning

A string with an 'X' in front of the first single quote is errored. Only the digits 0 to 9 and the letters A to F are valid characters, and the string must not be empty.

Error switch: EDT.

EDT3903 INVALID RANGE

Meaning

Either the line numbers in the specified range are invalid or a dash (-) is not followed by a second line number.

Error switch: EDT.

EDT3904 INVALID SUBSTRING

Meaning

A @SET statement contains an invalid substring. All substrings must comply with the syntax ln or +/-int.

Error switch: EDT.

EDT3905 INVALID VARIABLE

Meaning

A line number, string or integer variable has been specified incorrectly.

Error switch: EDT.

EDT3906 LINE NUMBER INVALID

Meaning

The value of a line number is invalid with regard to either

- the integer in the statement @SET ln-var=int-var, or
- the first line number variable in the statement @SET ln-var,cl=... , or
- the destination of the output in the statement @GETJV.

Moreover, a line number can be too high for the specified KEYLEN of a file opened by means of an @OPEN statement.

Error switch: EDT.

EDT3907 EMPTY STRING NOT PERMISSIBLE

Meaning

A string specified direct or indirect (i.g. by means of a EDT string variable or a SDF-P variable) is empty. But that is not permissible for this statement.

Error switch: EDT.

EDT3908 STRING MISSING OR INVALID

Meaning

A string is missing in a statement or is invalid. The two most common error causes are:

- a string is missing in a statement containing a random file name (file) and no @FILE statement is in effect
- a string requiring two single quotes contains only one.

Error switch: EDT.

EDT3909 @PARAMETER ERROR

Meaning

Some of the most common error causes are:

- the line number or increment is invalid
- one or more operands are missing in the statement
- invalid ON/OFF
- the number of a procedure file is '0'
- the value in @SETSW statement is greater than 31.

Error switch: EDT.

Response

Correct and re-enter the statement.

EDT3910 DUPLICATE FORMAL OPERAND
(B) Routing code: * Weight: 99

Meaning

A formal operand (&id) has been specified at least twice in the @PARAMS statement.

Error switch: EDT.

EDT3911 DUPLICATE KEYWORD
(B) Routing code: * Weight: 99

Meaning

A keyword has been specified at least twice in a @DO statement.

Error switch: EDT.

EDT3922 INVALID COLUMN (RANGE)
(B) Routing code: * Weight: 99

Meaning

The value specified for a column is invalid, or the specified column (range) is syntactically incorrect.

Error switch: EDT.

EDT3951 PROCEDURE NUMBER > 22

Meaning

Error switch: EDT.

EDT3952 INVALID SYMBOL
 (B) Routing code: * Weight: 99

Meaning

In some statements a special character has to be chosen for a symbol:

- in defining a new statement symbol ("@"), or
- for the loop symbol in @DO, or
- for the range symbol in @RANGE, or
- for the first operand ("single quote") in @QUOTE, or
- for the joker symbols in @SYMBOLS.

Response

Choose a valid special character as symbol.

EDT3991 SYNTAX ERROR IN EXTERNAL STATEMENT

Meaning

The external routine reports a syntax error in the specified statement.

Response

Correct and re-enter the statement.

EDT3999 (&00)

Meaning

Syntax error in an external statement.

(&00): Message returned by the external routine.

Response

Correct and re-enter the statement.

EDT4100 (&00)

Meaning

EDT message of Version 16.0.

For more detailed information see the "EDT" manual.

EDT4200 '(&00)': DMS ERROR CODE: '(&01)'

Meaning

All DMS errors are output in this form:

(&00): DMS macro (OPEN, etc.) during whose processing the error occurred.

(&01): hexadecimal error code.

For more detailed information about the DMS error enter the ISP command /
 HELP DMS(&01) or the SDF command /HELP-MESS DMS(&01) in system mode,
 or see the BS2000 manual "System Messages" or one of the BS2000 DMS
 manuals.

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4201 '(&00)': JVS ERROR CODE: '(&01)'

Meaning

All JVS errors are output in this form:

(&00): JVS macro (STAJV, etc.) during whose processing the error occurred

(&01): Hexadecimal error code.

For more detailed information about the JVS error enter the ISP command /
HELP JVS(&01) or the SDF command /HELP-MESS JVS(&01) in system mode,
or see the BS2000 manual "System Messages" or the BS2000 JVS manual.

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4202 '(&00)': SDF-P ERROR CODE: '(&01)'

Meaning

All SDF-P errors are output in this form:

(&00): SDF-P macro (PUTVAR, etc.) during whose processing the error occurred.

(&01): Hexadecimal error code.

For more detailed information about the SDF-P error enter the ISP command /
HELP SDP(&01) or the SDF command /HELP-MESS SDP(&01) in system mode,
or see the BS2000 manual "System Messages" or the BS2000 SDF-P manual.

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4203 '(&00)': XHCS ERROR CODE: '(&01)'

Meaning

All XHCS errors are output in this form:

(&00): XHCS macro (NLSCODE, etc.) during whose processing the error occurred

(&01): Hexadecimal error code.

For more detailed information about the XHCS error enter the ISP command /
HELP XHC(&01) or the SDF command /HELP-MESS XHC(&01) in system mode,
or see the BS2000 manual "System Messages" or the BS2000 XHCS manual.

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4204 '(&00)': TIAM ERROR CODE: '(&01)'

(B) Routing code: * Weight: 99

Meaning

All TIAM errors are output in this form:

(&00): TIAM-Macro (WRLST, etc.) during whose processing the error occurred

(&01): Hexadecimal error code.

For more detailed information about the TIAM error see the BS2000 manual "TIAM"
or the BS2000 manual "Macro Calls".

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4205 '(&00)': BLS ERROR CODE: '(&01)'
(B) Routing code: * Weight: 99

Meaning

All errors of the Binder Loader System are output in this form:

(&00): BLS macro (BIND) during whose processing the error occurred

(&01): Hexadecimal error code.

For more detailed information about the BLS error see the BS2000 manual "Binder Loader System" or the BS2000 manual "Macro calls".

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4206 POSIX-CALL '(&00)': ERROR '(&01)'
(B) Routing code: * Weight: 99

Meaning

All errors reported by POSIX-calls are output in this form:

(&00): Function which returns an error

(&01): error code returned in C-variable errno.

For more detailed information about the error see the BS2000 manual "C library functions" or the BS2000 manual "POSIX".

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4207 '(&00)': SDF ERROR CODE: '(&01)'
(B) Routing code: * Weight: 99

Meaning

All errors of SDF-macros are output in this form:

(&00): SDF-Macro (CMDSTA, etc.) during whose processing the error occurred

(&01): Hexadecimal error code.

For more detailed information about the SDF error see the BS2000 manual "SDF-A".

Processing of an @INPUT file is aborted due to this error.

Error switch: DMS.

EDT4300 ERROR AT SYSTEM COMMAND: ERROR CODE '(&00)'
(B) Routing code: * Weight: 99

Meaning

The command specified in the @SYSTEM statement is rejected by the CMD macro with the returncode X'10' or X'14'.

For more detailed information about the error cause enter the ISP command /HELP (&00) or the SDF command /HELP-MESS (&00) in system mode, or see the BS2000 manual "System Messages".

Error switch: DMS.

EDT4310 SDF: SYNTAX ERROR IN LINE (&00)
(B) Routing code: * Weight: 99

Meaning

While checking the syntax of data lines with @SDFTEST a syntax error has been detected in line (&00) and could not be corrected in a SDF error dialog.

Error switch: EDT.

Enable SDF guided error dialog, e.g. by @SY'/MOD-SDF-OPTION GUIDANCE=MIN'.

EDT4900 /SET-FILE-LINK IS IN EFFECT

Meaning

A /SET-FILE-LINK command with a link name used by EDT (EDTSAM, EDTISAM, or EDTMAIN) is active. However, the file name specified in the EDT statement (@GET, @READ, @INPUT, @OPEN, @ELIM, @WRITE or @SAVE) does not match the one specified in the /SET-FILE-LINK command. The statement is not processed.

Error switch: EDT.

EDT4901 ONE INPUT FILE IS ALREADY ACTIVE

Meaning

Two @INPUT statements cannot be active in EDT at the same time.

Error switch: EDT.

EDT4903 BOTH OPERANDS IN '@QUOTE' STATEMENT ARE THE SAME

Meaning

If both operands (q1 and q2) are specified in the @QUOTE statement, they must be different.

Error switch: EDT.

EDT4904 BTAM FILES NOT SUPPORTED

Meaning

An attempt was made to process a BTAM file by means of a @GET, @SAVE, @READ, @WRITE, @INPUT, @OPEN or @ELIM statement. However, EDT does not support BTAM files.

Error switch: EDT.

EDT4906 '(&00)' NOT POSSIBLE FOR CURRENT PROCEDURE FILE

Meaning

The statement (&00) refers to the current procedure file and therefore is impossible to be executed (i.g. @DO, @DROP).

Error switch: EDT.

EDT4907 '@DROP' NOT POSSIBLE DURING PROCEDURE FILE PROCESSING
(B) Routing code: * Weight: 99

Meaning

Error switch: EDT.

EDT4908 INVALID COMMAND

Meaning

The command specified in the @SYSTEM statement either contains an error or cannot be passed by the CMD macro.

Error switch: DMS.

Response

Correct and re-enter the statement or branch to system mode by means of @SY.

EDT4909 PROCEDURE FILE ALREADY ACTIVE

Meaning

A @PROC statement has been specified for the current procedure file.

Error switch: EDT.

EDT4910 S-VARIABLE MUST BE OF TYPE LIST

(B) Routing code: * Weight: 99

Meaning

The SDF-P variable specified in a @LOG, @GETLIST or @SETLIST statement is not of type LIST or has not been declared yet.

Error switch: EDT.

If the variable has not been declared yet, perform the system command /DECL-VAR NAME=...,MULT-ELEM=LIST before re-entering the statement.

EDT4912 EAM OPEN ERROR

Meaning

An EAM file cannot be opened during a @LIST statement in conjunction with the 'I' operand.

Error switch: EDT.

EDT4913 EAM WRITE ERROR

Meaning

A write error has occurred while writing to an EAM file (@LIST statement in conjunction with the 'I' operand).

Error switch: EDT.

EDT4914 EDT OR FILE FORMAT ERROR WITH INTERRUPT WEIGHT=60

Meaning

Data errors can occur when an attempt is made to access a disk file with invalid keys. When searching for the key, EDT always uses the first eight characters of the record. If this results in an invalid line number, data errors can be the consequence.

For more detailed information on the error cause see the "EDT" manual.

Error switch: EDT, DMS.

EDT4916 FILE NOT IN CATALOG

Meaning

A file name specified in a @FSTAT, @GET, @READ, @INPUT, @ELIM, @SAVE, @WRITE, @COPY, or @UNSAVE statement does not exist in the catalog.

If this error occurs in an @FSTAT statement, the DMS error switch is also set for reasons of compatibility. Processing of @INPUT files is, however, not aborted.

In new EDT procedures only the EDT error switch should be checked.

Error switch: EDT, DMS (see meaning text).

EDT4918 FORMAL OPERAND MISSING

Meaning

A formal operand (&id) has been expected but not found in a @PARAMS statement.

Error switch: EDT.

EDT4919 REQM ERROR FOR (&00) BUFFER

Meaning

During processing of a @FSTAT, @STAJV, @ERAJV, @LIST I or @SYSTEM statement using the (&00) macro, the required pages of virtual address space could not be provided by REQM for the (&00) buffer.

E.g.: for FSTAT 15 pages, for STAJV 8 pages, for EAM 1 page, for CMD 8 pages.

Error switch: EDT.

EDT4920 STATEMENT ILLEGAL DURING PROCEDURE FILE PROCESSING

Meaning

One of the following statements was to be processed while a procedure file was being processed: @INPUT, @UPDATE (format 2), @CODENAME or @SETF GLOBAL.

Error switch: EDT.

EDT4921 STATEMENT ILLEGAL DURING '@INPUT' PROCESSING

Meaning

The @UPDATE statement (format 2), @CODENAME or @GOTO has been read from a file opened by means of @INPUT.

Error switch: EDT.

EDT4923 INVALID FILE NAME

Meaning

The file name specified in a @FSTAT, @GET, @READ, @INPUT, @OPEN, @ELIM, @WRITE, @SAVE, @UNSAVE, @COPY or @DELETE statement does not comply with the conventions governing the definition of the file names.

Possible error cause: the file name specified in single quotes or in a string variable has a leading blank.

If this error occurs in the processing of @FSTAT statement the DMS error switch is also set.

Processing of @INPUT files is, however, not aborted.

In new EDT procedures only the EDT error switch should be checked.

Error switch: EDT, DMS (see 'Meaning' text).

EDT4924 INVALID FORMAL OPERAND

Meaning

A @PARAMS statement contains an invalid operand (&id).

Error switch: EDT.

EDT4925 STATEMENT ONLY PERMITTED IN WORK FILE 0
(B) Routing code: * Weight: 99

Meaning

The @OPEN (format 1) statement is not permissible in a procedure file.

A file can only be opened in real mode in work file 0.

Error switch: EDT.

EDT4926 INVALID KEY

Meaning

In the @GET '...' N, @READ '...' KEY, or @ELIM '...' statement an attempt was made to access a record using an invalid key. Processing of the statement has been aborted.

Processing of the @INPUT files has been aborted in case of DMS errors.

In batch mode or if EDT is reading in data from SYSDTA by means of RDATA, EDT terminates and the following message is displayed:

'EDT8001EDT TERMINATED ABNORMALLY'.

Error switch: EDT, DMS.

EDT4927 INVALID KEY IN FILE OPENED IN REAL MODE

Meaning

An ISAM file has been opened in real mode (@OPEN) and an attempt has been made to access a record by means of an invalid key. Processing of the statement has been aborted, and the file currently being processed has been closed. Processing of @INPUT files is aborted as in the case of DMS errors. In batch mode or if EDT is reading from SYSDTA by means of RDATA, EDT terminates and the following message is displayed:

'EDT8001 EDT TERMINATED ABNORMALLY'.

Error switch: EDT, DMS.

EDT4928 INVALID VALUE

Meaning

A @PARAMS keyword or a @DO operand has an invalid value. The most common error cause is an unpaired single quote.

Error switch: EDT.

EDT4929 ISAM 'RECORD-FORMAT=FIXED' NOT SUPPORTED

Meaning

An attempt has been made to process a file with fixed record length using the @OPEN statement and the /SET-FILE-LINK command with LINK-NAME=EDTMAIN.

Error switch: EDT.

EDT4930 'KEY-POSITION <>1' AND 'RECORD-FORMAT=FIXED' NOT SUPPORTED

Meaning

In a @GET or @SAVE statement an ISAM file has been assigned by means of a /SET-FILE-LINK ...,LINK-NAME=EDTISAM,ACCESS-METHOD=ISAM(RECORD-FORM=FIXED..) command, but the file does not have 'KEY-POSITION=1'.

Error switch: EDT.

EDT4931 KEY-LENGTH TOO BIG

Meaning

A @GET, @SAVE, @ELIM or @OPEN statement has been issued for a file with KEY-LENGTH >8.

Error switch: EDT.

EDT4932 LINE NUMBER NOT FOUND

Meaning

- A line number has been specified for a string, but the line is not in the procedure file or the file is empty. If the error occurs while an EDT procedure or an @INPUT file is being processed, this message is not displayed unless logging has been explicitly activated by means of the PRINT operand. In this case the EDT error switch is not set either.
- The range of lines specified in the @COMPARE statement is invalid. In this case the error message is always displayed, the EDT error switch is set.

Error switch: EDT (see 'Meaning' text).

EDT4933 MODULE LOADING NOT POSSIBLE

Meaning

It is not possible to load the module (e.g. IEDTCALL) by means of the @RUN or @USE statement.

Error switch: EDT.

EDT4934 MAIN FILE IS SAM

Meaning

The first file name specified in the @OPEN statement is the name of a SAM file. A copy of the SAM file can be processed in real mode by specifying '@OPEN <file1> AS <file2>'.
Error switch: EDT.

EDT4935 MAIN FILE OPENED REAL

Meaning

An attempt was made to process a statement that is not permissible as long as the main file is opened real by means of @OPEN (format 1) (e.g. @RENUMBER).
Error switch: EDT.

EDT4936 'KEY-POSITION <>5' AND 'RECORD-FORMAT=VARIABLE' NOT SUPPORTED
(B) Routing code: * Weight: 99

Meaning

It is not possible to process a file with that catalog properties by means of @OPEN (Format 2).
Error switch: EDT.

EDT4937 NO MORE SPACE FOR OPERAND VALUES

Meaning

A current operand value or a formal keyword value consisting of n characters uses (n+1) bytes of a virtual memory page reserved for procedure file arguments (values). With the exception of empty operands, this message is displayed if a value causes more than 4096 bytes to be used.

For more detailed information on the error see the "EDT" manual.
Error switch: EDT.

EDT4938 NO MORE SPACE FOR OPERANDS

Meaning

A formal operand of length n (including the '&' character) is using (n+4) bytes on a page of the virtual memory reserved for formal operands of procedure files. If an operand causes more than 4096 bytes to be used, this message is displayed.

The page for formal operands will not be allocated if no operands are used. It will be returned after all procedure files have been dropped by means of the @DROP statement, or if no more @INPUT files are active.

For more detailed information on the error see the "EDT" manual.
Error switch: EDT.

EDT4939 '@END' WITHOUT '@PROC' STATEMENT

Meaning

An @END statement has been specified but there is no procedure file which has to be ended, that means that the current work file is work file 0.
Error switch: EDT.

EDT4940 POSITION VALUES NOT ASCENDING

Meaning

The values in a @TABS statement to define the positions of the hardware tabulators must be in ascending order.

Error switch: EDT.

Response

Correct and re-enter the statement.

EDT4941 NO POSITIONS DEFINED

Meaning

Tabulators cannot be used until the positions have been defined.

Error switch: EDT.

Response

Define the positions using the @TABS statement.

EDT4942 STATEMENT ONLY POSSIBLE IN PROCEDURE FILE

Meaning

A @RETURN, @GOTO or @IF statement can only be executed while a procedure file is being processed.

Error switch: EDT.

EDT4943 CHANGE OF CCS NOT POSSIBLE – WORK FILES NOT EMPTY
(B) Routing code: * Weight: 99

Meaning

A change of the coded character set is only possible, if all work files are empty. Either a @CODENAME statement was issued, or a file with a CCS name different from the actual was to be input or opened (@READ, @OPEN,..).

Error switch: EDT.

Re-enter statement after closing opened files and deleting work files.

EDT4944 @PARAMS STATEMENT MISSING

Meaning

The @DO statement contains operands, but there is no @PARAMS statement in the procedure file, or it is not the first statement in the procedure file.

Error switch: EDT.

EDT4945 NOT POSSIBLE ON THIS TERMINAL

Meaning

The @UPDATE statement in format 2 has been specified on a printer, or an attempt was made to change the screen dimension with a @VDT statement for a terminal other than a 9763.

Error switch: EDT.

EDT4946 OVERFLOW ERROR

Meaning

The result of an arithmetic operation using a @SET statement (format 1) exceeds the highest positive or negative value of an integer variable ($2^{31}-1, -2^{31}$).

Error switch: EDT.

EDT4947 PAM FILE NOT SUPPORTED

Meaning

An attempt was made to process a PAM file by means of a @GET, @READ, @INPUT, @OPEN, @ELIM, @SAVE or @WRITE statement. PAM files are not supported by EDT.

Error switch: EDT.

EDT4948 POSITIONAL OPERAND AFTER KEYWORD OPERAND

Meaning

In a @DO statement a positional operand has been specified after a keyword operand.

Error switch: EDT.

EDT4949 PROCEDURE FILE IS EMPTY

Meaning

An EDT procedure started by means of a @DO statement is empty. The procedure file has been defined by means of a @PROC and an @END statement, but it contains neither data records nor EDT statements.

Error switch: EDT.

EDT4950 PROCEDURE FILE IS UNDEFINED

Meaning

In a @DO or @COMPARE statement a procedure file has been specified that has not been defined in a @PROC statement.

Error switch: EDT.

EDT4951 WORK FILE IS EMPTY. STATEMENT NOT PROCESSED

Meaning

The statement refers to a line number which cannot be found as the work file is empty.

EDT4954 REQM ERROR. PLEASE RECEIPT WITH "Y"

Meaning

The attempt by EDT to allocate additional memory is rejected with a return code, or the ENTRLINE routine has been called by an EDT subroutine (@RUN), but no virtual memory is available.

Error switch: not set.

Response

Y: EDT returns to the next free line number.

Else: The message will be repeated.

If this message is output during processing of an EDT procedure, the processing can be ended by issuing K2 and /INTR.

EDT4955 PROCEDURE FILE(S) NOT YET TERMINATED

Meaning

A @DROP statement is not permitted while procedure files are stored in the procedure stack, i.e. another @PROC statement has been specified before a preceding procedure was terminated.

Error switch: EDT.

EDT4956 SYSDTA EOF

Meaning

EDT issued a read instruction, but an end-of-file condition occurred.

If 'EOF' is reported by RDATA (@EDIT ONLY mode), the EDT will switch to WRTRD (@EDIT mode). If 'EOF' is reported by WRTRD or in batch mode, EDT will issue a BKPT. Then the EOF condition can be reset, and processing can be continued by means of the /RESUME-PROGRAM command.

Error switch: EDT.

EDT4957 SYSDTA NOT ASSIGNED OR READ ERROR

Meaning

The RDATA macro supplied the return code X'14' or X'18'. The EDT run will be aborted and the message 'EDT8001EDT TERMINATED ABNORMALLY' displayed.

Error switch: EDT.

EDT4958 @SYSTEM STATEMENT INCORRECT

Meaning

The command specified in the @SYSTEM statement contains an invalid operand or returned an DMS error. It is rejected by the CMD macro with return code X'10'.

Error switch: DMS.

EDT4959 PROCEDURE FILE ALREADY ACTIVE

Meaning

A @PROC statement for a procedure file already activated by a @DO statement is not permissible.

Error switch: EDT.

EDT4960 TIAM MACRO ERROR

Meaning

One of the macros WROUT, WRTRD, RDATA or MSG7 reported an error with return code X'04' or X'08'.

EDT terminates with the message 'EDT800 EDT TERMINATED ABNORMALLY'.

In the case of the return code X'08' an area dump is output additionally.

Error switch: not set.

EDT4961 TOO MANY PROCEDURE FILES ACTIVE

Meaning

Error in the @DO statement: more than 22 procedure files are being processed at the same time.

Error switch: EDT.

EDT4962 TOO MANY FILES

Meaning

No more memory is available for nested (not closed by @END) definitions of procedure files or INPUT files.

Error switch: EDT.

EDT4963 TOO MANY OPERANDS

Meaning

There are more current operands in the @DO statement than formal operands in the @PARAMS statement.

Error switch: EDT.

EDT4964 TOO MANY POP OPERATIONS

Meaning

A @ statement has been specified to pop in a three-stage procedure stack.

This means that there are more pop than push operations, or that there are never three push operations in the stack. (Control returns to the start only if the range is full.)

Error switch: EDT.

EDT4965 TOO MANY POSITIONAL OPERANDS

Meaning

There are more positional operands in a @DO statement than have been specified in a @PARAMS statement.

Error switch: EDT.

EDT4966 'UPDATE' FOR ISAM FILE NOT POSSIBLE

Meaning

A @WRITE statement with the UPDATE function has been specified for an ISAM file.
Error switch: EDT.

EDT4967 'UPDATE' FOR SAM FILE NOT POSSIBLE

Meaning

A @SAVE statement with the UPDATE function has been specified for a SAM file.
Error switch: EDT.

EDT4968 WORK FILE NOT EMPTY

Meaning

There were still some lines in the work file when an @OPEN statement was specified.
@OPEN is only permitted if the work file is empty.
Error switch: EDT.

EDT4969 WRONG VERSION: (&00) (&01)

Meaning

A file name has been specified in a statement with a wrong version number. In this message EDT displays the correct version number of the file. If the file is only to be read, the statement will be processed. In the case of write access, the statement will not be processed. If a statement containing a wrong version number (write and read access) is read from an @INPUT file, the procedure will be aborted.
Error switch: DMS.

EDT4971 FIRST FILE EMPTY OR NOT CATALOGED

Meaning

An AS file has been specified in an @OPEN statement but the first file is either empty or not cataloged.
Error switch: EDT.

EDT4972 @ELIM STATEMENT FOR SAM FILE ILLEGAL

Meaning

Error switch: EDT.

EDT4973 @UPDATE STATEMENT IN BINARY MODE NOT POSSIBLE

Meaning

After activation of binary mode by means of the @INPUT statement, an @UPDATE statement (format 2) has been specified for corrections.
Error switch: EDT.

EDT4974 LINE NOT IN PROCEDURE FILE

Meaning

The line number specified in a @GOTO statement does not exist in the procedure file.
Error switch: EDT.

EDT4975 BIND NOT SUCCESSFUL

Meaning

In the specified module library, DLL could not find any module with the ENTRY or CSECT name specified in a @RUN statement.
Error switch: EDT.

EDT4976 STATEMENT INHIBITED FOR USER

Meaning

A statement was given by the user (@RUN, @LOAD, etc.) which is not permitted by the calling program.

EDT4977 'RECORD-FORMAT=UNDEFINED' ILLEGAL

EDT4978 INVALID IN F-MODE

EDT4980 ILLEGAL OR UNKNOWN CCS NAME

Meaning

The CCSN specified in a @CODENAME statement or the CCSN of a file or library element to be read (@READ, @OPEN, @COPY, @INPUT) is illegal or unknown.
Error switch: EDT.

EDT4981 RECORD-SIZE > 256. FILE NOT WRITTEN

EDT4982 REQUESTED JV NOT CATALOGED

Meaning

The name of the job variable specified in a @GETJV, @STAJV, or @ERAJV statement has not been found in the catalog.
Error switch: EDT, DMS.

EDT5065 INVALID RANGE: LOWER LIMIT > UPPER LIMIT

Meaning

The first line number specified in the range is higher than the second one.

Response

Correct and re-enter the statement.

EDT5078 RECORD WITH SAME KEY EXISTS: 'INSERT' NOT POSSIBLE

Meaning

The key specified in the 'INSERT <...> AT' statement already exists.
It is not possible to insert a new record.
Correct the record key and re-enter the statement.

EDT5079 STRING TO BE INSERTED IS EMPTY. 'INSERT' NOT POSSIBLE

Meaning

It is not possible to insert an empty string by means of the INSERT statement.

EDT5080 OPERANDS '\$0'..' '\$9', 'FIRST', 'LAST' NOT SUPPORTED

Meaning

The specified operands <wkflvar>, FIRST (or FI) or LAST (or LA) are invalid here.

Response

Instead of @SETF FI use @SETF or @SETF %.

Instead of @SETF LA use @SETF \$.

Correct and re-enter statement.

EDT5121 'CLOSE REAL' STATEMENT VALID ONLY IN WORK FILE 0

Response

Process the file to be closed real in work file 0.

EDT5122 NO FILE NAME

Meaning

No file name has been specified, or the format of the specified file name is invalid.

Response

Enter the file name in the form F=<filename>, L=<libname>, or E=<elemname>.

EDT5123 'CLOSE REAL' NOT POSSIBLE: '(&00)' IS NOT OPENED REAL

Meaning

(&00): file.

Response

Re-enter the statement without the operand REAL.

EDT5124 'OPEN REAL' STATEMENT VALID ONLY IN WORK FILE 0

Response

Process the file to be opened real in work file 0.

EDT5125 'OPEN REAL' VALID ONLY FOR ISAM FILES

Meaning

It is not possible to process the file with OPEN REAL because the file is not an ISAM file.

Response

Process the specified file in virtual memory.

EDT5126 '(&00)' NOT POSSIBLE: WORK FILE 0 IS OPEN

Meaning

The file or library element could not be opened because an ISAM file has been opened real in work file 0 by means of the statement @OPEN (format 1).

A subsequent @OPEN (format 2) or @XOPEN is rejected.

Error switch: EDT.

Response

Enter the @CLOSE statement in order to close the file opened by means of the @OPEN statement.

EDT5170 TEXT IN HALT STATEMENT NOT PERMISSIBLE IN CONTROL MODE

EDT5171 NO EDT V15 OR EDT V16.0 STATEMENTS IN 'CONTROL' MODE

Meaning

It is not possible to process EDT statements for V15 or V16.0 in control mode.

EDT5177 NO FILE TO CLOSE

EDT5179 PLAM MEMBER MISSING. STATEMENT NOT PROCESSED

EDT5180 '@CLOSE' OR '@CLOSE NOWRITE' EXPECTED

Meaning

An attempt was made to process a file or library element by means of the @OPEN or @XOPEN statement, although another file or library element is already open in that work file. Because of the same reason a @DROP of this work file is not possible.

Response

Close the processed file or library element by means of @CLOSE or @CLOSE NOWRITE and re-enter the statement.

EDT5181 NO LIBRARY NAME DEFINED

Meaning

The statement could not be processed because the library name has not been defined.

EDT5182 'CLOSE REAL' NOT POSSIBLE FOR PLAM MEMBERS

Response

Close the opened PLAM member by means of @CLOSE or @CLOSE NOWRITE.

EDT5183 'CLOSE REAL' INVALID FOR SAM FILES

Response

Close the SAM file by means of @CLOSE or @CLOSE NOWRITE.

EDT5188 NUMBER OF LINES NOT PERMISSIBLE

Meaning

The number specified in the SPLIT operand of the @PAR statement for the number of lines in the second work window would mean that one of the two work windows would contain less than 2 lines.

EDT5189 '(&00)' NOT POSSIBLE: A FILE IS OPENED IN WORK FILE 9

Meaning

A @SHOW statement was issued but cannot be performed, for a file is opened in work file 9.

Response

Close the file opened in work file 9.

EDT5191 '(&00)' NOT POSSIBLE. WORK FILE (&01) IS NOT EMPTY

Meaning

Statement (&00) (e.g. @OPEN format 2, @XOPEN,...) can only be processed, if the work file (&01) is empty.

Error switch: EDT.

Response

Select another work file or delete the specified work file and re-enter the statement.

EDT5221 READ ERROR ((&00)): DMS ERROR CODE: '(&01)'

Meaning

The @OPEN or @COPY statement (format 2) has not been processed due to a read error of the access method (&00).

(&01): DMS error code.

For more detailed information about the DMS error enter the ISP command / HELP DMS(&01) or the SDF command /HELP-MESS DMS(&01) in system mode, or see the BS2000 manual "System Messages" or one of the BS2000 DMS manuals.

EDT5224 INVALID ACCESS-METHOD

Meaning

The file specified in a @COPY, @OPEN or @WRITE statement (format 2) cannot be processed by EDT because of the access-method.

At present, EDT can only process SAM and ISAM files.

Response

Convert the specified file into a SAM or ISAM file.

EDT5225 INVALID RECORD-FORMAT

Meaning

The RECORD-FORMAT of a file specified in a @COPY, @OPEN or @WRITE statement (format 2) cannot be processed by EDT.

At present, EDT supports only files with variable record format.

Response

Convert the specified file to a file with RECORD-FORMAT=VARIABLE.

EDT5226 '@OPEN' NOT POSSIBLE: RECORD SIZE > 256

(B) Routing code: * Weight: 99

Meaning

A file with fixed record size larger than 256 bytes cannot be handled by means of @OPEN. The contents of the records would get lost from column 257 on.

Error switch: EDT.

EDT5233 SET ERROR (ISAM): DMS ERROR CODE: '(&00)'

Meaning

The @COPY statement (format 2) has not been processed due to a SET error.

(&00): DMS error code.

For more detailed information about the DMS error enter the ISP command / HELP DMS(&00) or the SDF command /HELP-MESS DMS(&00) in system mode, or see the BS2000 manual "System Messages" or one of the BS2000 DMS manuals.

EDT5237 WRITE ERROR ((&00)): DMS ERROR CODE: '(&01)'

Meaning

The @CLOSE or @WRITE statement has not been processed due to a write error in the access method (&00).

(&01): DMS error code.

For more detailed information about the DMS error enter the ISP command / HELP DMS(&01) or the SDF command /HELP-MESS DMS(&01) in system mode, or see the BS2000 manual "System Messages" or one of the BS2000 DMS manuals.

EDT5241 FILE '(&00)' FOR COPY OPERATION DOES NOT EXIST

Meaning

The file specified in the COPY statement does not exist. The statement has not been processed.

(&00): File name.

EDT5244 'COPY' STATEMENT WITH 'KEEP' ONLY VALID FOR ISAM FILES

EDT5245 INVALID RECORD KEY

Meaning

It is not possible to read an ISAM file with an alphanumeric record key.

EDT5246 SECONDARY KEY(S) INCOMPLETLY SET

(B) Routing code: * Weight: 99

Meaning

The NKISAM file has been closed. While setting the secondary keys afterwards an error has been reported.

Error switch: EDT.

Check data of secondary keys.

EDT5250 ERROR CODE '(&00)' IN PLAM FUNCTION '(&01)'

Meaning

The PLAM function (&01) (i.g. DETACH, ATTACH,..) called when processing the statement supplied the error code (&00). The statement has not been processed.

EDT5251 ERROR CODE '(&00)' IN PLAM FUNCTION 'CLOSE'

Meaning

The PLAM function CLOSE called when processing the CLOSE statement supplied the error code (&00). The statement has not been processed.

EDT5252 MAXIMUM LINE NUMBER

Meaning

The line number 9999.9999 has been reached. When input from a file or a SDF-P variable the number of records or list elements is too high.

Error switch: EDT.

EDT5253 SPECIFIED FILE IS NOT A PLAM LIBRARY

Meaning

The file specified in the operand LIBRARY of a @OPEN, @COPY, @DELETE, @INPUT or @SHOW statement or predefined in a @PAR statement cannot be accessed by PLAM.

EDT5254 (&00) NOT IN SYSTEM

Meaning

The specified statement could not be processed because the subsystem (&00) is not available in the system.

Error switch: EDT, DMS.

EDT5255 ERROR CODE '(&00)' IN PLAM FUNCTION 'GETA'

Meaning

The PLAM function GETA called when processing the statement supplied the error code (&00). The statement has not been processed.

EDT5256 ERROR CODE '(&00)' IN PLAM FUNCTION 'ATTACH' / DMS ERROR CODE '(&01)'

Meaning

During statement processing the called PLAM function ATTACH reported the error code (&00). The statement has not been processed.

(&01): DMS error code.

For more detailed information about the DMS error enter the ISP command / HELP DMS(&01) or the SDF command /HELP-MESS DMS(&01) in system mode, or see the BS2000 manual "System Messages" or one of the BS2000 DMS manuals.

EDT5257 ERROR CODE '(&00)' IN PLAM FUNCTION 'OPEN'

Meaning

The PLAM function OPEN called when processing the OPEN statement supplied the error code (&00). The statement has not been processed.

EDT5258 FILE '(&00)' ALREADY EXISTS

Meaning

The file (&00) specified in the @OPEN statement (format 2) already exists. The statement has not been processed.

EDT5259 CCS '(&00)' INCOMPATIBLE WITH TERMINAL

Meaning

A file or library element which was to be read or opened had the catalog attribute (&00), or the CCS (&00) was asked for at a @CODENAME statement.

It is not possible to change the Coded Character Set to (&00), because the terminal cannot be set to this.

Error switch: EDT.

EDT5261 'DELETE' NOT PROCESSED. LIBRARY '(&00)' DOES NOT EXIST

EDT5263 ERROR CODE '(&00)' IN PLAM FUNCTION 'PUTA'

Meaning

The PLAM function PUTA called when processing the statement supplied the error code (&00).

The member has been closed but was not written back.

EDT5266 LIBRARY '(&00)' LOCKED

Meaning

The specified library (&00) is read-protected.
The statement has not been processed.

EDT5267 SPECIFIED LIBRARY '(&00)' DOES NOT EXIST

EDT5268 MEMBER '(&00)' IS LOCKED

Meaning

The specified member could not be accessed, as it is either protected or has already been opened.

EDT5270 MEMBER '(&00)' IN LIBRARY '(&01)' NOT FOUND FOR UPDATE OPERATION

Meaning

The member (&00) in library (&01) specified in the @OPEN statement could not be found.
The statement has not been processed.

Response

Check PLAM typ of required member.

EDT5271 S-VARIABLE NOT FOUND FOR UPDATE

(B) Routing code: * Weight: 99

Meaning

A @SETVAR statement was issued with the operand MODE=UPDATE, but the specified variable has not been defined.

Error switch: EDT.

EDT5272 S-VARIABLE ALREADY DECLARED

(B) Routing code: * Weight: 99

Meaning

A @SETVAR statement was issued with the operand MODE=NEW, but the specified variable has already been defined.

Error switch: EDT.

EDT5273 MEMBER '(&00)' IN LIBRARY '(&01)' ALREADY EXISTS

Meaning

The member (&00) in the library (&01) specified in the @OPEN statement with operand MODE=NEW already exists. The statement has not been processed.

Error switch: EDT.

EDT5274 S-VARIABLE NOT DECLARED
(B) Routing code: * Weight: 99

Meaning

A @GETVAR, @GETLIST or @SETLIST statement could not be processed, because the specified variable has not been defined yet.
Error switch: EDT.

EDT5275 'COPY' NOT POSSIBLE: MEMBER '(&00)' DOES NOT EXIST

Meaning

The @COPY statement has been not processed because the member specified in the statement does not exist.

Response

Check PLAM typ of required member.

EDT5278 FILE '(&00)' PROTECTED BY PASSWORD

Response

Contact the file owner.

EDT5279 FILE '(&00)' LOCKED

Meaning

The specified file is either read-protected or has already been opened.

EDT5281 FILE '(&00)' DOES NOT EXIST

EDT5282 FILE '(&00)' IS EMPTY OR LOCKED

Meaning

The specified file has last-page pointer 0.

Possible reasons are:

- The file is empty.
- The file is assigned to SYSLST or SYSOUT.

Response

Re-enter statement later.

EDT5283 ERROR CODE '(&00)' IN PLAM FUNCTION 'DELETE'

Meaning

When attempting to delete a PLAM member the PLAM DELETE macro issued error code (&00).

EDT5284 MEMBER '(&00)' DOES NOT EXIST

Meaning

The @DELETE statement for member (&00) could not be processed because the specified member does not exist.

Response

Re-enter the statement with the correct member name and PLAM typ.

EDT5285 'SHOW': PLAM ERROR CODE '(&00)'

Meaning

When processing the @SHOW statement, a PLAM macro issued the error code (&00).

EDT5286 INVALID USER TYPE

(B) Routing code: * Weight: 99

Meaning

The requested library element is not editable. The specified user type is not equivalent to one of the following PLAM types: S,M,P,J,D,X.

Error switch: EDT.

EDT5287 NO MEMBERS OF SPECIFIED TYPE OR LIBRARY IS EMPTY

Meaning

The @SHOW statement has not been processed because no members exist of the specified type, or the library is empty.

EDT5289 JV LINK NAME NOT DEFINED

Meaning

An attempt was made to access a JV by calling it by link name, but a job variable with this link name is not defined.

Error switch: EDT, DMS.

EDT5290 BUFFER TOO SMALL

Meaning

EDT has readied a buffer for the output of a system macro:

e.g. for FSTAT 15 pages and for STAJV 8 pages of virtual memory space.

However, this buffer was too small to hold the complete output, with the result that processing of the macro was rejected with an appropriate return code.

Error switch: EDT.

Response

Reduce the scope of the output by issuing the statement (@FSTAT, @STAJV or @ERAJV) with a partially qualified file or job variable name, or in case of @SDFTEST, change the options for SDF.

EDT5291 SYSDTA EOF

Meaning

When attempting to read the next statement from SYSDTA the end of the file (EOF) has been reached.

Response

Terminate EDT normally by means of the HALT statement.

EDT5293 REQM ERROR

Meaning

No virtual memory is available for file processing.

EDT5294 RELM ERROR

Meaning

An error has occurred while releasing virtual memory.

EDT5300 INTERNAL EDT ERROR '(&00)'

Meaning

Internal EDT runtime error.

(&00): Error code.

Response

Contact the system support service.

EDT5310 UFS FILE '(&00)' DOES NOT EXIST

(B) Routing code: * Weight: 99

Meaning

The specified file (&00) of the UFS-filesystem cannot be dealt with, for it does not exist.

Error switch: EDT.

EDT5311 UFS FILE ALREADY EXISTS

(B) Routing code: * Weight: 99

Meaning

The file specified in @XOPEN or @XWRITE cannot be handled with the operand MODE=NEW, for it already exists in the UFS-file system.

Error switch: EDT.

EDT5312 INVALID ACCESS TO UFS-FILE

(B) Routing code: * Weight: 99

Meaning

The file whose name was specified in a @XOPEN, @XCOPY or @XWRITE statement could not be opened for reading for the access was denied.

Error switch: EDT.

EDT5313 UNABLE TO CREATE UFS FILE '(&00)'

Meaning

The UFS file &00 could not be opened with MODE=NEW, for a directory is missing.
Error switch: EDT.

EDT5320 SDF: NO PROGRAM NAME FOR TEST OF STATEMENTS DEFINED
(B) Routing code: * Weight: 99

Meaning

The user issued @SDFTEST PROGRAM or marked a line starting with '/' with t, but has not defined an internal program name yet.
Error switch: EDT.

Response

Issue @SDFTEST with operand PROGRAM=name or define an internal program name with @PAR SDF-PROGRAM=name.

EDT5321 SDF: PROGRAM NAME UNKNOWN
(B) Routing code: * Weight: 99

Meaning

The internal program name which was to be used in the statement @SDFTEST PROGRAM is not known in any active syntax file.
Error switch: EDT.

EDT5322 SDF: TEST OPERATION ABORTED
(B) Routing code: * Weight: 99

Meaning

Processing of @SDFTEST or the statement t has been aborted.
Possible reason: more than 255 continuation lines.
Error switch: EDT.

EDT5323 SDF: EXTERNAL PROGRAM NAME NOT SUPPORTED

Meaning

Specification of external program name in @SDFTEST or PAR SDF-PROGRAM statement is not supported with actual SDF version.
Error switch: EDT

Response

Use internal program name instead.

EDT5340 S-VARIABLE EMPTY
(B) Routing code: * Weight: 99

Meaning

The variable specified in a @GETVAR statement is declared but has not set to any value.

EDT5341 S-VARIABLE LONGER THAN 256 CHARACTERS
(B) Routing code: * Weight: 99

Meaning

The @GETVAR statement could not be performed, for the value of the specified variable is string longer than 256 characters.
Error switch: EDT.

EDT5342 WRONG TYPE OF S-VARIABLE
(B) Routing code: * Weight: 99

Meaning

The @GETVAR or @SETVAR statement could not be processed, for the type of the specified variable does not match the operand on the right side of the equation mark.
Error switch: EDT.
A SDF-P variable of type INTEGER can only be put to an integer variable and vice versa.

EDT5343 WRONG TYPE OF LIST ELEMENT
(B) Routing code: * Weight: 99

Meaning

The statement @GETLIST or @SETLIST could not be processed for the elements of the specified list variable are not of type STRING.
Error switch: EDT.

EDT5350 COMPARE RESULT CANNOT BE SHOWN

Meaning

The output file is one of the work files to be compared.
Error switch: EDT.

EDT5351 COMPARE OPERATION ABORTED

Meaning

While processing the @COMPARE statement (format 2), an unrecoverable error occurred causing the compare operation to be aborted.
Error switch: not set.

EDT5352 COMPARE OPERATION ABORTED, RENUMBER

Meaning

Processing of the @COMPARE statement (format 2) has been aborted. The last byte of a line number used internally during comparison is not '0'. The work files must be renumbered before the compare operation can be tried again.
Error switch: EDT.

EDT5353 UNRECOVERABLE FORMAT ERROR ON SCREEN DISPLAY

Meaning

Error switch: not set.

EDT5354 STRUCTURE SYMBOL '(&00)' NOT FOUND

Meaning

The structure symbol (&00) does not exist in the specified line.
No positioning has been performed.
(&00): structure symbol.

Response

Correct and re-enter the statement.

EDT5356 K-LINE NOT COPIED BECAUSE OF TERMINAL CONTROL CHARACTERS

Meaning

K-line cannot be copied to the statement line because it contains screen control characters.
Error switch: not set.

EDT5357 LINE DOES NOT EXIST

Meaning

The line specified in format 2 of the @CODE statement by <lineno> does not exist.
Error switch: EDT.

EDT5358 LINE SHORTER THAN 256 BYTES

Meaning

The line specified in format 1 of the @CODE statement by <lineno> is shorter than 256 bytes.
Error switch: EDT.

EDT5359 MAXIMUM LINE NUMBER. COPY INCOMPLETE

Meaning

Processing of the COPY statement is aborted because the maximum permissible line number has been exceeded.
(See also error message: EDT5252 MAXIMUM LINE NUMBER.)
Error switch: EDT.

EDT5360 NO COPY. BUFFER EMPTY

Meaning

The copy buffer is empty, therefore A/B/O cannot be processed.
Error switch: not set.

EDT5362 <TEXT> SPECIFICATION ILLEGAL IN CURRENT STATEMENT

EDT5364 NO INSERT: MAXIMUM LINE NUMBER

Meaning

New data lines cannot be inserted by means of a statement or a statement code because this would cause the maximum permissible number of lines to be exceeded. (See also error message: EDT5252 MAXIMUM LINE NUMBER.)
Error switch: EDT.

EDT5365 NO INSERT: RENUMBERING INHIBITED

Meaning

The required lines cannot be inserted without renumbering the existing lines. Renumbering is, however, inhibited as the operand RENUMBER=NO has been specified in a @PAR statement.

Error switch: EDT.

EDT5366 NO P-KEYS ON THIS TERMINAL

Meaning

The @P-KEYS statement has been called on an 8161 or 3270 Data Display Terminal.

Error switch: not set.

EDT5368 SECOND STATEMENT LINE NOT EMPTY

Meaning

SPLIT OFF or @PAR with operand SPLIT=OFF has been specified in the first statement line of the screen, even though the second statement line contains a statement.

Error switch: not set.

EDT5371 TARGET FILE IS CURRENT WORK FILE

Meaning

The statement has not been processed because the target file is identical with the current work file.

EDT5372 ENTRY DOES NOT EXIST IN SPECIFIED LIBRARY OR TASKLIB

Meaning

The specified entry does not exist and could therefore not be loaded dynamically.

Response

Correct and re-enter the statement, or create the library.

EDT5373 NO MORE THAN 5 'USE' ENTRIES ARE PERMITTED

Meaning

5 is the maximum permissible number of entries that can be specified in a @USE statement.

Response

Delete a USE entry by means of the @USE statement and define a new entry.

EDT5375 NO 'USE' ENTRY DEFINED WITH SPECIFIED SYMBOL

Meaning

The specified USE entry has not been defined and thus cannot be deleted.

Response

Correct and re-enter the statement.

EDT5376 COMMAND BUFFER EMPTY

Meaning

The SHIH command has not been processed because no commands have been stored previously in the command buffer.

EDT5380 SOME JOB VARIABLES NOT ERASED

Meaning

An attempt was made to erase all JVs whose name was specified as partially qualified or contained a specified substring, but some of these JVs could not be erased.

Possible reasons:

- the job variable is only open for read access
- the job variable is protected as a monitoring job variable.

Error switch: EDT.

EDT5400 NOT SUPPORTED ON THIS INTERFACE

Meaning

This form of the statement is not possible at this interface.

Response

Use other form of the statement, e.g.:

- @PAR HEX=ON instead of HEX ON (cf. the @PAR statement);
- the statements @READ, @WRITE, @SET or @SAVE for the processing of DMS files.

EDT5402 ENTER AT LEAST 2 CHARACTERS FOR '@DELETE'

Meaning

In F mode at least two characters (@D or DE) have to be specified if the @DELETE statement is entered without any operands.

EDT5409 STATEMENT ILLEGAL IN THIS ENVIRONMENT

EDT5410 UNDEFINED ERROR IN USER PROGRAM

Meaning

EDT received an undefined return code from an user program.

Response

Correct the user program.

EDT5419 (&00)

Meaning

EDT received an undefined return code from an user program.

(&00): Message returned by the external routine.

EDT5450 NATIONAL EDT: INTERNAL ERROR. RETURNCODE = X'(&00)' AT CCS (&01)
(B) Routing code: * Weight: 99

Meaning

Processing of the user defined CCS (&01) was aborted on a national terminal due to an internal error.

(&00): Error code.

Response

Contact the system support service.

EDT5500 STATEMENT PROCESSING INTERRUPTED BY /INTR
(B) Routing code: * Weight: 99

Meaning

A statement sequence was to be processed in F-Mode dialog. The processing was interrupted by the user by means of a /SEND-MESSAGE TO=PROGRAM or /INTR command. Erros switch: not set.

Response

The rest of the statement sequence, which has not been processed, is output to the command line.

EDT5991 RUNTIME ERROR IN EXTERNAL STATEMENT
(B) Routing code: * Weight: 99

Meaning

The external routine reports a runtime error in the execution of the specified statement.

EDT5999 (&00)
(B) Routing code: * Weight: 99

Meaning

The external routine reports a runtime error in the execution of the specified statement. (&00): Message returned by the external routine.

EDT8000 EDT TERMINATED

Meaning

EDT termination message in the case of a normal program termination.

EDT8001 EDT TERMINATED ABNORMALLY

Meaning

EDT termination message in the case of an abnormal program termination (program error).

Response

Contact the system administrator.

EDT8002 (&00) TO EDT UNSUCCESSFULLY. RETURNCODE = X'(&01)'

Meaning

An error has occurred during the dynamic loading of EDT. The macro (&00) rejected the loading with returncode (&01).

Error switch: not set.

EDT8003 NO VIRTUAL MEMORY AVAILABLE

Response

Release virtual memory.

EDT8005 ERROR ON EDT INITIALIZATION

Meaning

Error switch: not set.

EDT8006 ERROR ON INSTALLATION OF EDT

Meaning

The main module EDTF in the linkage library of EDT cannot be called because it has an invalid version number.

Response

Check and correct the installation of EDT.

EDT8100 EDT INTERRUPTED BY USER
(B) Routing code: * Weight: 99

Meaning

This message is issued for information in SDF-P procedures.

EDT is loaded, but has been interrupted by @SYSTEM (without Operand) or by an explicit K2.

Error switch not set.

EDT8101 USER TERMINATED EDT ABNORMALLY
(B) Routing code: * Weight: 99

Meaning

The user terminated EDT by the statement '@HALT ABNORMAL'.

EDT8200 STXIT ROUTINE FOR RUNOUT ACTIVATED
(B) Routing code: * Weight: 99

Meaning

The end of the program run time has been reached, therefore EDT is terminated.

EDT8292 UNRECOVERABLE RDATA ERROR. PROGRAM ABORTED

Response

Contact the system administrator.

EDT8300 INTERNAL EDT ERROR '(&00)'

Meaning

EDT program error.

Response

Contact the system administrator.

EDT8900 NO VIRTUAL ADDRESS SPACE AVAILABLE

Meaning

During loading, EDT requests 4 pages in the virtual address space for data and variables by means of REQM. If REQM encounters an error, this message is displayed and EDT is terminated. If EDT is called as a subroutine, return code X'10' is supplied right-justified in register 15.

EDT8901 ERROR RECOVERY FAILED. EDT ABORTED

Meaning

The interrupt error recovery after a data error could not be completed successfully. Error switch: not set.

EDT8902 '@HALT' STATEMENT PROCESSED

Meaning

A data error or an unrecoverable error occurred in an EDT batch job. Error switch: EDT.

EDT8910 EDT INTERRUPTED AT LOCATION '(&00)', INTERRUPT WEIGHT=(&01)
(B) Routing code: * Weight: 99

Meaning

The program interruption was caused by the event "program check" or "unrecoverable program error" at location (&00). Detailed information about the error cause is given by the interrupt weight (&01).

Error switch: not set.

Response

Contact the system administrator.

8 Installation notes

This section is intended solely for the system administrator.

Installing EDT in BS2000/OSD V1.0

Installation ID: defluid

If the ACS subsystem is active, it is not necessary to copy the module \$.SYSLNK.EDT.166 to \$EDTLIB.

The contents of the alias catalog in the file SYSACF.EDT.166 must be transferred to the system-global alias catalog. In the alias catalog, the alias name \$EDTLIB is defined for the file name \$.SYSLNK.EDT.166.

Installation of EDT using IMON

As of BS2000/OSD V2.0, EDT supports the installation monitor IMON.

If EDT is installed under a user ID other than Defluid, the file name SYSLNK.EDT.166 in the ACF file must be replaced by the installation name before SYSACF.EDT.166 is added to the global alias catalog and the ACS subsystem is activated.

The REP file SYSREP.EDT.166 must be shareable (SHARE), otherwise the corrections will not be included when dynamically loading EDT.

For performance reasons EDT should be loaded as a subsystem. During activation, care should be paid to the order in which the subsystems are started, i.e. the EDTCON subsystem must be started before the EDT subsystem.

8.1 Product components

File name	Function	BS2000 version
EDT	Phase (load module) for EDT	as of OSD V1.0
SYSLNK.EDT.166	EDT module library	as of OSD V1.0
SYSLIB.EDT.166	User macro library	as of OSD V1.0
SYSTEMS.EDT.166	System message file (MSGMAKER)	as of OSD V1.0
SYSSSC.EDT.166.110	Subsystem declarations for SSCM V1.0	OSD V1.0
SYSSSC.EDT.166.120	Subsystem declarations for SSCM V2.0	as of OSD V2.0
SYSSII.EDT.166	Structure and installation information	as of OSD V2.0
SYSRMS.EDT.166	Correction depot for RMS	as of OSD V1.0
SYSNRF.EDT.166	NOREF file	as of OSD V1.0
SYSREP.EDT.166	REP file	as of OSD V1.0
SYSFGM.EDT.166.D	Release notice (German)	as of OSD V1.0
SYSFGM.EDT.166.E	Release notice (English)	as of OSD V1.0
SYSSDF.EDT.166	Syntax file for SDF (START-EDT command)	as of OSD V2.0
SYSACF.EDT.166	ALIAS catalog	as of OSD V1.0
SINPRC.EDT.166	Procedure for installing a private version	as of OSD V1.0

SYSLNK.EDT.166

The module library SYSLNK.EDT.166 contains the following modules:

Module	Function
EDTSTRT	Driver for START-EDT
EDT	Main module for EDT
EDXCODIR	Dynamically loadable module for CODE statement
EDXPKEY	Dynamically loadable module for PKEY statement
EDXUFS	Dynamically loadable module for POSIX accesses
IEDTGLE	Linkage module for EDT as a subroutine
CODTAB	Code table for CODE statement
EDTSSLNK	DSSM dynamically loadable module
EDTCON	Linkage module for EDT
EDCNATA	Dynamically loadable module for national application
EDTILCS	Dynamically loadable module for runtime system

The module EDTSSLNK is present in loadable form in SYSLNK.EDT.166 and has the following structure:

```
EDTSSLNK  START  0,PUBLIC,READ
          DC      V(EDT)
          DC      V(EDTF)
          DC      V(EDTC)
          DC      V(EDTXS)
          DC      V(EDTFXS)
          DC      V(EDTCXS)
          DC      V(EDTGLE)
          END
```

SYSSII.EDT.166

This file contains the structure and installation information. The product components are assigned logical names (logical IDs) which can be used in IMON to specify the location for installation.

IMON is not supported by EDT prior to BS2000/OSD V2.0.

Release Item	Logical ID
EDT	SYSPRG
SYSLNK.EDT.166	SYSLNK
SYSLIB.EDT.166	SYSLIB
SYSMES.EDT.166	SYSMES
SYSSSC.EDT.166.xxx	SYSSSC
SYSSII.EDT.166	SYSSII
SYSRMS.EDT.166	SYSRMS
SYSREP.EDT.166	SYSREP
SYSNRF.EDT.166	SYSNRF
SYSFGM.EDT.166.D	SYSFGM.D
SYSFGM.EDT.166.E	SYSFGM.E
SYSSDF.EDT.166	SYSSDF
SINPRC.EDT.166	SINPRC
*DUMMY.EDTSTART	SYSDAT.EDTSTART

SINPRC.EDT.166

SINPRC.EDT.166 contains a procedure which can be used to install a private version of EDT under any desired user ID. A private version should be installed for testing purposes only, and not lead to the coexistence of two versions of EDT.

For more detailed information on the SINPRC.EDT.166 procedure, please see the release notice.

8.2 EDTSTART start procedure

As of BS2000/OSD V2.0, IMON can be used to install a start procedure valid under all user IDs for each EDT version capable of co-existence. The administrator can freely select the location at which the procedure file is to be installed.

The logical ID SYSDAT.EDTSTART is defined for the procedure file in the SYSSII file.

The installation file name is made known to the installation monitor by means of the SET-INSTALLATION-PATH command. EDT retrieves this information with the IMON function GETINSP, and the path defined is used in place of \$EDTSTART.

If no file is assigned to the logical ID SYSDAT.EDTSTART, EDT uses the start-up procedure \$EDTSTART.

8.3 EDT as a subsystem

EDT consists of two subsystems:

- EDT and
- EDTCON.

The EDT subsystem (consisting of the EDT module) can be loaded in the upper address space.

The EDTCON subsystem (consisting of the modules EDTCON, IEDTGLE and EDTSSLNK) is loaded in the lower address space.

If EDT is to execute as either a main program or a subroutine in 24-bit address mode, the EDTCON module either creates a link to an EDT loaded in the lower address space or loads a private EDT dynamically in the lower address space.

As of BS2000/OSD V2.0, EDTCON has to be started first with START-SUBSYSTEM SUBSYSTEM-NAME=EDTCON, SYNCHRONOUS=*YES. Then EDT can be started with START-SUBSYSTEM SUBSYSTEM-NAME=EDT.

With earlier versions of the operating system, the order in which the components are started makes no difference.

8.4 Installation notes for the module CODTAB

The module CODTAB contains the following code table:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0					␣	&	-									0
1							/		a	j			A	J		1
2									b	k	s		B	K	S	2
3									c	l	t		C	L	T	3
4									d	m	u		D	M	U	4
5									e	n	v		E	N	V	5
6									f	o	w		F	O	W	6
7								~	g	p	x		G	P	X	7
8									h	q	y		H	Q	Y	8
9									i	r	z		I	R	Z	9
A					'	!	^	:								
B					.	\$,	#	[{					
C					<	*	%	@	\							
D					()	_	']		}					
E					+	;	>	=								
F							?	"								

When a message is output on the terminal, this code table converts the print codes for ä, ö, ü, Ä, Ö, Ü and ß into codes for the corresponding characters on the keyboards of the most commonly used data display terminals. The input table which permits conversion of these keyboard characters back to the print codes for ä, ö, ü, Ä, Ö, Ü, ß during message input is derived from this output table.

It is assumed that the following codes for the umlauts and the β character are valid for the keyboard and the printer:

Keyboard	Character	File
X'FB'	ä {	X'AB'
X'4F'	ö	X'AC'
X'FD'	ü }	X'AD'
X'BB'	Ä [X'8B'
X'BC'	Ö \	X'8C'
X'BD'	Ü]	X'8D'
X'FF'	β -	X'67'

Individual characters can be added to this table or modified as follows:

In this example, parentheses are to be changed to square brackets on the printer. It is assumed that “left parenthesis” has the code X'4D' on the keyboard, and the “right parenthesis” has the code X'5D'. The character “square brackets left” has code X'63' on the printer and the character “square brackets right” has code X'64'.

The table can now be modified using the UPDR statement of LMSCONV:

```
*COR 63, X'4D'
*COR 64, X'5D'
*COR 4D, X'07'
*COR 5D, X'07'
```

or using LMS:

```
/START=LMS
//MODIFY-ELEMENT ELEM=*LIB(ÖIB=$EDTLIB,ELEM=CODTAB,TYPE=R),WRITE-MODE=*ANY
//ADD-REP-RECORD ADD=X'63',NEW-CONTENT=X'4D'
//ADD-REP-RECORD ADD=X'64',NEW-CONTENT=X'5D'
//ADD-REP-RECORD ADD=X'4D',NEW-CONTENT=X'07'
//ADD-REP-RECORD ADD=X'5D',NEW-CONTENT=X'07'
//END-MODIFY
END
```

In order to avoid multiple assignments, the user must fill positions X'4D' and X'5D' of the code table with smudges (X'07'). The table is not checked for multiple assignments.

For major modifications to the code table, it may be better for the user to generate his/her own code table and to replace the module CODTAB in the library SYSLNK.EDT.166 by his/her own module with the same name. A source must therefore be created and translated by the assembler. The resulting object module can then be placed in the library SYSLNK.EDT.166 with the aid of the LMS program.

Example of the source program

```

CODTAB CSECT
CODTAB AMODE ANY
CODTAB RMODE ANY
      TITLE 'EDTF.V2 *** CODTAB *** '
      SPACE
*MODULE CONTAINS ONLY CONVERSION TABLES
*
*KEYBOARD * CHAR.   * FILE
*****
* X'FB'   * (ä)   { * X'AB'
* X'4F'   * (ö)   | * X'AC'
* X'FD'   * (ü)   } * X'AD'
* X'BB'   * (Ä)   [ * X'8B'
* X'BC'   * (Ö)   \ * X'8C'
* X'BD'   * (Û)   ] * X'8D'
* X'FF'   * (ß)   - * X'67'
*
*****
      EJECT
      SPACE
*   CONVERSION TABLE FOR "CODE MODE" - INPUT PROCESSING
*   CONVERSION TABLE FOR "LOWER ON " MODE
*   THE CONVERSION TABLE FOR LOWER OFF IS DERIVED FROM
*   THIS TABLE.
CODTAB1 DC   X'00070707070707070707070707070707'
          DC   X'07070707070707070707070707070707'
          DC   X'07070707070707070707070707070707'
          DC   X'07070707070707070707070707070707'
          DC   X'4007070707070707070707074A4B4C4D4E07'
          DC   X'5007070707070707070707075A5B5C5D5E5F'
          DC   X'6061070707070707070707076A6B6C6D6E6F'
          DC   X'0707070707070707070707077A7B7C7D7E7F'
          DC   X'0781828384858687888907BBBCBD0707'
          DC   X'07919293949596979899070707070707'
          DC   X'0707A2A3A4A5A6A7A8A907FB4FFD0707'
          DC   X'0707070707070707070707070707070707'
          DC   X'07C1C2C3C4C5C6C7C8C9070707070707'
          DC   X'07D1D2D3D4D5D6D7D8D9070707070707'
          DC   X'0707E2E3E4E5E6E7E8E9070707070707'
          DC   X'F0F1F2F3F4F5F6F7F8F9070707070707'

      SPACE 3
      END   CODTAB

```

Related publications

- [1] **EDT V16.6** (BS2000/OSD)
Subroutine Interface
User Guide
- [2] **EDT V16.6A** (BS2000/OSD)
Statement Formats
Ready Reference
- [3] **EDT-ARA** (BS2000/OSD)
Additional Information for Arabic
User Guide
- [4] **EDT-FAR** (BS2000/OSD)
Additional Information for Farsi
User Guide
- [5] **SDF V4.0A** (BS2000/OSD)
Introductory Guide to the SDF Dialog Interface
User Guide
- [6] **BS2000/OSD-BC V2.0A**
Commands, Volume 1, A-L
User Guide
- [7] **BS2000/OSD-BC V2.0**
Commands, Volume 2, M-SG
User Guide
- [8] **BS2000/OSD-BC V2.0A**
Executive Macros
User Guide
- [9] **Assembler** (BS2000)
Reference Manual
- [10] **ASSEMBH** (BS2000)
User Guide

Related publications

- [11] **XHCS V1.0**
(BS2000/OSD)
8-Bit Code Processing in BS2000/OSD
User Guide

- [12] **JV V11.2A** (BS2000/OSD)
Job Variables
User Guide

- [13] **SDF-P V2.0A** (BS2000/OSD)
Programming in the Command Language
User Guide

- [14] **LMS** (BS2000)
SDF Format
User Guide

- [15] **POSIX (BS2000/OSD)**
POSIX Basics for Users and System Administrators
User Guide

- [16] **POSIX (BS2000/OSD)**
Commands
User Guide

Index

- statement [120](#)
- statement [120](#)
- statement code [113](#)

- " within a character string [44](#)
- * statement code [87](#)
- + statement [120](#)
- + statement code [113](#)
- ++ statement [120](#)
- < statement [122](#)
- << statement [122](#)
- > statement [122](#)
- @ statement [208](#)
- @- statement [207](#)
- @+ statement [206](#)
- @AUTOSAVE statement [209](#)
- @BLOCK statement [211](#)
- @CHECK statement [213](#)
- @CLOSE statement [214](#)
- @CODE statement [216](#)
- @CODENAME statement [223](#)
- @COLUMN statement [224](#)
- @COMPARE statement [226](#)
- @CONTINUE statement [238](#)
- @CONVERT statement [239](#)
- @COPY statement [240](#)
- @CREATE statement [248](#)
- @DELETE statement [253](#)
- @DELIMIT statement [258](#)
- @DIALOG statement [259](#)
- @DO procedure in F mode [147](#)
- @DO procedure in L mode [149](#)
- @DO procedures [147](#)
- @DO statement [262](#)

- @DROP statement [271](#)
- @EDIT statement [273](#)
- @ELIM statement [275](#)
- @END statement [277](#)
- @ERAJV statement [279](#)
- @EXEC statement [280](#)
- @FILE entry
 - global [282](#)
 - local [282](#)
- @FILE statement [282](#)
- @FSTAT statement [284](#)
- @GET statement [287](#)
- @GETJV statement [289](#)
- @GETLIST statement [291](#)
- @GETVAR statement [293](#)
- @GOTO statement [294](#)
- @HALT statement [295](#)
- @IF statement [297](#)
- @INPUT procedures [149](#)
 - nested [150](#)
 - start [317](#)
- @INPUT statement [313](#)
- @LIMITS statement [321](#)
- @LIST statement [322](#)
- @LOAD statement [325](#)
- @LOG statement [327](#)
- @LOWER statement [328](#)
- @MOVE statement [329](#)
- @NOTE statement [333](#)
- @ON statement [334](#)
- @OPEN statement [376](#)
- @PAGE statement [385](#)
- @PAR statement [386](#)
- @PARAMS statement [396](#)
- @P-KEYS statement [384](#)

- @PREFIX statement 402
- @PRINT statement 404
- @PROC statement 408
- @QUOTE statement 413
- @RANGE statement 414
- @READ statement 415
- @RENUMBER statement 420
- @RESET statement 422
- @RETURN statement 423
- @RUN statement 426
- @SAVE statement 427
- @SDFTEST statement 430
- @SEARCH-OPTION statement 433
- @SEPARATE statement 434
- @SEQUENCE statement 437
 - format 2 438
- @SET statement 442, 469
- @SETF statement 471
- @SETJV statement 473
- @SETLIST statement 474
- @SETSW statement 476
- @SETVAR statement 478
- @SHOW statement 479
- @SORT statement 485
- @STAJV statement 487
- @STATUS statement 490
- @SUFFIX statement 494
- @SYMBOLS statement 496
- @SYNTAX statement 498
- @SYSTEM statement 500
- @TABS statement 502
- @TMODE statement 506
- @UNLOAD statement 507
- @UNSAVE statement 508
- @UPDATE statement 509
- @USE statement 514
- @VDT statement 516
- @VTCSET statement 517
- @WRITE statement 518
- @XCOPY statement 524, 526
- @XWRITE statement 528
- @ZERO-RECORDS statement 530
- 3270 DDT
 - special features 74
- 4-digit year specification 285
 - in catalog information 481, 488
- 7-bit code
 - XHCS 65
- 7-bit mode
 - XHCS 64
- A**
- A, B, O
 - statement codes 88
- aborting procedures 423
- access to POSIX 54
- actual parameters 158
- AFG key 25
- appending
 - strings 494
- ASCII code 56
- assigning
 - file link names 49
 - line numbers 417, 475
- asterisk 335
 - define 496
- automatic numbering 88
- Autosave 209
- AUTOSAVE statement 209
- B**
- base type 60
- batch mode 69, 138
- binary characters
 - enter 320
- binary operand 164
- blank lines
 - delete 224
- blanks
 - delete 224
- blanks in the statement line 79
- block mode
 - set 211
 - switch on 211
- BLOCK statement 211

- branch
 - address 238
 - conditional 155, 299
 - unconditional 155, 294
- branch destination
 - define 238
- branching
 - procedures 197
- buffer size
 - display 491
- C**
- C statement code 90
- calling
 - a procedure 146
 - EDT 21, 33
 - EDT as a main program 33
 - EDT as a subroutine 36
 - subroutines 426
- catalog entry
 - delete 508
- catalog information
 - 4-digit year specification 285, 481, 488
 - query 284
- cataloging
 - job variables 201, 473
- CCS 63
 - explicit switchover 65
 - implicit switchover 65
 - select 223
 - switch 65, 223
- CCSN 63
- chaining two records together 95
- changing
 - character 25
 - current increment 203
 - current line number 203
- char operand 164
- character
 - change 25
 - convert 216
 - correct 25
 - delete 25
 - insert 25, 93
 - nondisplayable 78
 - remove 25
- character for line break 434
- character set
 - coded 63
 - explicit switchover 65
 - extended 63
 - implicit switchover 65
 - switch 65
- character string
 - insert 224
- character strings
 - query 297
 - read in 251
- chars operand 164
- checking lines 213
- cl operand 164
- clearing copy buffer 87
- CLOSE statement 214
- closing a file 26
- closing a library element 26, 214
- clrng operand 164
- code 63
- code conversion 218
 - activate and deactivate 220
- CODE statement 216
- code table 216, 219
 - display 219
- coded character set 63
- coded character set name 63
- CODENAME statement 223
- CODTAB module 218, 598
- col operand 164
- column counter 390
 - display 131
- column ranges
 - delete 253
- COLUMN statement 224
- command return code 39
- comment operand 164
- comments 333
- COMPARE statement 226

- comparison
 - integer variables 300
 - line contents 299
 - line numbers 300
 - results 234
 - string variables 299
 - work files 194, 226, 234
- conditional branches 155
- constants
 - query 490
- contents of integer variables
 - display 492
- contents of line number variables
 - display 492
- continuation line 79
- continuation of statement line 79
- controlling
 - screen output 516
- conventions
 - ISAM and SAM files 49
 - POSIX file names 55
- conversion
 - uppercase/lowercase 239
- conversion tables
 - standard EDT 65
- converting
 - characters 216
 - line number into integer 443
- copy buffer 90, 103
 - clear 87
- copy facilities 240
- COPY statement 240
- copying 240
 - a line 240
 - cataloged file 240
 - files 244
 - into another line range 240
 - into the statement line 96
 - into the work file 240
 - library element 244
 - line number assignment 88
 - line range 240
 - lines 192
 - lines containing search string 358
 - marked lines 355
 - marked records 355
 - program library element 240
- copying and deleting marked lines 98
- copying options 244
- correcting a character 25
- correcting data 25
- corrections 25
- CPU time
 - display 490
- CREATE statement 248
- creating
 - data records 509
 - files 24, 376
 - ISAM files 427
 - library elements 24, 376
 - lines 248
 - procedures 145
 - texts 191
 - work files 24
- current line range symbol 46
- cursor
 - position 24, 25
- D**
- D statement code 92, 118
- data
 - correct 25
 - enter 24, 44
 - input 24
- data of any format
 - element type X 60
- data records
 - create 509
 - display completely 388
 - modify 509
 - with more than 80 characters 126
- data window 22, 76
 - bright 25
 - handling null characters 77
 - line 76
 - position 120
 - scroll 120

- send 26
- set maximum input length 394
- set to overwrite 25, 83
- statement in 119
- statement line 78
- date 442
 - place in a line 466
 - place in variable 466
 - query 466
- dd operand 164
- declaring
 - S variables 478
- decrementing
 - line number 207
- default code table 217
- default numbering 88
- default parameters
 - enter 386
- defaults
 - display 490
 - element type 393
 - file name 282
 - library name 393
 - query 491
- defining
 - increment value 393
 - input length 394
 - input mode 313
 - line ranges 46
 - procedures 408
 - record separator character 392
 - statement symbol 208
 - structure symbol 394
- DELETE statement 253
- deleting
 - after search string 371
 - blank lines 224
 - blanks at end of line 224
 - column ranges 253
 - files 253, 508
 - ISAM files 275
 - job variables 201, 279
 - library elements 253
 - line containing search string 373
 - line ranges 253
 - lines 26, 193
 - record marks 83, 118, 136, 193, 253
 - records 92
 - S list variable 474
 - search string 368, 371
 - texts 193
 - work files 193, 253, 271
- DELIMIT statement 258
- delimiter symbol
 - define 413
- delimiters
 - search string 337
- delta elements 61
- description of element types 60
- description of syntax 161
- destination 88
- dialog
 - terminate 295
- DIALOG statement 259
- difference between two lines 100
- directory
 - of library 59
- directory of a library
 - display 479
- directory of a user catalog
 - display 479
- disk file 20
- display
 - hexadecimal 128
- Display statement buffer
 - statement 133
- displaying
 - column counter 131
 - contents of integer variables 492
 - contents of line number variables 492
 - files 284
 - information on work files 408
 - starting column of search string 346
 - two work windows 134
- DMS error switch 297
 - reset 422
- DO statement 262
- domain operand 164

- DROP statement 271
- DUE key 26
- duplicating
 - line ranges 241
- E**
- E statement code 93
- EBCDIC code of a string 443
- EDIT FULL
 - @PAR 389
- EDIT LONG
 - @PAR 388
- EDIT LONG mode 76
- EDIT LONG statement 126
- EDIT statement 273
- edited data
 - element type P 60
- EDT
 - call 21, 33
 - call as a main program 33
 - call as a subroutine 36
- EDT constants 490
- EDT error switch 297
 - reset 422
- EDT execution
 - in batch mode 274
 - on command procedures 274
- EDT input sources 141
- EDT management 185
- EDT mode 490
- EDT session
 - initialize a string variable 35
 - interrupt 37
 - terminate 37, 280, 295
- EDT statements 44
- EDTISAM 49
- EDTMAIN 378
- EDTSAM 49
- edtsymb operand 164
- EFG key 25
- element 58
- element designation in program libraries 59
- ELEMENT TYPE
 - @PAR 393
- element type 60
 - C, load modules 60
 - D, text data 60
 - F 60
 - H 60
 - J, procedures 60
 - L 60
 - M, macros 60
 - P, list elements 60
 - R, object modules 60
 - S, source programs 60
 - U 60
 - X, data of any format 60
- elemname operand 164
- elemtyp operand 164
- ELIM statement 275
- END statement 277
- end-of-record character 76
- entering data 24
- entry operand 164
- entry point (ENTRY) 426
- ERAJV statement 279
- error switches
 - reset 422
- ESCAPE function 37
- escape symbol
 - define 514
- EXEC statement 280
- executing
 - procedures 198, 262
- execution mode 498
- execution of an EDT procedure 146
- explicit character set switchover 65
- extended character set 63
- Extended Host Code Support 63
- extending
 - S list variable 474
- external loops 156
- external statement routines 514

F

F keys 83
F mode 73
 procedures in 145
 record marks 136
 statement codes 85
 statements 137
 switch to 259, 273
F1 key 83
F2 key 25, 83
F3 key 83
file link name
 assign 49
 EDTISAM 49
 EDTSAM 49
file name
 declare 282
 preset 282
 query defined 491
file operand 165
file processing
 virtual 380
 with search string 334
FILE statement 282
file version number 275, 282, 287, 313, 376, 427,
 508, 518
files
 close 26
 compare 226, 234
 copy 241, 244, 376
 create 24, 376, 517, 518
 delete 253, 275, 508
 generate 24
 open 376, 380, 415
 process 187
 read 287, 376, 380, 415
 real processing 378
 save 427, 517, 518
 update 25
filler characters
 data window 77
 define 496
form feed 385
formal operand 165

formal parameters 158
fraction operand 165
freely selectable string
 asterisk 335
freetyp operand 165
FSTAT statement 284
FULL-SCREEN mode 273
function keys 83
fwkfnr operand 165
fwkfnr statement 125
fwkfv operand 165

G

general statement format 45, 161
GET statement 287
GETJV statement 289
GETLIST statement 291
GETVAR statement 293
global @FILE entry 282
global file names
 display 491

H

HALT statement 295
handling
 line numbers 189
hardware tabulator 502
hd operand 165
HEX
 @PAR 388
hex operand 165
HEX statement 128
hexadecimal
 display 128
 mode 388
hexadecimal characters
 enter 320
hexadecimal code 128
hexadecimal mode 128
 switch off 128
 switch on 128
hit line
 record 341

- hits
 - query 341
 - record 341
- hpos operand 165
- hpos-op operand 165
- I**
- implicit character set switchover 65
- inc operand 165
- INCREMENT
 - @PAR 393
- increment 421
 - change 203
 - current 205
 - effect on stack entry 205
- increment size 421
 - define 469
- increment value
 - define 393
 - set 203
 - store 203
- incrementing
 - line number 206
- INDEX
 - @PAR 390
- INDEX statement 130
- indirect specification
 - operands 45
 - search string 336
- INFORMATION
 - @PAR 390
- information
 - job variable 487
 - output 194
- information line 390
- initializing a string variable 35
- input 42
 - data 24
 - define length of 394
 - format 45
 - in a procedure 145
 - in data window 22
 - in L mode 138
 - in mark column 22
 - in statement line 22
 - length 44, 79
 - mode 313
 - of statements 45
- INPUT file 317
- INPUT statement 313
- insert area 100
- insert mode 26
- inserting
 - after search string 364
 - before search string 364
 - characters 25, 93
 - lines 26, 100
 - texts 191, 224
- installation notes 593
- int operand 165
- integer variables 143
 - assign values to 442, 443
 - convert to strings 449
 - display values of 492
 - output values of 492
- integers
 - convert into line numbers 457
- interactive mode 69, 138, 259
- internal loops 156
- internal representation of a string 457
- interrupting
 - an EDT session 37
 - EDT 197
- int-var operand 165
- ISAM files
 - conventions 49
 - create 376, 427
 - delete 275
 - process 376
 - read 25, 287
 - real processing 376, 378
 - store 24, 427
 - with fixed record length 50
 - write 427
- ISAM key 50
- ISO4 operand 481, 488

J

J statement code 95
job variables 67, 144, 201
 assign to a character string 289
 assign values to 473
 catalog 201, 473
 delete 201, 279
 delete entries 279
 display information on 487
 display on screen 289
 link name 473
 output 201
 output information on 487
 partially qualified name 279
 read 201
 read values 289
 write information to work file 487
 write to a work file 289

JV 67

K

K keys 84
K statement code 96
K1 key 84
K2 key 84
K3 key 84
key length 50
keyword parameters 158, 262, 396

L

L mode 138
 input in 138
 set 69
 statements in 139
 switch to 138
L mode statement symbol 44
L statement code 97
I statement code 100
last statement
 repeat 78
letters
 convert 328
LIBRARY
 @PAR 393

library 58
 define default name 393
 element designation 59
 process 59, 188
 structure 59
library directories
 display 479
library elements 58
 close 26
 copy 244
 create 24, 376, 518, 521
 delete 253
 generate 24
 open 376, 380
 process 58, 61
 query defined 492
 read 25, 376, 380
 save 214, 518
 store 24, 521
 update 25
 write to disk, tape 214

LIMIT

 @PAR 394

LIMITS statement 321

line

 break 434
 mark as destination 88
 operand 166

line break 434

line containing search string
 output 342

line feed 323

line length

 determine 443
 display 491
 for a printer terminal 274
 maximum 213
 query 443

line length for printing 69

line number display 22, 76, 390
 suppress 80

 switch on and off 130

line number of first hit 350

- line number variables 144
 - assign line numbers to 457
 - assign values to 442, 463
 - display contents of 492
 - output values of 492
- line numbering
 - automatic 391
- line numbers 245, 318
 - adopt 437
 - assign 88, 417
 - assignment of 475
 - change 203
 - check 321, 437
 - check sequence 439
 - convert into integers 443
 - convert to strings 449
 - decrement 207
 - handling 189
 - increment 206, 248
 - lowest, highest 321
 - output 321
 - renumber 421
 - retain 391
 - set 203, 469
 - starting value 408
 - store 203
 - symbolic 46
 - write in a line 463
- line range symbol 46
 - define 414
- line ranges
 - copy 241
 - define 46
 - delete 253
 - duplicate 241
 - output 404
 - symbolic line numbers 46
 - transfer from 329
- line scale 131
- line-break character 434
- lines
 - check 213
 - copy 90, 98, 192, 241
 - create 248
 - delete 26, 92, 98, 193
 - insert 26, 100
 - modify 116
 - move 192
 - number 437
 - output 194
 - place values in 442
 - renumber 391, 421
 - set to overwritable 116
 - sort 485
 - transfer 241
- lines containing search string
 - copy 358
- link name
 - file 49
 - job variable 473
- linkname operand 166
- list elements
 - element type P 60
- LIST statement 322
- list variable 68
- In operand 166
- In-sym operand 169
- In-var operand 169
- load modules
 - element type C 60
- LOAD statement 325
- load unit 426
- loading
 - program 325
 - subroutines 426
- local @FILE entry
 - explicit 282
 - implicit 282
- log control 327
- LOG statement 327
- logging
 - an EDT procedure 263
 - input in batch mode 327
 - new or updated lines 213
 - procedures 270
- loops
 - external 156
 - internal 156

LOWER

- @PAR 389

- LOWER statement 328

- lowercase letters 328, 389

M

- m operand 169

- M statement code 98

- m statement code 118

- macros

 - element type M 60

- main code 39

- management

 - EDT 185

- managing

 - procedures 198

- mark 353

- mark column 22, 75

- marked lines

 - copy 98, 355

 - delete 98

- marking

 - a line as destination 88

 - lines containing search string 352

- marking for copying 90

 - without clearing copy buffer 103

- memory

 - release 50

 - virtual 20

- memory area 20

- message level 39

- message operand 169

- messages

 - suppress 69

- metasymbols 17

- metasyntax 162

- mode 490

- modifying

 - data records 509

 - lines 116

 - texts 191

 - work windows 80

- modlib operand 169

- module

 - unload 507

- module library 426

- MOVE statement 329

- moving

 - lines 192

 - work windows 105

N

- n operand 169

- n statement code 100

- name operand 169

- negative searches 336

- nested @DO procedures 150

- nested @INPUT procedures 150

- new operating mode 25

- notational conventions 17, 162

- null characters

 - in the data window 77

 - in the statement line 79

- number

 - of current work file 411

 - of displayed work file 79

 - query 297

- number of work file

 - display 492

- number of work files 125

- numbering

 - automatic 88

 - default 88

 - lines 437

 - with defined increment 88

- numbers

 - of used work files 411

O

- object modules 426

 - element type R 60

- ON statement 334

- op operand 170

- OPEN statement 376

- open system 54

- opening
 - files [376, 380](#)
 - library elements [376, 380](#)
 - UFS file [526](#)
- operands
 - general information [45, 161](#)
 - indirect specification of [45](#)
 - overview [164](#)
- operating mode [73, 490](#)
 - EDIT FULL [25, 76](#)
 - F mode [73](#)
 - L mode [138](#)
 - switch [194, 273](#)
- operation
 - general information [45, 161](#)
- OPTIMIZE
 - @PAR [391](#)
- output [42](#)
 - on the screen [404](#)
 - optimize [391](#)
- outputting
 - catalog information [284](#)
 - contents of variables [490](#)
 - data records with more than 80 characters [126](#)
 - files [284](#)
 - information [194](#)
 - job variables [201](#)
 - last statements [124](#)
 - line numbers [321](#)
 - line ranges [404](#)
 - lines [194](#)
 - local file name [492](#)
 - string variables [404](#)
 - to printer [322](#)
 - two work windows [391](#)
 - work file contents [405](#)
- overview of statement codes [87](#)
- overview of statements [185](#)
- overwritable
 - set data window to [25](#)
- overwrite mode [389](#)
- P**
- P keys
 - define [384](#)
 - program [384](#)
- PAGE statement [385](#)
- PAR statement [386](#)
- param operand [170](#)
- parameters
 - actual [158](#)
 - defining [396](#)
 - enter defaults [386](#)
 - formal [158](#)
 - keyword [158, 262, 396](#)
 - positional [158, 262, 396](#)
 - transfer [158](#)
- partially qualified job variable name [279](#)
- passing statements [23](#)
- path operand [170](#)
- pfile operand [170](#)
- P-KEYS statement [384](#)
- placing an integer in a line [463](#)
- positional parameters [158, 262, 396](#)
- positioning
 - horizontally in work file [122](#)
 - in the work file [119, 120](#)
 - the cursor [25](#)
 - to record marks [121](#)
 - to records with record marks [83](#)
 - windows [471](#)
 - work files [188, 189](#)
 - work windows [105](#)
- POSIX [54](#)
- POSIX file system [54](#)
- POSIX files [54](#)
 - open and read [526](#)
 - process [57](#)
 - read [524](#)
 - save [528](#)
- ppath operand [170](#)
- PREFIX statement [402](#)
- prefixing
 - strings [402](#)
- principle of operation [20](#)
- PRINT statement [404](#)

- printable characters
 - enter 320
 - printing 322
 - contents of work file 322
 - procedures 141
 - @DO 147
 - @INPUT 149
 - abort 423, 424
 - branch to 294
 - branch to a line 155
 - branches 155
 - branching in 197
 - call 146
 - comment 333
 - create 145, 408
 - element type J 60
 - execute 146, 198, 262
 - in F mode 145
 - log 270
 - loops 156
 - manage 198
 - multiple execution 156
 - nested 150
 - parameters 158
 - start 262, 313, 317
 - terminate 277, 424
 - within a BS2000 system procedure 153
 - processing
 - files 187
 - libraries 59, 188
 - library elements 61
 - real 378
 - processing sequence
 - in the mark column 85
 - screen 81
 - with split screen 81
 - procno operand 170
 - program
 - load 325
 - start 280
 - unload 507
 - programmable keys 384
 - PROTECTION
 - @PAR 389
 - public space 20
- Q**
- querying
 - a hit 341
 - catalog information 284
 - character strings 297
 - numbers 297
 - switches 297
 - QUOTE statement 413
- R**
- r operand 170
 - R statement code 103
 - range operand 170
 - RANGE statement 414
 - range symbol 46
 - define 414
 - range* operand 170
 - RDATA 69
 - READ statement 415
 - reading
 - character strings 251
 - files 25, 376, 380, 415
 - from SYSDTA with RDATA 69
 - into the work file 25
 - ISAM files 25, 287
 - job variables 201
 - library elements 25, 376, 380
 - S list variables 291
 - S variables 293
 - SAM files 25
 - UFS file 524
 - real processing 378
 - record marks 136
 - delete 118, 136, 193, 253
 - position to 121
 - process 83
 - search for 121
 - set 118, 136
 - record separator 119
 - record separator character
 - define 392

- recording
 - hit line 341
 - hits 341
- records
 - chain together 95
 - copy 90
 - delete 92
 - display complete 126
 - split 119
- rel operand 171
- releasing
 - memory 50
 - storage space 70
- RENUMBER
 - @PAR 391
- RENUMBER statement 420
- renumbering lines 391
- replace character
 - slash 335
- replacing
 - after search string 364
 - before search string 364
 - search string 361
- RESET statement 422
- resetting
 - error switches 422
- restoring screen contents 84
- return
 - conditional 299
 - unconditional 294
- RETURN statement 423
- rng operand 171
- rng* operand 172
- RS key 26
- RUN statement 426
- S**
- S list variable
 - delete 474
 - extend 474
 - read 291
- S statement code 105
- S variables 68, 144
 - assign 293
 - assign values to 478
 - define 478
 - output 293
- SAM files
 - close 518
 - conventions 50
 - read 25, 415
 - real processing 378
 - store 24, 518
 - with fixed record length 50
 - write 518
- SAVE statement 427
- saving
 - ISAM files 427
 - library elements 214
 - UFS files 528
- SCALE 131
 - @PAR 390
- scale 390
 - display 131
- SCALE statement 131
- screen
 - lines 76
 - lines, display number of 491
 - output optimization 391
 - restore contents 84
 - split 80
- screen dialog
 - terminate 295, 423
- screen display
 - uppercase and lowercase 328
- screen display of the 3270 Data Display Terminal 74
- screen output
 - control 516, 517
- screen-oriented 22
- SDF syntax check 430
- SDFTEST statement 430
- search operand 172
- search string 334
 - default value for uppercase/lowercase 433
 - delete 368
 - delete after 371
 - delete before 371

- delimiters 337
- indirect specification 336
- processing with 334
- replace 361
- specify 335
- uppercase/lowercase notation 335
- wildcard 335
- searches
 - display starting column 346
 - mark lines 352
 - output lines containing search string 342
 - record hits 341
- searching 334
 - delete a line 373
 - for record marks 121
 - insert after search string 364
 - insert before search string 364
 - replace after search string 364
 - replace before search string 364
 - with negation 336
- searching for
 - line number of first hit 350
 - strings 334
- selecting
 - CCS 223
 - format of work window 130
- semantics check 85
- sending a data window 26
- SEPARATE statement 434
- SEPARATOR
 - @PAR 392
- SEQUENCE statement 437
- Set empty line mode
 - statement 530
- SET statement 442, 469
- SETF statement 471
- SETJV statement 473
- SETLIST statement 474
- SETSW statement 476
- setting
 - Autosave 209
 - block mode 211
 - execution mode 498
 - L mode 69
 - record marks 83, 118, 136
 - syntax check 498
 - tabs 502
- SETVAR statement 478
- SHIH statement 133
- SHOW statement 479
- slash
 - define 496
 - replace character 335
- smudge character 78, 218
- software tabulator 502
- SORT statement 485
- sorting lines 485
- source programs 60
 - element type S 60
- space
 - public 20
- spec operand 173
- special work files 48
- specifying a search string 335, 336
- SPLIT
 - @PAR 391
- SPLIT statement 134
- splitting
 - records 119
 - the screen 80
 - work windows 80, 391
- stack entry 205
- STAJV statement 487
- START-EDT command 33
- starting
 - @INPUT procedures 317
 - EDT 280
 - procedures 262, 313
 - subroutines 426
- starting column of search string
 - display 346
- statement
 - fwkfv 125

- statement code [23, 75, 85, 90](#)
 - * [87](#)
 - +/- [113](#)
 - A, B, O [88](#)
 - C [90](#)
 - D [92, 118](#)
 - E [93](#)
 - J [95](#)
 - K [96](#)
 - L [97](#)
 - M [98](#)
 - m [118](#)
 - n/l [100](#)
 - overview [87](#)
 - processing sequence [85](#)
 - R [103](#)
 - S [105](#)
 - T [107](#)
 - U [112](#)
 - x [116](#)
- statement delimiter [78](#)
- statement format
 - general [45, 161](#)
- statement line [22, 78](#)
 - continuation [79](#)
 - copy into [96](#)
 - handling of blanks [79](#)
 - handling of null characters [79](#)
 - maximum input length [79](#)
 - statements in the [119](#)
- statement name [45](#)
- statement sequence [78](#)
- statement symbol [44](#)
 - define [208](#)
 - display [491](#)
- statements [23](#)
 - enter [45](#)
 - format [45](#)
 - general information [44](#)
 - in F mode [137](#)
 - in L mode [139](#)
 - in the statement line [119](#)
 - output last [124](#)
 - overview [185](#)
 - pass [23](#)
 - redisplay [124](#)
 - redisplay last [78](#)
 - separate [78](#)
 - status display [22, 79](#)
 - column number [79](#)
 - line number [79](#)
 - number of work file [79](#)
 - STATUS statement [490](#)
 - storage space
 - release [70](#)
 - superfluous [70](#)
 - storing
 - a new work file [24](#)
 - a SAM file [24](#)
 - an ISAM file [24](#)
 - storing a library element [24](#)
 - str operand [173](#)
 - string operand [174](#)
 - string variables [143](#)
 - assign values to [442, 449](#)
 - create [248](#)
 - initialize [35](#)
 - output [404](#)
 - strings
 - append [494](#)
 - delete [368, 371](#)
 - EBCDIC [443](#)
 - insert [364](#)
 - insert as prefix [402](#)
 - internal representation [457](#)
 - replace [364](#)
 - search for [334, 358](#)
 - str-ln operand [175](#)
 - strng operand [176](#)
 - STRUCTURE
 - @PAR [394](#)
 - structure
 - library [59](#)
 - of work window [75](#)
 - structure depth
 - positioning by [114](#)
 - structure symbol [114](#)
 - define [394](#)

structured-name operand 176
str-var operand 177
STXIT routine 37
subcode1 (SC1) 39
subcode2 (SC2) 39
subroutine
 call 426
 call EDT as a 36
 load 426
 start 426
 user program as a 426
subsystem EDT 593
SUFFIX statement 494
suppressing
 line number display 80
 messages 69
switches
 query 297
 reset 422
 set 476
switching
 between L mode and F mode 273
 CCS 223
 character sets 65
 operating mode 194
 to F mode 259
 to L mode 138
 work files 125, 188, 189, 408
switching off
 line number display 130
switching off hexadecimal mode 128
switching on
 line number display 130
switching on hexadecimal mode 128
symbolic line numbers 46
symbols
 define 496
 line numbers 46
 range 46
SYMBOLS statement 496
syntax check 85, 430, 498
syntax description 161, 162
SYNTAX statement 498

syntax test
 by SDF 107
SYSDTA 69
system command
 issue 500
SYSTEM statement 500

T

T statement code 107
tab character
 define 502
tab operand 177
tabs
 set 502
TABS statement 502
target positions 245
target range 245
task
 USERID 506
task information
 display 506
task sequence number 506
 display 490
 query 490
task switch 5 69
task switches 69
 set 476
terminating
 EDR 197
 EDT session 280, 295
 procedures 277
 screen dialog 295, 423
terminating an EDT session 37
text
 create 191
 delete 193
 enter 24
 insert 191, 224
 modify 191
 operand 177
text corrections 25
text data
 element type D 60

text delimiters 258, 337
 define 413
 display 491
text input
 binary characters 320
 hexadecimal characters 320
 printable characters 320
time of day 442
 display 490
 place in a line 466
 place in variable 466
 query 466
TMODE statement 506
transferring
 line ranges 329
 lines 241
TSN 506
 display 490

U

U statement code 112
UFS files
 open and read 526
 read 524
 save 528
umlauts 221
unconditional branches 155
UNLOAD statement 507
unloading
 module 507
 program 507
UNSAVE statement 508
UPDATE statement 509
updating
 files 25
 library elements 25
uppercase letters 328
uppercase/lowercase conversion 239
uppercase/lowercase notation 328
 default value for search 433
USE statement 514
user catalog directories
 display 479

user ID
 display 490
 query 490
USERID of task 506
usersymb operand 177

V

values
 assign to job variables 473
 read from job variables 289
variables 143
 display contents of 490
 integer variables 143
 line number variables 144
 query contents of 491
 string 143
VDT statement 516
ver operand 178
vers operand 179
virtual file processing 380
virtual memory 20
vpos operand 179
vpos-op operand 179
VTCSET statement 517

W

wildcards 335
 define 496
window
 position 471
wkflnr statement 125
wkflvar statement 125
work area 20
work file concept 48
work files 20
 compare 194, 226, 234
 create 24
 current 76
 delete 193, 253, 271
 display information on 408
 display number of 411
 free 411
 number 79, 125
 numbers of 492

- output contents of 405
- position 188, 189, 471
- position horizontally 122
- position in 119, 120
- print contents of 322
- query 411
- save 214
- special 48
- store 24
- switch 125, 188, 189, 408, 471
- used 411
- variable 125
- write to disk or tape 214

work window 20

- data window 22, 76
- description 75
- divide 134
- EDT 21
- empty 22
- general information 75
- line number display 22, 76
- mark column 22, 75
- modify 80
- move 105, 113
- move horizontally 122
- parts of 22
- position 105, 113
- position by structure depth 114
- select format of 130
- split 80, 134, 391
- standard 75
- statement code 75
- status display 22
- structure of 75

working with EDT 21

write protection 389

write protection at record level 389

WRITE statement 518

writing

- ISAM files 427
- library elements 214

X

- x statement code 116
- XCOPY statement 524, 526
- XHCS 63
- xpath operand 179
- XPG4 54
- XWRITE statement 528



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>.

Copyright Fujitsu Technology Solutions, 2009

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.

Copyright Fujitsu Technology Solutions, 2009